Programen espezifikazio, egiaztapen eta eratorpen formala

Javier Alvez Gimenez
Xabier Arregi Iparragirre
Jose Gaintzarain Ibarmia
Paqui Lucio Carrasco
Montse Maritxalar Anglada

Universidad del País Vasco
Euskal Herriko Unibertsitatea
ueu

Programen espezifikazio, egiaztapen eta eratorpen formala

Javier Álvez Gimenez
Xabier Arregi Iparragirre
Jose Gaintzarain Ibarmia
Paqui Lucio Carrasco
Montse Maritxalar Anglada

Udako Euskal Unibertsitatea eta Euskal Herriko Unibertsitatea
Bilbo, 2016

UEUren ISBNa: 978-84-8438-596-7
Lege-gordailua: BI-833-2016

Inprimategia: PRINTHAUS S.L., Bilbo
Azalaren diseinua: Igor Markaida Uriagereka
Hizkuntza-zuzenketen arduraduna: Ander Altuna Gabiola

Banatzaileak: UEU. Erribera 14, 1, D BILBO. Telf. 946790546
Helbide elektronikoa: argitalpenak@ueu.eus
www.ueu.eus

Euskal Herriko Unibertsitatea
Helbide elektronikoa: argitalpenak@ehu.eus
www.ehu.eus/argitalpenak

Elkar Banaketa: Igerabide, 88 DONOSTIA

Arrazoibide logikoak berekin dituen edertasuna, jakingura, erronka eta zehaztasuna hauteman eta aintzat hartzen dituzten pertsona guztiei

Aurkibidea

Irudien zerrenda

Hitzaurrea

Programazioa, edo softwarearen garapena, inolako ondoriorik eragin gabe, jakinaren gainean eta era guztiz hedatuan, akastun produktuak sortu eta saltzen diren jarduera ekonomiko gutxi horietakoa (beharbada bakarra) da: fabrikatzaileak ez dira auzitara eramanak edo isunduak izaten eta kontsumitzaileen elkarteak ere ez dira kexatzen. Software-produktu bat erosten denean, Windows edo MacOS bezalako sistema eragile bat dela, edo salgai diren milaka edo milioika aplikazioetako bat dela, noizbehinka produktuak «huts egitea» normaltzat jotzen du mundu guztiak, hau da, aurreikusita zegoen bezala funtzionatzeari uztea eta ezohiko jokabidea izatera igarotzea (esate baterako, programa ustekabean ixtea eta ordenagailua «esekita» gelditzea). Software horren garapen-akatsek eragindako hutsegite horiek erabiltzaileari eragozpen batzuk sortzea beste ondoriorik ez dute izaten batzuetan. Baina, beste batzuetan, handiagoak edo txikiagoak izan daitezkeen diru-galerak edo giza bizitzen galerak ere ekar ditzakete. Gai honi buruzko hausnarketa txiki bat egiteak nahikoa izan beharko luke bi galdera hauek era naturalean sortzeko: Zergatik gertatzen da hori? Saihestezina al da?

Ia informatikaren hastapenetatik, programazio-erroreen kopurua jaisteko edo errore horien ondorioak arintzeko norabidean etengabeko lana egin da. Hastapenetan, programak zeroz eta batekoz osatutako segida luzeak besterik ez zirenean, erroreak agertzea oso ohikoa zen. Nahi gabe, batekoa idatzi beharreko leku batean zeroa idazteagatik, besterik gabe, aurretik inola ere ezin suma zitezkeen ondorioak eragin zitzakeen errorea gerta zitekeen[1]. Zentzu horretan, goi-mailakoak deitzen diren programazio-lengoaien agerpenak lehenengo aurrerapen handia ekarri zuen: programazio-erroreak sortuko lituzketen akats txiki asko automatikoki detektatuak izan zitezkeen. Gainera,

1. Programazioan errore txikiek ondorio izugarriak eragin ditzakete. Teknologiako beste arlo batzuetan hori ez da ohikoa. Adibidez, tresna elektriko batek behar duen tentsioa gaizki kalkulatu bada, ziur aski nahiko ondo funtzionatuko du kalkulatutakoaren aldean desberdina den tentsio batekin, aldea txikia baldin bada behintzat. Aldiz, programazioan, programa bateko akats batek ondorio oso txarrak izan ditzake, liburu honetako sarrerako kapituluan azaldutako adibideren batean ikus daitekeen bezala.

oinarrizkoenak diren erroreen detekzioa —idaztean nahi gabe egindako erroreak barne— bermatzeko tekniken etengabeko garapena egon da lehenengo programazio-lengoaia horietatik gaurkoetaraino. Hala ere, esan beharra dago egungo programazio-lengoaia denek ez dietela erroreak detektatzeko teknika horiei etekinik ateratzen. C edo C++ bezalako lengoaia batzuek eraginkortasuna lehenesten dute fidagarritasunaren aurretik. Hau da, beharbada C lengoaiaz idatzitako programa bat Javaz idatzita balego baino azkarrago exekutatuko da, baina erroreak izateko aukera gehiago ere izango ditu.

Ondorio bezala, gaur egun, arazoa sortzen duten erroreak ez dira idaztean egindako akatsak bezalako oinarrizko erroreak, errore logikoak deitzen direnak baizik. Garatzaileek egin beharko luketena egiten ez duten programak eraikitzen dituztenean agertzen diren erroreak dira errore logiko horiek. Programak egin behar duena gaizki ulertzeagatik, exekuzio-denboran gerta daitezkeen egoera batzuk kontuan ez hartzeagatik, kasu batzuk tratatzea ahazteagatik edo, besterik gabe, programatzailearen gaitasun ezagatik sor daitezke errore horiek.

Erroreak detektatzeko ohiko soluzioa testing-a da, hau da, programa datu (edo proba-joko) desberdin askorekin exekutatzea, programak erroreren bat baldin badu, errorea detektatzea ahalbidetuko duen proba-jokoren bat exekutatuko dugulako itxaropenarekin. Arazoa da zorterik ez edukitzea gerta dakigukeela eta gure proba-jokoetatik batek ere ez ahalbidetzea erroreren baten detekzioa. Berez, oraindik okerragoa da egoera, errore batzuk ezin dira detektatu testing-aren bidez, ez baitira programazio-erroreak baizik eta gure softwareak ebatzi beharko lukeen problemaren alderdi batzuei buruzko ezjakintasunari lotutako erroreak. Adibidez, 1983an Sobietar Batasuneko alerta-sistemak Estatu Batuetatik 5 misilez osatutako erasoa jaurtiki zutela detektatu zuen. Zorionez, arduradun zegoen ofizialak ez zuen erantzun erasorik abian jarri, amerikarrak erasotzen ari baziren, 5 misilekin bakarrik egitea oso arraroa iruditu baitzitzaion. Programa eraikitzean, ez zen kontuan hartu, baldintza batzuen menpean, eguzkiak lainoetan sortzen duen islak misilen bidezko eraso baten antzekoa den radar-seinalea eragin dezakeela. Errore hori ez litzateke izango testing bidez detektagarria; izan ere, arazoa ez zen programak radar-seinaleak ez zituela aurreikusita bezala prozesatzen, arazoa zen ez zela aurreikusi radar-seinale batzuk engainagarriak izan zitezkeela.

Testing-a beti erabili beharreko oso tresna baliotsua izan arren, ziurtatu nahi badugu software jakin batek ez duela errorerik, funtsean bi gauza egin beharko ditugu:

- Aplikazioak egin behar duena xehetasunez eta era zehatzean espezifikatu edo modelatu behar dugu. Eraiki nahi dugun softwareak nola

funtzionatuko duen eta bere funtzionamendu-baldintzetara egokituko den ala ez era xehatuan aztertzea ahalbidetuko digu espezifikazio zehatzak. Gainera, espezifikazio edo eredu hauek, bezeroari edo erabiltzaileari eraiki nahi den aplikazioa bere beharretara egokitzen den ala ez egiaztatzea ahalbidetuko diote, etxe edo pisu baten planoek etxebizitza hori guk nahi duguna den jakiteko aukera ematen diguten bezala. Eta, oraindik gehiago, aplikazioa eraikitzerakoan software-garatzaileei espezifikazioek gidatzat balioko diete, etxe baten eraikitzaileak eraikuntza horretarako arkitektoak egindako planoak erabiliko dituen bezalaxe.

- Eraikitako softwareak espezifikazioek diotena egiten duela bermatu behar dugu. Hori bi eratara egin daiteke. Lehenengoa, softwarea eraikitzetik zuzena dela bermatzea da. Hori lortuko da, softwarea espezifikazioak transformatuz eraiki bada. Bigarren era, software horrek espezifikazioak betetzen dituela egiaztatzea da (hau da, logikoki frogatzea). Frogapenaren zati handi bat automatizatzen duten software-tresnen laguntzarekin egin ohi da hori gehienetan.

Gaur egun, guztiz fidagarria, hau da, (ia) errorerik gabeko softwarea eraikitzeko behar diren ezagutzak, metodoak eta teknologia badaude. Horren erakusgarri, adibidez, liburu honetako sarrerako kapituluan deskribatzen den Parisko garraio-sareko softwarea da. Beste erakusgarri bat, azken 10 edo 15 urteetan software-akatsengatik istripu larririk ez egon izana da. Horren zergatia da pertsonen bizitza arriskuan ipin dezakeen software kritikoaren kasuan fidagarritasuna bermatzeko beharrezkoa den teknologia guztia erabiltzea. Zergatik, orduan, ez dira teknika horiek aplikatzen kritikoa ez den softwarearen garapenean?

Enpresek ematen duten arrazoi nagusietako bat gaitutako langilerik eza da. Softwarea garatzen dihardutenetatik gutxik dute softwarearen fidagarritasuna bermatzen duten teknikak erabiltzeko, bai logikan, bai espezifikazioan eta bai egiaztapenerako tekniketan beharrezkoa den oinarrizko heziketa. Testuinguru horretan, Informatika Ingeniaritzako ikasketa-planen azken erreformetan, espainiar unibertsitate askok[2] softwarearen garapenaren oinarrian dauden materiak gutxitzea (guztiz kentzea ez denean) aukeratu izana tamalgarria gertatzen da. Zentzu horretan, liburu hau norabide onean doa, espainiar unibertsitate askotako joera tamalgarri horren kontra. Zehatz-mehatz, liburu bikain honek fidagarria den softwarea eraikitzeko erabiltzen diren metodoen atzean dauden programen espezifikaziorako, transformaziorako eta egiaztapenerako oinarrizko tekniketarako sarrera izan daitekeen trebakuntza-ikastaro baterako materiala eskaintzen du.

2. Beste herrialde batzuetako izen handiko unibertsitateetan ez bezala.

Liburu honen egileak orain dela urte askotatik ezagutzen ditut, eta gai hauetan eta hauekin zerikusia duten beste gai batzuetan irakaskuntza- eta ikerkuntza-esperientzia luzea dute. Esperientzia hori inolako zalantzarik gabe gomendatzen dudan liburu honetako testuaren argitasunean eta azalpenen zehaztasunean nabari daiteke.

Bartzelona, 2015eko uztaila Fernando Orejas

1. Sarrera

Kapitulu honetan liburuaren edukiak deskribatuko dira, Software Ingeniaritzan duten garrantzia nabarmenduz. Software Ingeniaritzaren arloari eman zaizkion definizio ugariren artean, IEEE erakundearena[1] da honako hau:

> Softwarearen garapena eta mantentzea ikuspegi sistematiko, diziplinatu eta kuantifikagarri batez egitea, eta ikuspegi horren aplikazioaren azterketa [75].

Horretarako, 1.1. atalean *softwarearen garapenerako metodo formal*en bilakaeran erabakigarriak izan diren faktoreak historikoki kokatuko ditugu. Ikuspegi historiko hori baliagarria zaigu liburu honetan lantzen diren gaiak aurkezteko, eta, bidenabar, gai horien eragina nabarmentzeko. 1.1. atala [76]-n oinarrituta dago partzialki. 1.2. atalean, liburuaren egitura eta edukiak era zehatzagoan deskribatuko dira. Eskerrak emateko ataltxoarekin bukatuko da kapitulua.

1.1. SOFTWAREAREN GARAPENERAKO METODO FORMALAK

Aurrez planteatutako zeregin edo problema baten ebazpena da, funtsean, programa informatikoa. Askotan, jatorrizko zeregin hori emaitza batzuk kalkulatzera edo zerbitzuren bat ematera orientatuta egoten da. Programa informatikoak eraikitzeko jarduera aztertzen duen arloari *programazio* esaten zaio. Hain zuzen ere, programazio-eredu baten oinarriak deskribatzera zuzendua dago liburu hau. Eredu horri hainbat izendapen eta kalifikatzaile jarri izan zaizkio: *programazio metodikoa* edo *programazio zientifikoa* izan dira zabalduenak. Berrikiago *kontratu bidezko programazioa* ere erabili izan da. Esan daiteke ikuspegi honen bilakaera 60ko hamarkadan hasi zela, *programazio egituratua* deitutakoaren eta ALGOL [11] eta Pascal [57] programazio-lengoaien sorrerarekin batera. Programazio egituratuaren oinarrizko ideiak oso ondo daude bilduta Dijkstra-ren [32] maisu-lanean. Ikuspegi egituratu horren funtsezko ideietako bat diseinatutako programen zuzentasuna era sendoan eta

1. Institute of Electrical and Electronics Engineers.

sinesgarrian ziurtatu ahal izateko bidea erraztea zen. Horren haritik iritzikorronte berria sortu zen, programazioa ordura arte aurreikusten zena baino jarduera zientifikoagoa (artisau kutsu gutxiagokoa) zela ulertzearen aldekoa hain zuzen.

Programa bat *zuzena* dela esango dugu, haren diseinua eragin zuen problema ondo ebazten badu, beti (edozein direla ere datuak) esperotako emaitzak lortuz eta inolako errorerik edo ustekabeko ondoriorik sortu gabe. Definizioz, programen ezinbesteko propietatea zuzentasuna da, izaki bizidunena bizia den bezalaxe. Jakina, badira programa guztietan desiragarriak diren eta programak ondo diseinatuta daudela edo ez daudela esatera garamatzaten beste propietate batzuk ere. Propietate horietatik, besteak beste, eraginkortasuna, argitasuna, erabilgarritasuna, moldagarritasuna, sendotasuna eta ondo dokumentatuta egotea azpimarra ditzakegu.

Programa zuzen bat diseinatzea, oro har, zailtasunetatik libre ez dagoen lan neketsua da. Programaren zuzentasuna argi erakutsiko duen dokumentazio edo argudiaketa on bat osatzea ere ez da samurra, batez ere balizko irakurlea konbentzitu nahi bada, eta zuzentasuna bermatzen duten propietateak agerian utzi nahi badira. Azken batean, zuzentasuna argumentatzea zenbat eta errazagoa izan, orduan eta aukera gehiago egongo dira programa aise ulertua izateko, eta, era berean, aise dokumentatua, optimizatua, egokitua edo mantendua izateko.

Horiek horrela izanda ere, programazioaren historia labur samarrean ugariak dira pertsonei nahiz ondasunei kalte konponezinak eragin dizkieten software-erroreen adibideak. Hondamendi horietako batzuk oso ezagunak dira, esate baterako, Ariane 5 suziriarena, eta, beste batzuk ez horrenbeste, Therac-25 erradioterapia-makinarena aipa daiteke kasurako. Lehenengoa 1996an gertatu zen, Ariane 5 suziri europarrak abiatu eta minutu-erdira eztanda egin zuenean. 64 bit-eko zenbakiak 16 bit-eko zenbaki bihurtzean egindako akatsak eragin zuen leherketa, zehazki, bihurketa egiten zuen programari onar zezakeena baino zenbaki handiagoa iritsi zitzaionean. Bigarren hondamendia izatez lehenago gertatu zen Kanadan, 1985 eta 1987 artean, Therac-25 erradioterapia-makinan. Makina horrek 4 pertsonaren heriotza eragin zuen, eta beste batzuei lesio larriak eragin zizkien. Ezbeharraren kausa *race condition* izeneko programazio-akats bat izan zen; errore hori gertatzen da programa kanpo-gertaeren ordena jakin baterako diseinatzen denean, baina gertaeren benetako sekuentzia bestelakoa denean. Aski ezagunak dira, halaber, programazio-errore sinpleen eraginez hondatu izan diren NASAren suziriak. Horien artean aipagarria da 1962an espazioan galdu zen Mariner 1 izenekoa. Artizarrera bidalitako lehenengo espazio-zunda zen. Galera hori ibilbidea kontrolatzen zuen programan, formula edo espresioren

baten transkripzioan, eginiko errore sinpleren batek eragin ote zuen espekulatu izan da. Errakuntza posibleen artean, apostrofo bat (') koma batekin (,) nahastu izana ere aipatu izan da. Beste errore sinple batek eragin zuen 1999an Mars Climate Orbiter zundaren suntsiketa, Marte planetaren atmosferarekin izandako marruskaduraren ondorioz. Berez, zunda horrek ez zukeen inoiz Marteren atmosfera ukitu behar, baina nabigazio-programaren akats batek ibilbidea aldarazi zion; hain zuzen ere, ibilbidea kalkulatzeko, neurriak sistema metriko hamartarrean jasotzea espero zuen programak, baina lurreko kontrol-sistemak eredu anglosaxoiaren araberako neurriak bidali zizkion, hots, miliak, hazbeteak, etab.

Era askotako arrazoiak daude programatzean erroreak egiteko arrisku handia dagoela baieztatzeko. Gainera, akats horiek, itxuraz sinpleak eta hutsalak iruditu arren, ondorio lazgarriak eragin ditzakete. Aurreko adibideetan ikusi den bezala, sistema metriko baten ordez beste bat erabiltzeak, eragiketa aritmetiko bat beste batekin nahasteak, edo objektu baten tamainari muga maximo txikiegia jartzeak, zeharo alda dezakete programaren portaera. Hala, programak egin beharko ez zukeen zerbait egingo du, edo ez du ondo egingo egin beharko zukeena, eta zenbaitetan kalte larriak eragin. Ikusitako adibideak egoera orokor baten isla dira. Programa akastunen ugaritasuna eta errore horiek kodean aurkitzeko zailtasuna dira programazioaren arloari oinarri matematikoa emateko interesa eta ekimena piztu zuten arrazoi nagusiak.

Azken hamarkadetan bi teknika mota sortu dira programazio-erroreei aurrea hartzeko. Alde batetik programak arazteko (alegia, erroreak detektatzeko eta zuzentzeko) teknikak eta tresnak garatu dira. Tresna horiek zenbaitetan automatikoak badira ere, arazte-lan hutsak ezin du errorerik eza bermatu[2]; nork ziurta dezake ez dagoela detektatu gabeko errorerik? Erabat ziur egoteko erarik ez digu ematen arazketak, programaren fidagarritasunean dugun konfiantza areagotzeko balio du bakarrik. Hala ere, nork eta nola arazten duen, horren araberakoa da konfiantza, eta, hori, bistan denez, ez da oso zientifikoa. Programazioari izaera zientifikoagoa emateko asmoz, teknika matematikoagoak edo teknika formalak sortu ziren. Programak diseinatzeko eta analizatzeko dauden teknika formal gehienak *matematika diskretu*an, eta, bereziki, *logika matematiko*an oinarritzen dira. Matematika diskretua da software-ingeniaritzaren euskarri matematiko/zientifikoa, bestelako ingeniaritzen euskarria *kalkulu infinitesimala*, *estatistika* edo *zenbakizko analisia* diren bezalaxe. Edozein ingeniaritzak darabiltza metodo aurredefinituak eta estandarrak bere jardueraren funtsezko parte gisa, eta horietako askok osagai

2. Ederki azaldu zuen hori E. Dijkstra-k [32]-n: «Programa-arazketa akatsak badaudela erakusteko erabil daiteke, baina inoiz ez akatsik ez dagoela erakusteko».

matematiko nabarmena dute.

Programatzaileei (software-ingeniaritzako espezialistei) metodo erabilgarriak eta sendoak eskaintzeko beharrak proposamen ugari sorrarazi ditu, eta horren isla da azken berrogeita hamar urteetako literatura. Programa zuzenen diseinu sistematikorako metodo eta teknika horiek, funtsean, bi fasetan bereizten dute programen diseinua. Lehenengoan, ebatzi beharreko problema era zehatz, doi eta osatu batean espezifikatu behar da. Programak espezifikatzean egoki jaso behar dira egoera onargarri guztiak, eta adierazi behar da, halaber, nolako jokabidea izango den egoera horietan guztietan. Fase horretan egindako akatsak hurrengora, programazio-fasera, igarotzen dira, eta horrek zaildu egiten du programatu beharrekoaren ulerkuntza eta erroreen detekzioa, nahastu egiten baitira modelatze-erroreak eta programazio-erroreak.

60ko hamarkadaren erdialdera arte programek ez zuten hainbesteko pisurik gizartean, eta diseinatzen igarotzen zen denborari edo lortutako produktuaren kalitateari ez zitzaion hainbesteko garrantzirik ematen. Garai horretan nabarmendu zen programa, izatez, esanahi zehatza eta doia duen objektu matematikoa dela. Horrela izanda, programa zehaztasun matematikoz aztertua eta landua izan daiteke haren portaerari buruzko propietateak frogatzeko, bereziki, zuzentasuna frogatzeko. Ildo horretan, 60ko hamarkadaren bukaerarako zenbait metodo eta teknika sortuz programazioari izaera zientifikoagoa emateko xedea zuen ikerlerro garrantzitsu bat bazen (E. W. Dijkstra, R. Floyd eta C. A. R. Hoare ikerlerro horretan zeuden). Berehala jaio zen *programen egiaztapena:* programa batek bere espezifikazioak dioena betetzen duela egiaztatzeko balio duen frogapen matematikoa (logikoa). Bide horren hastapeneko lantzat har daiteke A. Turing-en [89], baina programen egiaztapenaren benetako ernamuinak R. Floyden [40] eta C. A. R. Hoareren [52] dira. Ordutik hona, programazioko metodo formalen arloa etengabeko bilakaeran aritu da. Teknika eta metodo ugari sortu dira, eta programazio-rako oinarri matematiko sendoak landu dira. Egindako aurrerapenek, bai programazioan bai erlazionatutako beste arlo batzuetan (bereziki, software-ingeniaritzarako duen aplikazioan eta *arrazoibide matematikoan*), formalismo berri erabilgarriagoak sorrarazi dituzte, eta beste aplikazio-eremuetarako aukerak zabaldu dituzte. Softwarearen garapen formalerako prozesuaren zenbait atal automatizatzea ere ekarpen garrantzitsua da arloaren bilakaeran.

Metodo formalek programatzaileari laguntzeko helburua dute, eta programa zuzenen diseinua errazteko tresna baliagarri izan nahi dute. Programen diseinurako erabil daitekeen metodologia bakoitzaren egokitasuna, egoera eta testuinguru desberdinen araberakoa da. Programa zuzenak diseinatzean, zein aspekturi aurre egin nahi diogun, horren araberako teknikak erabili behar

dira. Horretarako beharrezkoa da programatzaileak metodo horien ezagutza izatea, eta gai izatea egokiro aplikatzeko, esate baterako, problemak espezifikatzean, parte hartzen duten datu-egiturak espezifikatzean, ebazpen algoritmikoa asmatzean eta formalizatzean, exekutagarria lortzean, planteatutako ebazpenaren zuzentasuna justifikatzean edota diseinatutako programa dokumentatzean. Tekniken eta tresnen ezagutza ezak problemen ebazpen egokia eragozten duela adierazteko, karikatura adierazgarria egin zuen A. H. Maslow-ek [69]-n, horretarako atsotitz ezaguna erabiliz[3]: «Mailua baizik ez baduzu, den-dena iltzea balitz bezala landuko duzu». Programatzaileari tresna hobeak eskaintzeko asmoz, azken hamarkadetan aurrerapen nabarmena gertatu da teknika formalei dagokienez, eta emaitza praktiko batzuk harrigarri samarrak ere izan dira.

Jakina da softwarea garatzeko metodo formalen erabilera eztabaidagarria izan dela, baina badirudi gero eta onartuago dagoela. Aldekoek beti esan izan dute formalismoa onuragarria dela, eta industriak garapen modu zorrotzagoak eta laguntza-tresna egokiak bereganatu beharko lituzkeela. Aurkakoek zailtasun serioak eta intrintsekoak ikusten dizkiete metodo formalei, tamaina handiko proiektu industrialetan erabiliak izateko. Jarrera horren inguruko zenbait mito [50] sortu eta puztu dira, baina mito horiek indarra galdu dute arloak heldutasuna lortu ahala eta industria egokitu ahala. Gaur egun ekarpenak askotarikoak dira, metodoak aplikazio-eremuetara egokitzen ari dira, eta bilakaera sendoa da beste arlo batzuekin lankidetzan. Azken urteotan bereziki nabarmendu dira programazio formalerako laguntza-tresna automatikoak (edo erdi-automatikoak), eta esan daiteke arlo horretan oso ekimen aktiboa eta emankorra nabari dela. Metodo formalak eguneroko lanean benetan erabilgarriak izan daitezen, beharrezkoa da garatzaileek prestakuntza espezializatua jasotzea. Eta, noski, prestakuntza behar horrek zaildu egiten du metodo horiek esparru industrialean ezartzea. Zailtasun hori eta beste zenbait oztopo identifikatu eta landu dira mundu akademikoaren eta industrialaren arteko lankidetza estuan. Industriak gero eta interes handiagoa erakutsi du metodo formaletan, eta horretan zerikusi handia izan du aplikazio informatikoak gero eta garrantzitsuagoak eta konplexuagoak izateak[4]. Egun, softwarea garatzeko prozesuetan tresna sendoak integratzea lortu da dagoeneko, eta tresna horiek erabilerrazagoak dira. Testuinguru horretan tamaina handiko aplikazio industrial ugarik atera diote etekina metodo formalen erabilerari.

3. «I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail».

4. Nahiz eta esan behar den hardwarearen industriak urteetako aldea atera diola softwarearenari bilakaera honetan.

Metodo formalen erabilera arrakastatsuaren adibide deigarria da Parisko garraioaren automatizazioa. 1989an jarri zen martxan SACEM sistema, eta geroztik sistema horrek kontrolatu du A (RER) lineako tren guztien abiadura. A (RER) sistema Módula-2n idatzita dago eta 21.000 kode-lerro ditu. Kode horren % 63 (segurtasunaren ikuspegitik kritikotzat hartzen dena) automatikoki lortu zen espezifikazio formal batetik, eta, ondoren, egiaztatu egin zen modu erdi-automatikoan. Hau da, softwarea garatzeko teknika formal osagarriak erabili ziren sistemaren segurtasuna bermatzeko. Izatez, Parisko garraioan bada formalki garatutako software kritiko gehiago ere. Horrela, aipagarriak dira honako bi sistema automatiko hauek: batetik, metroaren 14. linean ezarritako sistema, eta, bestetik, París-Roissy lineako gidaririk gabeko autobusa. Lehen sistema arazorik gabe martxan dago 1998tik, eta bigarrena 2007tik.

Oro har, 90eko hamarkadaz gero, metodo formalak erabiliz garatu diren aplikazio industrial ugari eman dira ezagutzera. Gainera, garapen formal horiek dokumentatu egin dira, eta agerian geratu dira metodoen eta erabilitako laguntza-tresnen abantailak. Esate baterako, aipatu ditugun bi aplikazioetarako (Parisko metroa eta autobusa) formalki garatu den softwarea [18]-n eta [12]-n deskribatu da. Horrela garatutako aplikazio industrialetatik anitz garraiorako (nabigaziorako) kontrol-sistemak dira, hala lurreko nola aireko, ureko edota garraio espazialerako. Maiz samar aplikazio horiek erakunde akademikoekin elkarlanean garatu dira. Adibidez, Nieuwe Waterwegeko (Rotterdam-etik gertu) marea-barreraren kontrolerako sistema [24] CMG Public Sector B.V., Division Advanced Technology erakundeak garatu zuen, Twenteko Unibertsitateko *Formal Methods and Tools* taldearen sostenguarekin.

Metodo formalak erabiliz garatu izan diren tamaina handiko sistemen artean aipagarriak dira, halaber, sistema eragileak. Adibidez, L4.verified [5] proiektukoak nabarmentzekoak dira. Bereziki, seL4 [39] mundu errealeko software kritikoa da, formalki egiaztatu dena frogapen formaletarako sistema automatiko baten laguntzaz. 8.700 kode-lerro ditu seL4-k, eta horiez gain baditu behe-mailako lengoaia batean idatzitako beste atal batzuk ere. seL4 izan zen osorik eta formalki egiaztatu zen lehenengo sistema eragilea. Era berean, badira formalki eta osorik egiaztatu diren bestelako aplikazioak ere, hala nola konpiladoreak, komunikazio-protokoloak eta abar. Frogapen automatikorako tresnen nukleoak (*kernel*) ere formalki egiaztatu izan dira.

Metodo formalek industrian lortutako arrakastan eragin handia izan du zenbait enpresa handik softwarearen garapen formalerako tresnak sortzearen alde egindako apustuak. Tresna horiek erabiliz lortu da aplikazio industrial

5. `http://ertos.nicta.com.au/research/l4.verified/`

fidagarriagoak garatzea. Enpresa horien artean, adibide gisa, honako hauek aipa daitezke: Intel, IBM, Sony, Siemens edo Microsoft. Horietako enpresa batzuk beren software-produktuetan erroreak ez egiteagatik murgildu dira arlo honetan. Nolanahi ere, badira metodo formalen ikerkuntzan ahalegin handia egiten ari diren enpresa garrantzitsuak, eta softwarea zorrotz garatzeko tresna praktiko oso erabilgarriak ere sortu dituzte. Nabarmentzekoa da *software development* arloaren barnean metodo formalei buruzko proiektu ugari dituen Microsoft Research dibisioa. Proiektu horiek aurrera eramateko, Microsoftek ospe handiko profesionalak kontratatu ditu, besteak beste, aitzindari izandako C. A. R. Hoare zientzialaria. Proiektu horien bidez softwarea formalki garatzeko prozesuan lagungarri izango diren tresnen multzo bat sortzen ari da Microsoft. Frogatzaile automatikoak ere sortu dituzte, eta frogatzaile horiek tresna orokorragoen azpisistema gisa erabiltzen dira propietate jakin batzuk betetzen al diren frogatu behar denean. Microsoftek sortutako tresnen artean Dafny[6] izeneko lengoaia eta egiaztatzailea nabarmenduko dugu, liburu honen edukiekin lotura estua du eta. Dafny banaketa askekoa da, eta web nabigatzaile baten bidez ere erabil daiteke. Dafnyren webguneak aukera ematen du sistema online atzitzeko, sistemaren instalatzekoak deskargatzeko, eta dokumentazio nahiz laguntza egokiak jasotzeko.

Programen egiaztapen automatikorako (edo automatizaturako) tresnak ugariak dira, eta softwarearen industrian ohikoak diren hainbat lengoaiatan erabil daitezke. Egiaztatzaile ezagunenen eta erabilienen artean daude Spark, Key, ACL2, Isabelle, Jack, KIV, Spec#, eta egiaztatzaile horiek hainbat lengoaiatan idatzitako programak onartzen dituzte, adibidez, Java, Java Card, C, C++, C#, Ada, Occam eta ML. Egiaztapenaren osagarri gisa, sistema horiek bestelako tresnak ere integratzen dituzte, esate baterako, erroreen arazketarako erabiltzen diren proba-kasuen edo kontra-ereduen sortzaileak, edo baita kode-sortzaileak ere. Dafnyk, adibidez, C# kodea sortzen du. Sistema horiek erabiliz garatu diren aplikazio industrial konplexuen erreferentzia anitz topa daiteke literaturan. [93]-n autoreek metodo formalen aplikazio industriala zertan den sakonki aztertu dute. Artikulu horretan erabilgarri dauden tresnak nahiz garatutako aplikazio industrialak aztertu dira, baita aplikazio horien kodearen tamaina, arloko jarduera zientifikoa konferentzietan, aldizkarietan, biltegietan eta abar ere. [93]-n deskribatutako aplikazio industrialen artean daude hegaldien kontrolerako, finantza elektronikoetarako edo segurtasunerako sistemak, eta mikroprozesadoreak.

Azken urteotan nabarmen aurreratu da arrazoibide logikoa automatizatzeko zereginean, eta horrek ekarpen handia egin die metodo formalen euskarri gisa erabiltzen diren tresna praktikoei. Honela dio C. A. R. Hoarek

6. `http://research.microsoft.com/en-us/projects/dafny/`

2006ko [17] artikuluan:

> Azken 10 urte hauetan, frogak eraikitzeaz arduratzen den softwarearen eraginkortasuna 1.000ko faktoreaz hobetu da. Hori, hardwarearen abiaduran, ahalmenean eta eskuragarritasunean gertatutako hobekuntzetan lortutako 100eko faktoreari gehitu behar zaio. Bai softwareak bai hardwareak hobetzen jarraitzen dute, horrela datozen 10 urteetan programen egiaztapenari bidezkoa den aukera emanez.

Azpimarratzekoa da, era berean, lau urte lehenago (2002) Bill Gates-ek programen egiaztapenak gero eta garrantzi handiagoa duela aitortu izana[7]:

> Hamarkada askotan zehar, egiaztapen formala konputazio-zientzien Grial Santua izan da, baina orain oso garrantzitsuak diren arlo batzuetan —esate baterako, kontrolatzaileen egiaztapenean— softwarearen fidagarritasuna bermatzea ahalbidetzen duten tresnak eraikitzen ari gara.

Frogapen automatikoan egiten ari diren aurrerapenak hainbat eratan aplikatzen ari dira programen analisian eta garapenean. Izatez, bi arloen arteko elkar hartzea oso emankorra gertatzen ari da.

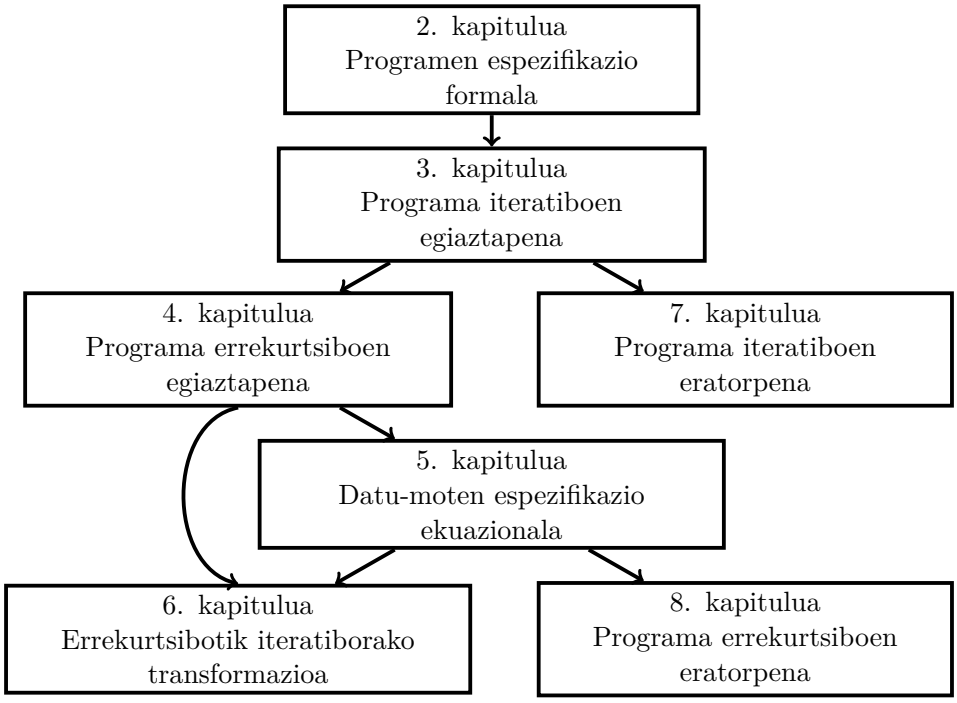
Berriki, 2012an, *DO-178C* dokumentua argitaratu da *Software Considerations in Airborne Systems and Equipment Certification* gaiari buruz. Dokumentu hori argitaratu izanak erakusten du aurrera egin dela metodo formalen gaian, eta, bereziki, softwarearen egiaztapenean. *DO-178C* dokumentuak ezartzen du zein oinarri bete behar dituzten hegazkinetarako software komertzialek, erakunde ziurtatzaileek (Estatu Batuetako Federal Aviation Administration (FAA) eta European Aviation Safety Agency (EASA) esaterako) onar ditzaten. Dokumentu horrek metodo formalei buruzko gehigarria du, eta bertan metodo formalak eta softwarearen egiaztapena honela azpimarratzen dira:

> Metodo formalak sistema digitalen softwarearen espezifikaziorako, garapenerako eta egiaztapenerako erabiltzen diren eta matematikatan oinarrituta dauden teknikak dira. Metodo formalen oinarri matematikoa logika formala, matematika diskretua eta ordenagailuak irakur ditzakeen lengoaiak dira. Ingeniaritzaren beste alorretan bezala, analisi matematiko egokiak egiteak diseinuen zuzentasuna eta sendotasuna bermatzen lagundu dezakeela uste izateak eragin du metodo formalak erabiltzea.

Horrenbestez, badirudi metodo formalak gero eta gehiago erabiliko direla softwarearen garapenean, eta tresna automatiko sendoagoak eta erabilerrazagoak eskainiko direla. Arlo honetan, azkarrago aurreratzeko eta emaitza

7. Windows Hardware Engineering Conference (WinHEC 2002) konferentziako sarrerako hitzaldian.

hobeak lortzeko, ezinbestekoa da software-ingeniaritzako prestakuntzan metodo formalen oinarriak lantzea, bai eta tresna eta aplikazio interesgarrienak ere.



1.1. irudia. Aholkatutako irakurtze-ordena.

1.2. LIBURUAREN DESKRIBAPENA

Liburu honen helburua programazioaren metodologia eta teknologiaren arloan funtsezkoak diren oinarrizko metodo formalez osatutako bilduma bat aurkeztea da, adibide eta ariketen bidez. Liburu honetan, programen eta datu-moten espezifikazioa, programa iteratibo eta errekurtsibo errazen egiaztapena, espezifikazio (edota programa) errekurtsiboak programa iteratibo bihurtzea, eta, aurre-ondoetako espezifikazioetatik abiatuz, bai programa iteratiboen bai errekurtsiboen eratorpena lantzen dira.

Liburu honetako edukiek, ebatzitako eta proposatutako ariketak barne,

Euskal Herriko Unibertsitatean hainbat hamarkadatan zehar eman den Programazioaren Metodologia izeneko irakasgaian dute jatorria. Gaur egun irakasgai hori Informatika Ingeniaritzako Graduan lehenengo mailako bigarren lauhilekoan ematen da. Gradu horretako lehenengo lauhilekoan, programazioko oinarrizko kontzeptuak azaltzen dituen beste irakasgai bat dute ikasleek. Liburu hau, Programazioaren Metodologia irakasgaiko testu-liburu bezala erabil daiteke. Urteetan zehar irakasgaia pixka bat aldatuz joan den arren, muinari eutsi egin zaio. Egun, irakasgaia muin horretara mugatuta dago, eta hori da hain zuzen ere liburu honetan era xehatuan aurkezten eta lantzen dena. Hala eta guztiz, denbora-urritasuna eta zailtasuna direla eta, liburu honetako alderdi edo atal batzuk ez dira aurkezten aipatutako irakasgai horretan.

Liburuko kapitulu bakoitzean, guztiz garatutako adibideak egoteaz gain, ebatzi gabe dauden ariketez osatutako hainbat atal ere badaude. Ebatzi gabeko ariketa horiek beren aurretik garatu diren adibideen zailtasun-maila bera dute gutxi gorabehera. Gainera, kapitulu bakoitzaren bukaeran kapituluan zehar aurkeztutakoa lantzeko aukera ematen duen ariketa-bilduma ere badago. Bestalde, kapitulu bakoitzak interesgarriak diren aipamen historikoak eta erreferentziak jasotzen dituen bibliografia-oharrei buruzko atala ere badu. Sarrerako kapitulu honetan egindako baieztapen batzuk era zehatzagoan dokumentatzeko balio dute kapitulu bakoitzeko erreferentziek.

Liburua irakurtzeko aholkatzen den ordena 1.1. irudian eskematizatu da. Horrela, 2., 3. eta 4. kapituluek programa agintzaileen —bai iteratiboen bai errekurtsiboen— espezifikazioari eta egiaztapenari buruzko funtsezko muina osatzen dute. Bestalde, 5. kapituluan datu-motak espezifikatzeko teknika formal bat jaso da. Hor azaldutakoa 6. kapituluan abiapuntu bezala erabiliko da, datu-moten gaineko eragiketen espezifikazioetatik abiatuz, eragiketa horiek inplementatzen dituzten programak transformazioen bidez nola lor daitezkeen azalduko baita. Bere aldetik, 7. kapitulua 2. eta 3. kapituluetan oinarrituta dago; izan ere, programa bat eta haren zuzentasunaren frogapena era sistematikoan eta zorrotzean (matematikoan) eraikitzen dituen diseinu-metodoaren oinarriak finkatzeko, programen zuzentasuna frogatzeko balio duen metodoa erabiltzen da. Bukatzeko, 8. kapitulua batez ere 4. kapituluan oinarrituta dago, baina 5. kapituluan aurkeztutako ekuazio bidezko espezifikazioak ere erabiltzen dira bertan.

1.3. ESKER ONAK

Liburu honi sorrera eman dion irakasgaia ematen unean-unean aritu diren Euskal Herriko Unibertsitateko Lengoaia eta Sistema Informatikoak Saileko kideei benetako esker ona adierazi nahi diegu. 80ko hamarkadaren hasieratik, ikasketa-plan desberdinak tarteko, irakasgaia bilakatuz eta aldatuz joan da, eta haren edukiak landuz eta hobetuz joan dira. Hainbeste urteetako ibilbidean, asko izan dira beren ekarria utzi duten pertsonak. Eskerrik asko horiei guztiei beren ekarpenengatik. Ikasleen parte-hartzea ere eskertzen dugu; izan ere, haien gogo biziarekin, galderekin eta iruzkin eta iradokizunekin, liburuaren sorrera bultzatzeaz gain, haren edukiak aberasten ere lagundu dute, bereziki adibideei eta ariketei dagokienez.

2. Programen espezifikazio formala

Programak egiten duenaren azalpen edo deskribapen xehea da *programaren espezifikazioa*. Deskribapen hori sarrerako datuen eta irteerako emaitzen arteko erlazioa adieraziz ematen da.

Onura ugari ditu programak garatzean eta mantentzean espezifikazioak erabiltzeak, besteak beste, lantaldeen arteko komunikazioa errazten delako eta programak aztertzeko eta egiaztatzeko egokiak diren deskribapen zehatzak lortzen direlako (esate baterako, programen testak edo probak diseinatzeko). Espezifikazioak idaztean lengoaia informalak edo erdiformalak erabiliz gero, espezifikazioen erabilgarritasuna mugatu egiten da; izan ere, anbiguotasunak eta doitasun ezak errazago agertzen dira azalpenak edo deskribapenak hain era zehatzean adierazi beharrik ez dagoenean. Aitzitik, propietateak eta ondorioak inferitzean deskribapen zehatzak eta arrazoibide zorrotzak ahalbidetzen dituzten lengoaia matematikoetan oinarritzen dira espezifikazio formalak. Espezifikazioen izaera formal hori ezinbestekoa da zenbait prozesu matematiko gauzatu nahi badira, hala nola zuzentasun-frogapen formalak, espezifikazioetatik abiatutako programen eratorpen formalak eta semantikari eutsiz egindako programen transformazioak. Aurre-ondoetako espezifikazio formalek bi formula dituzte: aurreko baldintza eta ondoko baldintza.Aurreko baldintzak programaren hasieran onargarriak diren konputazio-egoera guztiak errepresentatzen ditu; ondoko baldintzak, aldiz, programaren bukaeran onargarriak diren egoera guztiak errepresentatzen ditu. Konputazio-egoera batean edo, soilago esanda, egoera batean, aldagaiei balioak atxikitzen zaizkie, hau da, aldagaien balioespena egiten da. Formula batek formula hori egiazkoa egiten duten egoeren multzoa errepresentatzen du. Hori horrela, programa baten punturen batean (ez derrigor hasieran edo bukaeran) gerta daitezkeen egoera guztien multzoa formulen bidez adierazteko gaitasuna oso baliagarria da programak dokumentatzeko, eta are ezinbestekoa zuzentasunari buruz edo beste zenbait propietate nabarmenei buruz arrazoitzeko.

Kapitulu honetan, hasteko, espezifikazioaren kontzeptua (2.1. atala) eta lehen mailako logika —egoera multzoak errepresentatzeko duen erabilgarritasunaren ikuspegitik— aurkeztuko ditugu (2.2. atala). Programei buruz

arrazoitzeko bereziki erabilgarriak diren lehen mailako logikaren alderdiak zehatz-mehatz aztertuko dira. Adibidez, aldagai libreen (programarenak) eta aldagai atxikien (laguntzaileak) arteko aldea ezartzen duten kuantifikatzaileak. Jarraian, programa desberdinetan gerta daitezkeen konputazio-egoerei buruzko baieztapenak adierazten dituzten asertzioak/formulak idaztea eskatzen duten ariketa batzuk datoz (2.3. atala). 2.4. atalean aldagaiak terminoekin ordezkatzen dituen eragiketa aztertuko da. Erakutsiko den bezala, eragiketa horrek programazioko esleipenarekin eta, horrenbestez, konputazio-egoeren oinarrizko aldaketarekin zerikusi estua du. 2.5. atalean, formulen (bereziki aurreko baldintzen, ondoko baldintzen eta inbarianteen) ahultze eta gogortzearekin erlazionatuta dagoen inplikazio logikoa aztertuko da. 2.6. atalean, lehen mailako logika tresnatzat hartuta, aurre-ondoetako espezifikazio formalaren teknika aurkeztuko dugu. Espezifikazio-teknika hori ebatzi beharreko problemaren deskribapen matematikoa egiteko egokia dela erakutsiko dugu, horretarako, batez ere, adibideak eta ariketak erabiliz. Gainera, teknika hori ondo erabiliz gero, espezifikazioetako deskribapenak zehatzak, argiak eta trinkoak izango dira, behar adinako murrizketak ezarriz (eta, horrela, onartezinak diren programak saihestuz), baina beharrezkoak ez diren baldintzak ezarri gabe; hau da, murriztearen eta orokortzearen arteko orekari behar bezala eutsi ahal izango zaio. Bukatzeko, 2.7., 2.8. eta 2.9. ataletan aurre-ondoetako espezifikazioari buruzko ariketak, erreferentzia bibliografiko batzuk eta kapitulu osoan zehar ikusitakoa laburbiltzen duten ariketak proposatuko dira hurrenez hurren.

2.1. ESPEZIFIKAZIOAREN KONTZEPTUA

Hiztegiek diotenez, *espezifikazioa* egin beharreko lanaren deskribapen zehatza da. Definizio horren haritik, *programen espezifikazioa* diogunean, programen egitekoaren deskribapen zehatzaz ari gara. Bi osagai izan ohi ditu programa baten espezifikazioak: batetik, datuek bete behar dituzten propietateak adierazten dira; bestetik, emaitzen ezaugarriak deskribatzen dira (datuen arabera). Bi alderdi horiek elkarren osagarri dira, eta bien elkarketak osotasuna ematen dio deskribapenari. Programa baten datuen eta emaitzen arteko erlazioa funtzio matematiko baten gisara uler daiteke: datuak lirateke funtzioaren sarrera, eta emaitzak, berriz, funtzioaren irteera. Espezifikazioa funtzioaren deskribapen zehatza da, horretara. Programa, ordea, funtzioaren inplementazioa da, hau da, datuetatik abiatuta emaitzak lortzeko era jakin bat. Eta, noski, espezifikazio bat emanda, espezifikazio hori betetzen duten programa bat baino gehiago inplementa daiteke.

Espezifikazioak informatikako hainbat alorretan erabiltzen dira. Programen diseinuan berebiziko garrantzia du espezifikazioak. Izan ere, edozein programa diseinatzerakoan premiazkoa da ebatzi beharreko problemaren ezagutza zehatza (anbiguotasunik gabea). Ezagutza hori diseinuari ekin aurretik eduki behar da, diseinatu ahala problemaren deskribapenaz jabetzea akatsiturri larria delako. Problema espezifikatzeak ideiak egituratzera behartzen gaitu eta alderdi garrantzitsuak agerian uztera, bidenabar. Alderdi horiek, espezifikazioan ez bada, diseinuan agertuko dira edo, are okerrago, ez dira agertuko, eta orduan programa ez da egokia izango, ez baitu nahi dugun problema ebatziko. Programak eta aplikazio informatikoak talde-lanean egin ohi dira. Hori horrela, espezifikazioa komunikaziorako ezinbesteko bitartekoa da. Sistema bat modulutan egituratzeko, sistemaren eginbeharrak zehatz-mehatz ezagutu behar dira, eta modulu bakoitzaren diseinatzaileak inolako anbiguotasunik gabe ezagutu behar ditu bai problema orokorra eta baita beste moduluekiko elkarrekintzak ere. Programen testak diseinatzeko ere ezinbestekoak dira espezifikazioak: programak ezin dira testeatu beren eginkizuna zehaztasunez ezagutzen ez bada. *Kutxa beltz* deritzen testek espezifikazioa dute abiapuntu. Bestalde, ez dugu ahantzi behar programak behin egin eta askotan irakurtzen direla. Programa bat mantentzeko beharrezkoa da dokumentazio egokia erabiltzea, edonork programa horrek zer egiten duen zehazki uler dezan aparteko ahaleginik gabe. Gerora ere, aldaketaren bat egin beharko denean, ezinbestekoa izango da dokumentazioa. Kontuan hartu behar da, gainera, aldaketak egiten dituena eta hasierako diseinua egin zuena ez direla pertsona bera izan ohi.

Liburu honetan, espezifikazioa programen egiaztapenaren, eratorpenaren eta transformazioaren abiapuntu bezala erabiliko da. Egiaztapenaren helburua programek beren espezifikazioa betetzen dutela formalki frogatzea da, hau da, egin beharrekoa egiazki egiten dutela frogatzea. Beraz, egiaztapenean oinarritzen diren diseinu-metodologien oinarria ere badira espezifikazioak. Programen eratorpena, transformazioa eta sintesi automatikoa egiten duten metodoek ere espezifikazioetan dute oinarria. Arlo honetan, exekutagarriak diren espezifikazio-lengoaiak ere badira, hau da, programazio-lengoaia bilakatu diren espezifikazio-lengoaiak. Programazio-lengoaia horiek *deklaratiboak* direla esan ohi da, nolabait problemaren deskribapenean oinarrituz egiten baitituzte kalkuluak, problema ebazteko balio duen algoritmo bati jarraitu ordez. Laburbilduz, espezifikazioak bereziki baliagarriak eta beharrezkoak dira bai programen diseinuan, mantentze-lanetan, egiaztapenean, eratorpenean, transformazioan eta sintesian, baita programen proben diseinurako ere, bai eta softwarea garatzen diharduten taldeetako kideen arteko komunikazio egokia bermatzeko ere.

Espezifikazioak benetan erabilgarriak izan daitezen, badira zaindu beharreko zenbait ezaugarri. Oro har, espezifikazioek, egokiak izango badira, honako propietate hauek bete behar dituzte:

- Argitasuna: ulerterrazak.
- Laburtasuna: erredundantziarik gabeak.
- Doitasuna: anbiguotasunik eza.
- Murriztasunaren eta orokortasunaren arteko oreka: alferrikako xehetasunik gabeak, baina xehetasun garrantzitsu guztiak dituztenak.

Programen espezifikazioak idazteko formatu (edo egitura) eta lengoaia askoren artean aukera daiteke. Bi taldetan banatu ohi dira espezifikazio-lengoaiak: formalak eta informalak. Lengoaia bat formaltzat hartuko dugu haren sintaxia eta semantika formalki (zehaztasun matematikoz) definituta baldin badaude. Espezifikazio informalerako teknikek, aldiz, deskribatu beharreko ezaugarriak edo alderdiak hitz esanguratsu baten bidez nabarmendutako klausulen bidez adierazteko formatua edo egitura finkatzen dute. Beraz, formalak ez diren espezifikazio-tekniketan egiturazko moldea ezartzen da, baina ez klausulak adierazteko lengoaiek nolakoak izan behar duten. Espezifikatzen duenaren esku geratzen da, hortaz, notazio matematiko/teknikoa edo lengoaia naturala aukeratzea klausula bakoitza egoki idazteko. Ohikoak dira, esate baterako, datuak, esperotako emaitzak eta portaera, aldatuko diren datuak, edota salbuespenezko portaerak deskribatzen dituzten klausulak.

Espezifikazio bat idaztean beharrezkoa den formaltasun-maila espezifikazioari emango zaion erabilerak baldintzatuko du: taldeen arteko komunikaziorako, programen testen diseinurako edo programen dokumentaziorako lengoaia informala nahikoa izan daiteke; baina programen egiaztapenerako eta eratorpenerako (hain zuzen ere liburu honetan lantzen diren gaiak), sintesirako eta transformazio automatikorako, edo espezifikazioen exekuziorako, ezinbestekoa da espezifikazioak automatikoki prozesatzeko eta matematikoki analizatzeko aukera ematen duen lengoaia formal bat erabiltzea. Gainera, espezifikazio-teknika formalen erabileran trebatzeak abstrakzio- eta formalizazio-gaitasunak lantzea dakar, eta gaitasun horiek oso mesedegarriak dira bestelako teknika informalekin lan egitean ere. Liburu honetan lehen mailako logikan oinarritzen den espezifikazio-teknika formal bat erabiliko dugu.

2.2. LEHEN MAILAKO ASERTZIOAK: EGOERAK ADIERAZTEKO FORMULAK

Aurreko atalean esan dugun bezala, programa baten espezifikazioak sarrerako datuen eta lortu beharreko emaitzen arteko erlazioa adierazten du: azken batean, datuetatik abiatuz programak *zer* ebazten duen edo *zer* lortzen duen zehazten du. Baina espezifikazioak ez du adierazten *nola* iritsi datuen eta emaitzen arteko erlazio hori betetzera; hau da, espezifikazioak ez du argitzen *nola* kalkulatzen diren emaitzak sarrerako datuetatik abiatuz. Kontuan hartuta, gainera, espezifikazio bat era desberdinetan inplementa daitekeela, garbi dago espezifikazioak ez duela deskribatzen programak zein eratan ebazten duen problema. Aginduzko programek aldagaien balioak aldatzen dituzten agindu-sekuentziak egikaritzen dituzte. Hori dela-eta, emaitzetara iristeko egin behar den aldagaien transformazio-prozesua esplizituki errepresenta daiteke aldagaien edozein hasierako baliotarako. Programa bat hartuta, une jakin batean programako aldagaiek duten balioa deskribatzen duen (*aldagaia*, *balioa*) erako bikoteez osatutako multzoari *konputazio-egoera* (edo besterik gabe, *egoera*) esaten zaio, eta une jakin batean programako aldagaiek duten balioa deskribatzen du. Nabarmentzekoa da exekuzioaren une batera iritsitakoan, oraindik baliorik ez duten aldagaiak egon daitezkeela. Hori horrela denean, une hori baino lehenagoko egoeretan aldagai horiek ez dute baliorik egokitzen dien (*aldagaia*, *balioa*) erako bikoterik izango. Aginduek, aldagaien balioak aldatuz, egoerak aldarazten dituzte. Adibidez, har dezagun datu bezala negatiboa ez den x zenbaki osoa eta positiboa den y zenbaki osoa jasotzen dituen, eta emaitza bezala x eta y zenbakien arteko z zatidura osoa eta zatiketa osoaren h hondarra kalkultatzen dituen honako P programa hau:

```
(1)
  z := 0;
(2)
  h := x;
(3)
  while h >= y loop
    z := z+1;
  (4)
    h := h-y;
  (5)
  end loop;
(6)
```

Demagun hasierako egoera, (1) puntuan, honako hau dela

(1) $\{(x,31),(y,7)\}$

Programako gainerako aldagaiak (z eta h) hasierako egoeran ez dira agertzen, oraindik ez baitzaie baliorik esleitu. Lehenengo agindua (`z := 0;`) exekutatutakoan, z aldagaiak 0 balioa hartuko du. Beraz, bigarren egoera, (2) puntuan, honako hau izango da:

(2) $\{(x,31),(y,7),(z,0)\}$

Era berean, bigarren agindua (`h := x;`) exekutatutakoan, programaren exekuzioari dagokion egoera honela geratuko da:

(3) $\{(x,31),(y,7),(z,0),(h,31)\}$

Ondoren, *while* aginduaren exekuzioa hasiko da. Iterazioak aurrera egin ahala, honako taula honetan erakusten diren balioak hartuz joango dira aldagaiak:

Iterazioa	Egoera	x	y	z	h
1	(4)	31	7	1	31
	(5)	31	7	1	24
2	(4)	31	7	2	24
	(5)	31	7	2	17
3	(4)	31	7	3	17
	(5)	31	7	3	10
4	(4)	31	7	4	10
	(5)	31	7	4	3

Azkenean, *while* aginduaren exekuzioa bukatu ondoren, honako egoera hau izango dugu:

(6) $\{(x,31),(y,7),(z,4),(h,3)\}$

Horrenbestez, P programak hasierako egoera honetatik abiatuz

(1) $\{(x,31),(y,7)\}$

bukaerako egoera hau lortuko du:

(6) $\{(x,31),(y,7),(z,4),(h,3)\}$

Transformazio hori sarrerako datu zehatz batzuetatik abiatuz egindako konputazio zehatz bat da. Programa batek burutu ditzakeen konputazio guztiak esperotako emaitzetara iristen badira, orduan programa hori zuzena izango da. Baina, oro har, programa batek buru ditzakeen konputazioen kopurua infinitua izango da, edo, bestela esanda, programaren abiapuntu izan daitezkeen egoeren edo sarrerako datuen kopurua infinitua da. Hori horrela izanda, egoerak erabiltzea ez da nahikoa izaten programa bat espezifikatzeko. Gabezia horri aurre egin eta orokorragoa den adierazpide bat edukitzeko, *asertzioak* erabiltzen dira. Asertzioak programetako puntu desberdinetan iruzkin gisa ipintzen diren espresio logikoak dira. Asertzio bakoitzak berari dagokion puntuan aldagaiek zein propietate beteko dituzten adierazten du. Horrela izanda, programaren edozein konputazio asertzio bat dagoen puntura iristen denean, programaren egoera osatzen duten aldagaien balioek asertzio horrek dioena beteko dute. Beraz, asertzio bakoitzak programan dagokion puntuan gerta daitezkeen (eta infinitua izan ohi den) egoeren multzoa errepresentatzen du. Beste era batera ipinita, asertzioaren bidez, asertzioak dioena egiazko egiten duten (edo *betetzen* duten) egoerak errepresentatzen dira. Asertzioen sintaxia eta semantika datozen ataletan definituko badira ere, asertzioak zer diren ulertzen lagun dezakeen adibidea dator jarraian. Har dezagun honako asertzio hau:

$$\{\ \varphi \equiv x = y * z + h\ \}$$

Asertzio horri buruz honako baieztapenok egin ditzakegu:

- $\{(x,31),(y,7),(z,4),(h,3)\}$ egoeran *egiazkoa* da
- $\{(x,31),(y,7),(z,4),(h,0)\}$ egoeran *faltsua* da
- $\{(x,31),(y,7)\}$ egoeran *indefinitua* da

Asertzioen bidez, programen konputazioak eta, aldi berean, programak zuzenak izateko aldagaien artean bete beharreko erlazioak deskriba daitezke. Horregatik, asertzioak programak dokumentatzeko erabil daitezke. Esate baterako, zatidura osoa eta hondarra kalkulatzen dituen programa berriro hartuz, honela dokumenta daiteke:

$\{\, x \geq 0 \wedge y > 0 \,\}$
```
  z := 0;
  h := x;
  while h >= y loop
```
$\{\, x = y*z+h \wedge y > 0 \wedge x \geq h \geq 0 \wedge z \geq 0 \,\}$
```
    z := z+1;
    h := h-y;
  end loop;
```
$\{\, x = y*z+h \wedge y > h \geq 0 \wedge z \geq 0 \,\}$

Honako beste adibide honetan hutsa ez den $A(1..n)$ array-ko (edo bektoreko)[1] elementuen batura x aldagaian kalkulatzen duen programa daukagu:

$\{\, n \geq 1 \,\}$
```
  i := 0;
  x := 0;
  while i < n loop
```
$\{\, x = \sum_{j=1}^{i} A(j) \wedge 0 \leq i \leq n \,\}$
```
    i := i+1;
    x := x+A(i);
  end loop;
```
$\{\, x = \sum_{j=1}^{n} A(j) \,\}$

Asertzioak adierazteko lengoaia formal bat erabiliko dugu: lehen mailako logika, hain zuzen ere. Hau da, asertzio bakoitza lehen mailako logikan idatzitako formula bat izango da. *Lehen mailako logika* (*predikatu-logika* eta *predikatu-kalkulua* ere esaten zaio) lehen mailako lengoaietan idatzitako propietateen inferentzia aztertzeko sistema formal bat da. Lehen mailako lengoaiak lengoaia formalak izaten dira eta honako osagai hauek erabiltzen dituzte: i) aldagai indibidualak, ii) (lehen mailako) terminoei aplikatzen zaizkien funtzioak eta predikatuak, iii) eragile logikoak (ukapena, konjuntzioa, disjuntzioa, inplikazioa, inplikazio bikoitza), eta iv) aldagai indibidualen gaineko kuantifikatzaileak. Gure programetan aldagai indibidual gehienak zenbaki osoen motakoak izango dira. Hau da, formulek zenbaki osoen propietateak eta erlazioak adieraziko dituzte, edo zenbaki osozko arrayenak[2]. Batzuetan boolearrak eta karaktereak ere erabiliko ditugu.

1. *array* eta *bektore* izendapenetatik edozein erabiliko dugu dimentsio bakarreko egitura indexatuak aipatzeko.

2. Liburu honetan zenbaki errealak ez dira erabiliko, hortaz, zenbakia aipatzen dugunean defektuz zenbaki osoez arituko gara

2.2.1. Lehen mailako asertzioen sintaxia

Lengoaia formal guztiek matematikoki definitutako sintaxi eta semantika zehatzak izaten dituzte. Asertzioen idazkerako sintaxiari dagokionez, lehen mailako logikaren lengoaia erabiliko dugu, oinarrizko alfabeto bezala lengoaia matematikoan ohikoak diren sinbolo denak bai eta ezagunenak diren agindu bidezko programazio-lengoaietan (adibidez, ADAn) erabiltzen diren sinboloak ere hartuz. Gainera, batzuetan sinbolo horiek euskarara itzuliko ditugu (esate baterako, *bikoitia* erabiliko dugu ingelesezko *even* erabili ordez). Hona hemen erabiliko ditugun sinboloak nolakoak izango diren erakusten duten adibide batzuk:

- Aldagai-sinboloak: x, y, z, ...
- Konstante-sinboloak: 0, 1, ..., $'a'$, $'b'$, $'c'$, ...
- Funtzio-sinboloak: $+$, $-$, $*$, ..., %[3], *abs*, ...
- Predikatu-sinboloak: $<$, $=$, $\geq$, $\neq$, ..., *bikoitia*, *bakoitia*, ...

Aldagai-, konstante-[4] eta funtzio-sinboloak unitate indibidualak adierazteko erabiltzen dira; horiek konbinatuta *terminoak* osatzen dira. Predikatuak (eragileekin eta kuantifikatzaileekin konbinatuz) formulak osatzeko erabiltzen dira. Hau da, *termino*ek elementu indibidualak adierazten dituzte, eta formulek elementu horien gaineko propietateak. Funtzio- eta predikatu-sinbolo orok bere *aritatea* du. Sinbolo baten aritatea sinbolo horrek zenbat argumentu behar dituen adierazten duen zenbaki arrunta da. Esate baterako, *bikoitia* predikatuaren aritatea 1 da (beraz, bakuna da), eta $<$ sinboloaren aritatea 2 da (beraz, bitarra da). Sinbolo baten «aritatea n dela» esateko, sinbolo hori n-tarra dela ere esango dugu.

Era errekurtsiboa erabiliz, *(lehen mailako) terminoak* honela defini daitezke:

- Termino sinpleak: aldagaiak eta konstanteak termino sinpleak dira.
- Termino konposatuak: $n > 0$ izanda, f funtzio n-tarra baldin bada eta $t_1, \ldots, t_n$ terminoak baldin badira, orduan $f(t_1, \ldots, t_n)$ termino konposatua da.
- Termino oro sinplea edo konposatua izango da.

3. % sinboloaren bidez zatiketa osoaren hondarra kalkulatzen duen eragiketa adieraziko dugu.

4. Konstanteak 0 argumentuko funtzio gisa ikus daitezke, baina hemen nahiago dugu beste sinbolo mota bat bezala ikusi.

Lehenago (konstanteen eta aldagaien sinboloak ikusi ditugunean) termino sinpleen adibideak eman ditugu. Honako hauek termino konposatuen adibideak dira:

$$x+1,\ x\ \%\ 2,\ (5-A(i))*(-z),\ abs(z-3) \text{ eta } A(i)/A(j)$$

Kontuan izan funtzio aritmetiko gehienak (batuketa, biderketa, etab.) infixuak direla, hots, bi argumentuen artean idazten direla.

7. kapituluan azalduko dugun bezala, arrayen osagaiak (hau da, $A(i)$ erako elementuak) ezin daitezke aldagai indibidualtzat hartu programen egiaztapena egiterakoan. Arrayen osagaiei eginiko esleipenak egoki tratatzeko teknika kapitulu horretan aurkeztuko dugu. Bien bitartean, 2. kapitulutik 6.era, arrayen osagaiei eginiko esleipenak (adibidez, $A(i) := x$;) saihestu egingo ditugu.

Terminoaren kontzeptua oinarritzat hartuta, *(lehen mailako) formularen kontzeptua* errekurtsiboki honela definitzen da:

- Formula sinpleak (formula atomiko edo atomo ere esaten zaie): $n \geq 0$ izanda, p predikatu n-tarra baldin bada eta $t_1, \ldots, t_n$ terminoak baldin badira, $p(t_1, \ldots, t_n)$ formula sinplea da. $True$ eta $False$ ere formula sinpleak dira[5].

- Formula konposatuak: formula sinpleak edo konposatuak oinarritzat hartuz beste formula konposatuak lortzeko $\neg$ (ezeztapena, ukapena), $\wedge$ (konjuntzioa, eta), $\vee$ (disjuntzioa, edo), $\rightarrow$ (inplikazioa) eta $\leftrightarrow$ (baliokidetza, inplikazio bikoitza) eragile logikoak, eta $\forall$ kuantifikatzaile unibertsala eta $\exists$ kuantifikatzaile existentziala erabiltzen dira. Ezeztapena eragile bakuna da (formula bakar bati aplikatzen zaio), eta beste guztiak bitarrak dira (bi formula behar dituzte) eta, gainera, infixuak dira (formula bien artean idazten dira). Kuantifikatzaileek x aldagai (indibidual) bat eta φ formula bat behar dituzte, eta $\forall x\ \varphi$ edo $\exists x\ \varphi$ egitura duten formulak lortzen dira.

- Formula oro sinplea edo konposatua izango da.

Honako bost espresio hauek atomoak dira: $x < y$, $z = 2$, $A(i) > (A(j) + 1)$, $bakoitia(x)$ eta $bikoitia(2)$. Beste hauek, berriz, formula konposatuen adibideak dira:

5. $True$ eta $False$ formula atomikoak ez dira nahastu behar T eta F bezala adieraziko ditugun mota boolearreko bi balioekin.

$$\neg(\ x = y\ )$$
$$x > 0\ \wedge\ x \leq 10$$
$$\forall z\ (\ bakoitia(z)\ \vee\ bikoitia(z)\ )$$
$$(\ y = 1\ \vee\ x = 5\ )\ \rightarrow\ \neg bakoitia(x)$$
$$\forall z\ (\ bikoitia(z)\ \rightarrow\ \exists v\ (\ z = v + v\ )\ )\ \wedge\ x \leq 10$$

$\forall x\ \varphi$ eta $\exists x\ \varphi$ erako formuletan, φ formula $\forall x$ eta $\exists x$ kuantifikatzaileen *esparrua* dela esaten da. Formula konposatuetan, x algagai baten agerpena *atxikia* dela esaten da, $\forall x$ edo $\exists x$ kuantifikatzaileen esparruan agertzen bada. Atxikiak ez diren agerpenei *libre* deritze. Adibidez, lerro batzuk lehenago ikusi ditugun formula konposatuetatik azkenekoa hartzen badugu, x aldagaiaren agerpena librea da, baina z eta v aldagaien agerpen guztiak atxikiak dira. Aldagai batek gutxienez agerpen libre bat baldin badu formula batean, orduan aldagai hori formula horretan libre dago eta formula horretako aldagai libreen multzoan egongo da. φ formulan libre agertzen diren aldagaiez osatutako $AL(\varphi)$ multzoa honela defini daiteke:

- $AL(p(t_1, \ldots, t_n)) = ald(t_1) \cup \ldots \cup ald(t_n)$.

  Hor $ald(t)$ espresioak t terminoan agertzen diren aldagai guztiez eratutako multzoa adierazten du.

- $AL(True) = AL(False) = \emptyset$.

- $AL(\neg\varphi) = AL(\varphi)$.

- $AL(\varphi \wedge \psi) = AL(\varphi \vee \psi) = AL(\varphi \rightarrow \psi) = AL(\varphi \leftrightarrow \psi) = AL(\varphi) \cup AL(\psi)$.

- $AL(\forall x\ \varphi) = AL(\exists x\ \varphi) = AL(\varphi) \setminus \{x\}$.

  Hor $\setminus$ eragileak multzoen arteko diferentzia (edo kenketa) adierazten du.

2.2.2. Lehen mailako asertzioen semantika

Asertzio batek konputazio-egoera batean balioa izan dezan (egiazkoa/faltsua), egoera horrek aldagai libre guztiei balio bana egokitu beharko die. Baldintza hori betez gero, egoera horretan asertzioak interpretazio semantikoa izango du eta, interpretazio horren arabera, asertzio hori egoera horretan egiazkoa ala faltsua den jakin ahal izango da. Aurrerantzean, formula bat egiazkoa dela adierazteko T balioa erabiliko dugu eta faltsua dela adierazteko F balioa erabiliko dugu. Asertzio bat programako puntu batean idatzita dagoenean, asertzio hori puntu horretan egiazko (T) egiten duten egoeren multzoa errepresentatzen ari da.

Lehenbizi terminoei semantika eman behar diegu, hau da, interpretatu egin behar ditugu egoeretan. Terminoek balioak errepresentatzen dituzte; izan ere, termino batek mota egokiko balio bat hartzen du interpretatzen denean. Alde batetik, egoera bakoitzean egoera horretan dagokien balioa hartzen dute aldagaiek. Beste aldetik, konstanteek eta funtzioek beren esanahi estandarra dute erabilitako programazio-lengoaian edo lengoaia matematikoan. Horrenbestez, egoera bakoitzean termino bakoitzak duen balioa funtzioen esanahi estandarra aldagaien balioei eta konstanteei aplikatuz lortzen da. Adibidez, $s_1 = \{(x,31),(y,7),(z,12)\}$ egoeran interpretatuta $(x+y)/2$ terminoaren balioa 19 zenbaki osoa da, eta $s_2 = \{(x,31),(y,-7),(z,12)\}$ egoeran interpretatuta haren balioa 12 da. Ohartzekoa da termino hori ebaluatzeko z aldagaiaren balioa ez dela behar, baina horrek ez du inolako arazorik sortzen. Alderantzizko kasua, ordea, problematikoa da: zein da $(x+y)/2$ terminoaren balioa $s_3 = \{(x,31),(z,12)\}$ egoeran? Arazo hori dela-eta, s egoera batean termino baten aldagai guztiek balioa baldin badute, termino hori egoera horretan *definituta* dagoela esango dugu. Berriro adibidera joz, $(x+y)/2$ terminoa s_3 egoeran indefinituta (edo definitu gabe) dagoela esango genuke.

Era berean, formulek ere T (egiazkoa) edo F (faltsua) balioa izango dute egoera batean interpretatzen direnean. Edozein egoeratan $True$ atomoa T da eta $False$ atomoa F da. Asertzioetan erabiliko diren predikatu-sinboloek ere esanahi estandarra dute eta gainerako formula atomikoak interpretatzean predikatuen esanahi hori hartu beharko da kontuan. Horrela, adibidez, s_1 egoeran $((x+y)\,/\,2) > z$ formulari T balioa dagokio, baina s_2 egoeran F balioa dagokio, eta s_3 egoeran definitu gabe dago. φ formula s egoeran *definituta* egongo da, haren aldagai libre guztiek balio bat baldin badute egoera horretan. Aitzitik, s egoeran φ formulako aldagai libreren batek baliorik ez badu, φ formula egoera horretan ez dagoela definituta esango dugu. Egoera bakoitzean egoera horretan definituta dauden formulak bakarrik interpreta daitezke. Formula konposatuak interpretatzeko eragile logikoen semantika estandarra erabiliko dugu. Semantika hori honako egia-taula hauetan jasotzen da:

φ	ψ	$\neg\varphi$	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \to \psi$	$\varphi \leftrightarrow \psi$
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

Kuantifikatzaileak nola interpretatzen diren definitzeko, egoerak aldatzen dituen eragiketa bat behar dugu. Beraz, x aldagaia, x-ren mota bereko v balioa eta s egoera emanda, $s[x \leftarrow\!\shortmid v]$ bezala adieraziko dugun egoeran x aldagaiak

v balioa du eta beste edozein aldagaik s-n daukan balio berbera izango du. Hori horrela, x aldagaiak s egoeran balioa izan ala ez, $s[x \leftarrow\!\shortmid v]$ egoeran duen balioa beti v izango da. Kuantifikatzaile unibertsalaren semantika definitzeko, s egoeran $\forall x\, \varphi$ formula T dela esango dugu, baldin eta soilik baldin x-ren motako edozein v baliorentzat (hau da, x-ren motako v balio guztientzat) $s[x \leftarrow\!\shortmid v]$ egoeran φ formula T bada. Kuantifikatzaile existentzialaren kasuan, s egoeran $\exists x\, \varphi$ formula T izango da, baldin eta soilik baldin v balioren batentzat (hau da, gutxienez x-ren motako v balio baten kasuan) $s[x \leftarrow\!\shortmid v]$ egoeran φ formula T bada.

Asertzioak idaztean, oso ohikoa da aldagai kuantifikatu batek har ditzakeen balioak mota bereko azpimultzo batera murriztu behar izatea (gehienetan $\mathbb{Z}$ multzora). Azpimultzo hori errepresentatzen duen formulari kuantifikazioaren *definizio-eremu* esaten zaio, eta $D(x)$ notazioaz adierazten da. Hona hemen ohikoak diren definizio-eremuen adibide batzuk:

$$1 \leq x \leq 10$$
$$x > 10$$
$$1 < x \leq 100 \,\wedge\, z > 0 \,\wedge\, x \neq z$$
$$x \geq 2 \,\wedge\, x \,\%\, 2 = 0$$

Asertzioetan, definizio-eremua finkatzeaz gain, definizio-eremuko balioek bete behar duten propietatea beste formula baten bidez definitzen da. *Propietate* deritzo formula horri, eta $P(x)$ notazioaz adierazten da. D definizio-eremuko elementu guztiek P propietatea betetzen dutela esateko erabili ohi den formula honako hau da:

$$\forall x\, (\, D(x) \,\rightarrow\, P(x)\,)$$

D definizio-eremuan P propietatea betetzen duen elementuren bat badela baieztatzen duen formula, berriz, hauxe da:

$$\exists x\, (\, D(x) \,\wedge\, P(x)\,)$$

D definizio-eremuak murrizketarik ezartzen ez duenean (beraz, definizio-eremua kuantifikatutako aldagaiaren motako balio guztiez osatutako multzoa bezala definituz), hau da, $D(x)$ $True$-ren baliokidea denean, bi formula horiek honela sinplifikatzen dira hurrenez hurren: $\forall x\, P(x)$ eta $\exists x\, P(x)$. Era berean, $D(x)$ $False$-ren baliokidea denean (hau da, definizio-eremua hutsa denean), honako hau dugu:

$$\forall x\, (\, False \,\rightarrow\, P(x)\,) \quad True\text{-ren baliokidea da}$$
$$\exists x\, (\, False \,\wedge\, P(x)\,) \quad False\text{-ren baliokidea da}$$

Kasu horietan, kuantifikatzaile unibertsalari ($\forall$) dagokion formula $True$ formularekin ordezka daiteke, eta kuantifikatzaile existentzialari ($\exists$) dagokion formula $False$ formularekin ordezka daiteke.

Errepikatzen den eragiketa baten ikuspuntua hartuta ere azal daiteke kuantifikatzaileen semantika. Unibertsalki kuantifikatutako formula bat konjuntzioz osatutako segida gisa ikus daiteke. Segida hori batzuetan infinitua ere izan daiteke. Bestalde, existentzialki kuantifikatutako formula bat disjuntzioen segidatzat har daiteke. Adibidez,

$$\forall x \; (\; 1 \leq x < 4 \; \rightarrow \; A(x) < 8 \;)$$

formula konjuntzioz eratutako honako segida honen baliokidea da:

$$A(1) < 8 \; \wedge \; A(2) < 8 \; \wedge \; A(3) < 8$$

eta

$$\exists y \; (\; 2 < y \leq 5 \; \wedge \; B(y) \geq 1 \;)$$

formula disjuntzioz eratutako honako beste segida honen baliokidea da:

$$B(3) \geq 1 \; \vee \; B(4) \geq 1 \; \vee \; B(5) \geq 1$$

Konjuntzioz edo disjuntzioz osatutako segiden bidez adierazteko aukera eduki arren, propietate asko era laburragoan adieraz daitezke kuantifikatzaileak erabiliz. Horren erakusgarri dira honako adibide hauek:

- *«*$A(1..10)$ *zenbaki positiboz eratutako bektorea da»*:

$$\forall i \; (\; 1 \leq i \leq 10 \; \rightarrow \; A(i) > 0 \;)$$

- *«*$A(1..10)$ *bektoreak badu gutxienez zenbaki negatibo bat»*:

$$\exists i \; (\; 1 \leq i \leq 10 \; \wedge \; A(i) < 0 \;)$$

- *«*$A(1..10)$ *bektoreak negatiboa den zenbaki bakar bat du»*:

$$\exists i \; (\; 1 \leq i \leq 10 \; \wedge \; A(i) < 0 \; \wedge \; \forall j \; (\; (\; 1 \leq j \leq 10 \; \wedge \; i \neq j \;) \; \rightarrow \; A(j) \geq 0 \;) \;)$$

Kuantifikatzaileak konjuntzioz edo disjuntzioz osatutako segiden bidez ulertzeko ikuspegi horretatik, kuantifikatutako formularen $D(x)$ definizioeremua hutsa denean, $\forall x \; (\; D(x) \; \rightarrow \; P(x) \;)$ kuantifikazio unibertsala konjuntzioaren neutrotzat har daiteke: $True$. Era berean, $False$, disjuntzioaren neutroa, $\exists x \; (\; D(x) \; \wedge \; P(x) \;)$ kuantifikazio existentzialaren baliokidea da $D(x)$ hutsa denean.

Aurreko adibideetan kuantifikatzaileak ez dira guztiz beharrezkoak; izan ere, konjuntzioz edo disjuntzioz eratutako segida baliokideen bidez idatz daitezke. Hala ere, badira konjuntzioz edo disjuntzioz eratutako segida infinituen bidez bakarrik adierazi ahal izango liratekeen propietateak ere. Propietate horiek denek definizio-eremu (D) infinitua dute. Adibidez:

- *«Bada 5 zenbakiaz zatigarria den zenbaki positiboa»*:

$$\exists x \, (\, x \geq 1 \, \wedge \, x \, \% \, 5 = 0 \,)$$

- *«2 baino handiagoa eta bikoitia den edozein zenbaki ez da zenbaki lehena»*:

$$\forall x \, (\, (x > 2 \, \wedge \, x \, \% \, 2 = 0 \,) \, \rightarrow \, \exists i \, (\, 1 < i < x \, \wedge \, x \, \% \, i = 0 \,) \,)$$

edo

$$\forall x \, (\, (x > 2 \, \wedge \, x \, \% \, 2 = 0 \,) \, \rightarrow \, \neg \forall i \, (\, 1 < i < x \, \rightarrow \, x \, \% \, i \neq 0 \,) \,)$$

Formulak idazteko kuantifikatzaileak erabiltzen diren bezala, aldagai atxikia duten eragileak ere erabil daitezke terminoak idazteko. Aldagai atxiki horiek definizio-eremu baten barruko balioak hartuko dituzte. Batuketen eta biderketen segidak laburtzeko erabiltzen diren eragileak dira ezagunenak:

$$\sum_{x=i}^{j} f(x) = f(i) + f(i+1) + \ldots + f(j-1) + f(j)$$

$$\prod_{x=i}^{j} f(x) = f(i) * f(i+1) * \ldots * f(j-1) * f(j)$$

Batuketa ($+$) eta biderketa ($*$) hurrenez hurren disjuntzioarekin eta konjuntzioarekin parekatuz gero, batukaria —alegia, funtzio batean aldagai baten balioa aldatuz lortzen diren espresioen batuketez eratutako segida— eta biderkaria —antzeko segida biderketekin— kuantifikatzaile existentzialarekin eta unibertsalarekin pareka daitezke, hurrenez hurren. $\sum_{x=i}^{j} f(x)$ espresioan agertzen den x aldagaia, laburdura adierazteko erabiltzen da, baina ez da agertzen batukari hori garatuz lortzen den $f(i)+f(i+1)+\ldots+f(j-1)+f(j)$ espresioan. Beraz, x aldagaia kuantifikatzaileetako aldagai atxikiaren parekoa da. $\sum$ eta $\prod$ sinboloak x aldagai atxikiaren definizio-eremuko informazioarekin hornitzen dira, eta, horren ondoren, x darabilen funtzioa definitzen duen terminoa idazten da. Definizio-eremua hutsa denean (ikusi ditugun adibideetan $i > j$ denean), batukariaren emaitza 0 balioa da (batuketaren elementu neutroa), eta biderkariaren emaitza 1 da (biderketaren elementu neutroa).

Aurrekoen antzeko eragiketa, programak espezifikatzean arras erabilgarria dena, $\mathcal{N}$ *kontaketa*-eragiketa da. $\mathcal{N}x\ P(x)$ terminoak P propietatea zenbat elementuk betetzen duten adierazten du. Esanahi hori adierazteko, x aldagai bat erabiltzen da, eta aldagai hori P propietatea adierazten duen formulan agertuko da libre. Alabaina, $\mathcal{N}$ eragilearen eraginpean jarrita, $P(x)$ formulako x aldagaiaren agerpen guztiak atxikiak izango dira. Kuantifikatzaile existentzialarekin gertatzen den bezala, maiz $D(x) \wedge P'(x)$ egitura erabiltzen da $P(x)$ adierazteko. Adibidez,

$$\mathcal{N}x\ (\ 1 \leq x \leq 7 \wedge x\ \%\ 2 = 0\ )$$

ondo eratutako terminoa da, eta haren balioa 3 da.

Jarraian erakutsiko den bezala, orain arte eman dugun $\mathcal{N}$ kontaketa-eragiketaren definizioa ez da espezifikazio formalean erabiltzeko behar bezain zehatza eta, ondorioz, formalki birdefinituko dugu.

Esate baterako, demagun s egoeran $A(1..3)$ arrayak $(4,2,4)$ balioa duela. Orduan garbi ikusten da

$$\mathcal{N}x\ (\ 1 \leq x \leq 3 \wedge A(x) \geq 4\ )$$

terminoak 2 balioa duela, honako propietatea

$$1 \leq x \leq 3 \wedge A(x) = 4$$

betetzen duten elementuen multzoa $\{1,3\}$ delako. Baina, agian ez dago hain garbi beste termino honek

$$\mathcal{N}x\ (\ \exists i\ (\ 1 \leq i \leq 3 \wedge x = A(i) \wedge x \geq 4\ )\ )$$

2 edo 1 balio duen, izan ere

$$\exists i\ (\ 1 \leq i \leq 3 \wedge x = A(i) \wedge x \geq 4\ )$$

propietatea betetzen duten elementuek $\{4,4\}$ multimultzoa edo $\{4\}$ multzoa osa baitezakete. Definizio zehatz batek anbiguotasun hori argitu egin beharko luke. Hona hemen definizio zehatz hori: $\mathcal{N}x\ P(x)$ terminoaren semantika $\{\ x\ \mid\ P(x)\ \}$ multzoaren kardinalitatea (elementuen kopurua) dela esango dugu. Horren arabera,

$$\mathcal{N}x\ (\ \exists i\ (\ 1 \leq i \leq 3 \wedge x = A(i) \wedge x \geq 4\ )\ )$$

terminoaren balioa $s = \{(A(1),4),(A(2),2),(A(3),4)\}$ egoeran 1 da.

s egoeran $P(x)$ faltsua baldin bada edozein x-tarako, orduan $\mathcal{N}x\ P(x)$ terminoaren balioa 0 izango da, $\{\ x \mid P(x)\ \}$ multzoa hutsa baita eta, horrenbestez, haren kardinalitatea 0. Adibidez,

$$\mathcal{N}x\ (\ 1 \leq x < i \wedge P(x)\ )$$

terminoa hartuz, i aldagaiari 1 baino txikiagoa edo berdina den balio bat egokitzen dion edozein egoeratan termino horren balioa 0 izango da, edozein dela ere P propietatea.

Hona hemen aipatutako azken eragileen bidez adierazitako zenbait propietate:

- *«$A(1..n)$ bektoreko elementu guztien batura bere lehenengo $k \geq 1$ elementuen biderkadura baino handiagoa da»*:

$$1 \leq k \leq n \wedge \sum_{i=1}^{n} A(i) > \prod_{i=1}^{k} A(i)$$

- *«$A(1..10)$ bektoreko lehenengo k elementuen biderkadura x da»*:

$$x = \prod_{i=1}^{k} A(i) \wedge 1 \leq k \leq 10$$

- *«x balioa y aldiz agertzen da $A(1..n)$ bektorean»*:

$$\mathcal{N}i\ (\ 1 \leq i \leq n \wedge A(i) = x\ ) = y$$

Har dezagun orain, x eta y aldagai osoei eta n tamaina konstanteko A bektoreari balioak egokitzen dizkien s egoera bat. Zehazki, izan bedi hurrengo egoera:

$$s = \{(x,3),(y,4),(n,6),(A(1..n),(2,\ 5,\ 3,\ 6,\ 1,\ 8))\}$$

Honako formula hauen balioa s egoeran F da:

- $x \geq y$
- $(x\ \%\ 2 = 0) \leftrightarrow (y\ \%\ 2 = 0)$
- $\forall i\ (\ 1 \leq i \leq n \rightarrow A(i) \leq x + y\ )$
- $\exists i\ (\ 1 \leq i \leq n \wedge x = A(i)\ ) \wedge \exists j\ (\ 1 \leq j \leq 6 \wedge y = A(j)\ )$

Aldiz, beste formula hauek T balioa hartzen dute s egoeran interpretatuta:

- $x+1=y \wedge y>2$
- $\exists z\,(\, z>0 \wedge z+x=y\,)$
- $\forall z\,(\, z>1 \rightarrow z+x>y\,)$
- $\exists i\,(\, 1\leq i\leq n \wedge (\, x=A(i) \vee y=A(i)\,)\,)$
- $\exists i\,(\, 1\leq i\leq n \wedge \forall j\,(\, i<j\leq 6 \rightarrow A(i)>A(j)\,)\,)$ [6]

Formula batean libre agertzen den aldagairen bati aurreko s egoeran ez bazaio baliorik egokitzen (esate baterako, z edo $B(1..m)$ aldagaiak agertuko balira), orduan formula hori, edozein dela ere, indefinituta dago s egoeran, hau da, ez dauka interpretaziorik s-n. Adibidez, honako formula hau s egoeran indefinituta dago:

$$\exists i\,(\, 1\leq i\leq n \wedge B(i)>z\,)$$

Komeni da nabarmentzea, ordea, $\exists z\,(\, z>0 \wedge z+x=y\,)$ formula egiazkoa dela s-n.

Asertzioetako aldagai libreen eta atxikien erabilera programako aldagaien araberakoa da. Asertzioek idatzita dauden puntuan gerta daitezkeen konputazio-egoerak errepresentatzen dituzte. Egoerek programako aldagai guztiei edo batzuei balioak egokitzen dizkiete. Adibidez, bektore bateko elementuen batuketa egiten duen programara eta haren asertzioetara itzuliz:

$\{\, n\geq 1\,\}$
```
  i := 0;
  x := 0;
  while i < n loop
```
$\{\, x=\sum_{j=1}^{i} A(j) \wedge 0\leq i\leq n\,\}$
```
    i := i+1;
    x := x+A(i);
  end loop;
```
$\{\, x=\sum_{j=1}^{n} A(j)\,\}$

asertzioetan erabiltzen diren aldagai libreak ($A(1..n)$, n, i, x) programako aldagaiak dira. Aitzitik, asertzioetako aldagai atxikiak, batukariaren j

6. $i=6$ denean betetzen da.

aldagai atxikia kasurako, ez dira programakoak. Izatez, ez luke zentzurik batukariaren aldagai atxikitzat programako aldagairen bat, demagun i edo x, aukeratzeak. Izan ere, $x = \sum_{x=1}^{i} A(x)$ edo $x = \sum_{i=1}^{i} A(i)$ asertzioak idatzi izan balira, bistakoa da (gutxienez) nahasgarriak liratekeela. Programan agertzen ez den beste edozein aldagai (j, esate baterako), arazorik gabe erabil daiteke aldagai atxiki moduan. Bestalde, asertzio batek programan agertzen ez diren aldagai libreak baldin baditu, indefinituta dago programaren konputazio-egoeretan, eta, ondorioz, errepresentatzen dituen egoerak ez datoz bat programaren konputazioekin. Horrenbestez, programa baten asertzioetan agertzen diren aldagai libre guztiek programako aldagaiak izan behar dute. Gainera, programako aldagai bat asertzio batean agertuko bada, aldagai horrek balio bat izan behar du asertzio horri dagokion programako puntuan (edozein konputaziotan). Aurreko adibideko programan, zentzugabea litzateke $\{\, n \geq 1 \wedge i = 0 \wedge x = 0 \,\}$ jartzea programaren hasierako asertziotzat, i eta x aldagaiak hasieratu gabe daudelako puntu horretan. Hiru enuntziatu hauetan laburbilduko dugu asertzioetako aldagaiei buruz esandakoa:

- Programa bateko asertzioetan agertzen diren aldagai libre guztiek programako aldagaiak izan behar dute.

- Asertzioetako aldagai atxiki guztiek aldagai berriak izan behar dute, hau da, ez dute agertu behar ez asertzioetako aldagai libre gisa ez programako aldagai gisa.

- Programako puntu bati egokitzen zaion asertzioan agertzen diren aldagai libre guztiek balio bat izan behar dute puntu horretako edozein konputazio-egoeratan.

Aurrerantzean $s(\varphi)$ notazioa erabiliko dugu s egoeran φ formulak duen balioa izendatzeko. Adibidez, $s(\, \exists z \,(\, z > 0 \wedge z + x = y \,)\,) = \mathsf{T}$ idatziko genuke s egoera $\{(x,3),(y,4),(n,6),(A(1..n),(2,\ 5,\ 3,\ 6,\ 1,\ 8))\}$ balitz.

2.3. ARIKETAK: LEHEN MAILAKO ASERTZIOAK

1. Adierazi zein diren formula zuzenak baieztapen bakoitzarentzat (bat baino gehiago izan daiteke zuzena). Horretaz gain, egokitu baieztapen horietako bakoitzari parametro bezala formulako aldagai libreak dituen predikatu bat (lehenengo baieztapenean dagoeneko predikatua agertzen da):

1.1. $handiena(A(1..n), x) \equiv A(1..n)$ bektoreko elementu handiena x da.

$a)\ \exists i\ (\ 1 \le i \le n \wedge A(i) = x\ ) \wedge \forall j\ (\ 1 \le j \le n \rightarrow x \ge A(j)\ )$ [ ]
$b)\ 1 \le i \le n \wedge A(i) = x \wedge \forall j\ (\ 1 \le j \le n \rightarrow x \ge A(j)\ )$ [ ]
$c)\ \exists i\ (\ 1 \le i \le n \wedge A(i) = x\ ) \wedge$ [ ]
$\forall j\ (\ 1 \le j \le n \rightarrow A(i) \ge A(j)\ )$

1.2. _____ $\equiv A(1..n)$ bektorean ez dago errepikatutako elementurik.

$a)\ \forall i\ (\ 1 \le i \le n \rightarrow \forall j\ (\ i < j \le n \rightarrow A(i) \ne A(j)\ )\ )$ [ ]
$b)\ \forall i\ (\ 1 \le i \le n \rightarrow \mathcal{N} j\ (\ 1 \le j \le n \wedge A(i) = A(j)\ ) = 1\ )$ [ ]
$c)\ \forall i\, \forall j\ (\ 1 \le i \le n \wedge 1 \le j \le n \wedge i \ne j \rightarrow A(i) \ne A(j)\ )$ [ ]

1.3. _____ $\equiv A(1..n)$ bektorean lehenengo elementua baino handiagoak diren k elementu daude.

$a)\ k = \mathcal{N} i\ (\ 1 < i \le n \wedge A(i) > A(1)\ )$ [ ]
$b)\ k = \mathcal{N} i\ (\ 1 \le i \le n \wedge A(i) > A(1)\ )$ [ ]
$c)\ A(1) < \mathcal{N} i\ (\ 1 \le i \le n \wedge A(i) = k\ )$ [ ]

1.4. _____ $\equiv A(1..n)$ bektoreko elementuak goranzko ordenan daude.

$a)\ \forall i\ (\ 1 \le i \le n \rightarrow A(i) < A(i+1)\ )$ [ ]
$b)\ \forall i\ (\ 1 \le i < n \rightarrow A(i) < A(i+1)\ )$ [ ]
$c)\ \forall i\ (\ 1 \le i \le n \rightarrow A(i-1) < A(i)\ )$ [ ]

2. Bete hutsuneak (____) honako formula hauetan:

2.1. $A(1..n)$ bektorean ez daude bi zero jarraian.

$$\forall i\ (\ 1 \_ i \_ n \rightarrow (\ A(i) \ne 0 \vee A(i+1) \ne 0\ )\ )$$

2.2. x balioa $A(1..n)$ bektorean i posizioa baino lehenago agertzen al den adierazten du *badago* izeneko aldagai boolearrak.

$$(\ \_\_\_\_\_ \leftrightarrow \exists j\ (\ 1 \le j < i \wedge A(j) = x\ )\ ) \wedge (\ 1 \le i \le n\ )$$

2.3. $A(1..n)$ bektoreko $A(i..j)$ sekzioko elementuen batura x da.

$$(1 \leq i \leq j \leq n) \wedge \_ = \sum_{k=i}^{j} A(k)$$

2.4. $B(1..n)$ bektorea $A(1..n)$ bektoreko osagaiak eskuinera posizio bat zirkularki errotatuz lortzen dena da.

$$\forall i (2 \leq i \leq n \rightarrow B(i) = \_\_) \wedge B(1) = \_\_$$

3. Idatzi euskaraz honako formula hauen esanahia:

3.1. $\forall i (2 \leq i \leq n \rightarrow A(1) \neq A(i))$

3.2. $\exists i (i > 0 \wedge x * i = z)$

3.3. $1 \leq ind \leq 6 \leq n \wedge \forall j (6 < j \leq n \rightarrow A(ind) \geq A(j))$

3.4. $\forall j \forall k (1 \leq j \leq n \wedge 1 \leq k \leq m \wedge j \neq k \rightarrow A(j) \neq B(k))$

3.5. $\mathcal{N} i (1 \leq i \leq k \wedge A(i) = 4) = 2$

4. Adierazi honako baieztapen hauek lehen mailako formulen bidez eta egokitu formula horietako bakoitzari parametro bezala formulako aldagai libreak dituen predikatu bat:

4.1 x balioa $A(1..n)$ bektorean lehenengo aldiz k posizioan agertzen da.

4.2 z balioa 1 zenbakiarekin hasten den ondoz ondoko zenbaki osoen sekuentzia bat osatzen duten zenbakien batura da. Adibidez, $10 = 1+2+3+4$.

4.3 y baino txikiagoa den 2ren berretura handiena x da.

4.4 x balioa $A(1..n)$-ren edozein osagai baino txikiagoa edo berdina da.

4.5 $A(1..n)$ bektoreak zehazki k zero dauzka.

2.4. ALDIBEREKO ORDEZKAPENA

Programako aldagaien balioak, alegia, egoerak, sarrerako datuetatik abiatu eta programak egindako konputazioetan zehar eginiko esleipenen bidez aldatzen dira. Egoera-aldaketa horiek asertzioen bidez adierazten dira: asertzio batek esleipenaren aurreko egoera errepresentatzen du, eta beste asertzio batek esleipenaren ondorengo egoera. Esleipenek eragindako egoera-aldaketei dagokien eragiketa *aldibereko ordezkapena* deritzo.

Formula bateko aldagai bat termino batekin ordezkatzen duen eragiketa honako era honetan definitzen da formalki: izan bitez φ formula, x aldagaia eta t terminoa, φ formulan x-ren agerpen libre guztien ordez t terminoa jarriz lortzen den formula φ_x^t bezala adieraziko dugu. Adibidez,

$$(x \geq 0 \wedge x + z < 5)_x^v \equiv (v \geq 0 \wedge v + z < 5)$$

φ_x^t kalkulatzean, t terminoan agertzen den aldagairen bat φ-ren kuantifikatzaileren batean ere agertzen bada eta, gainera, x aldagaia kuantifikatzaile horren esparruan baldin badago, orduan *izen-atzematea* edo *izen-talka* gertatzen da. Talka horren eraginez ezin da baieztatu φ formulak x-ri buruz dioena eta φ_x^t formulak t terminoari buruz dioena baliokideak direnik. Esate baterako,

$$(\forall i \, (\, 1 \leq i \leq n \, \rightarrow \, A(i) = x \,))_x^{i-1}$$

ordezkapenean, $\forall i$ kuantifikatzaileak $i-1$ terminoko i aldagaia atzematen du. Atzemate edo talka hori dela-eta, ordezkapenaren emaitza den formula honetan

$$\forall i \, (\, 1 \leq i \leq n \, \rightarrow \, A(i) = i - 1 \,)$$

$i-1$ terminoari buruz esaten denak ez du zerikusirik hasierako formulak

$$\forall i \, (\, 1 \leq i \leq n \, \rightarrow \, A(i) = x \,)$$

x-ri buruz esaten duenarekin. Oro har, izenaren atzemateak eragiten duen arazoa saihesteko, nahikoa da ordezkapena egin aurretik hasierako formulan kuantifikatzailearen aldagaia berrizendatzearekin, azken batean, horrek ez baitu formularen semantika aldatuko. Berriro adibidera joz, i aldagai atxikiari k izena jarriz gero, egin beharreko ordezkapena honako hau izango da:

$$(\forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = x \,))_x^{i-1}$$

Ordezkapena egindakoan honako formula hau lortuko litzateke:

$$\forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = i - 1 \,)$$

eta, orain bai, formula horrek $i-1$ terminoari buruz baieztatzen duena, eta beste honek

$$\forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = x \,)$$

x aldagaiari buruz dioena, berdinak dira.

Asertzio batean x aldagaia t terminoaren ordez jartzeak `x := t` aginduaren egiaztapenarekin lotura estua duenez, `t` terminoan agertzen diren aldagaiak programako aldagaiak dira. Beraz, asertzioek aurreko atalean emandako jarraibideak betetzen badituzte, bereziki aldagai atxikiak ez badira programan agertzen, programaren egiaztapenean ez da gertatuko izenaren atzemate

arazorik. Hori dela-eta, atal honen bukaeran n aldagaietarako ($n \geq 1$) aldibereko ordezkapena definitzean, terminoen aldagaiak eta kuantifikatzaileenak disjuntuak direla suposatuko dugu.

Hona hemen aldagaiak terminoekin nola ordezkatzen diren erakusten duten beste adibide batzuk:

- $(x^3 \geq 0 \wedge x + z < 5)_x^{A(i)} \equiv (A(i)^3 \geq 0 \wedge A(i) + z < 5)$
- $(\prod_{i=1}^{k} A(i))_k^{k+1} \equiv \prod_{i=1}^{k+1} A(i)$
- $(\exists i (i > x \wedge bikoitia(i)))_x^{x+y} \equiv \exists i (i > x + y \wedge bikoitia(i))$

Ordezkapena, aldi berean n aldagai n terminorekin ordezkatzen dituen eragiketa bezala ere defini daiteke oro har. Definizio formalak atzematerik ez gertatzea eskatzen du baldintza bezala:

> Izan bitez φ formula, $n \geq 1$, binaka desberdinak diren $x_1, \ldots, x_n$ aldagaiak eta φ formulan kuantifikatuta ez dauden aldagaiak bakarrik izan ditzaketen $t_1, \ldots, t_n$ terminoak. Hori horrela izanda, aldi berean $i \in \{1, \ldots, n\}$ balio bakoitzeko φ formulan x_i-ren agerpen libre guztien ordez t_i terminoa jarriz lortzen den formula $\varphi_{x_1,\ldots,x_n}^{t_1,\ldots,t_n}$ bezala adieraziko dugu.

$n = 1$ deneko kasu partikularrean, definizio horrek aurretik emandakoa hobetzen du, izen-atzematearen arazoa saihesten duen baldintza gehituz. Liburuan zehar, ordezkapenak erabiltzen direnean atzematerik ez dagoela suposatuko da.

Honako adibide honek aldibereko ordezkapen bat erakusten du:

$$(x^3 \geq 0 \wedge 0 < x + i < n)_{x,i}^{A(i),i+1} \equiv (A(i)^3 \geq 0 \wedge 0 < A(i) + i + 1 < n)$$

Ordezkapenak sekuentzialki egingo balira, honako hau izango genuke:

$$((x^3 \geq 0 \wedge 0 < x + i < n)_x^{A(i)})_i^{i+1} \equiv (A(i+1)^3 \geq 0 \wedge 0 < A(i+1) + i + 1 < n)$$

2.5. BALIOZKO INPLIKAZIOAK

Programak egiaztatzean sortzen diren egitekoen artean, emandako inplikazio bat edozein egoeratan egiazkoa dela frogatzea da nabarmenenetakoa. Inplikazio bat egiazkoa baldin bada, baliozkoa dela esaten da. Beraz, $\varphi \to \psi$ inplikazioa baliozkoa da, baldin eta soilik baldin $s(\varphi \to \psi) = \mathsf{T}$ bada edozein s egoeratan. Atal honetan, inplikazio bat edozein egoeratan betetzen al den erabakitzeko erabiliko dugun irizpidea aurkeztuko dugu. Inplikazioaren egia-taulak erabiliz honako hau froga daiteke:

$s(\varphi \to \psi) = \mathsf{T}$ edozein s-tarako baldin eta soilik baldin

$$\{\, s \mid s(\varphi) = \mathsf{T} \,\} \subseteq \{\, \mathsf{s} \mid \mathsf{s}(\psi) = \mathsf{T} \,\}$$

$\{\, s \mid s(\varphi) = \mathsf{T} \,\} \subseteq \{\, \mathsf{s} \mid \mathsf{s}(\psi) = \mathsf{T} \,\}$ betetzen denean φ formula ψ formula baino gogorragoa dela esango dugu, edo ψ formula φ formula baino ahulagoa dela. Intuitiboki, formula batek egoerek bete beharrekoa zenbat eta gehiago murrizten duen, orduan eta gogorragoa izango da formula hori. Adibidez, $lehena(x) \wedge x > 3$ formula $\neg(x \,\%\, 3 = 0)$ baino gogorragoa da:

$$\{\{x = 5\}, \{x = 7\}, \ldots\} \subseteq \{\{x = 1\}, \{x = 2\}, \{x = 4\}, \{x = 5\}, \{x = 7\}, \ldots\}$$

Azken buruan, $\varphi \to \psi$ baliozkoa da, baldin eta soilik baldin φ gogorragoa bada ψ baino.

Muturreko kasuei erreparatuta, $True$ da formularik ahulena (edozein s egoeratan egiazkoa baita) eta $False$ da formularik gogorrena (faltsua da edozein dela ere s egoera). Ondorioz, $\varphi \to True$ eta $False \to \varphi$ baliozkoak dira edozein φ formulatarako, hau da, lege logikoak dira. Programen egiaztapenean eta arrazoiketa logikoa eskatzen duten bestelako arloetan oso erabilgarriak diren beste lege logiko asko ere baliozko inplikazioak dira, eta era horretan froga daitezke. Hona hemen adibide batzuk: $\varphi \to (\varphi \vee \psi)$, $(\varphi \wedge \psi) \to \varphi$, $\varphi^t_x \to \exists x\, \varphi$, eta $(\forall x\, \varphi) \to \varphi^t_x$. Programen egiaztapenean sarritan erabiltzen diren erako baliozko inplikazioak dira honako beste adibide hauek:

- $x > 0 \;\to\; x \geq 0$
- $(x = 0 \wedge y = 1) \;\to\; y = z^x$
- $k = 0 \;\to\; \forall j\, (\, 1 \leq j \leq k \;\to\; A(j) = 0\,)$
- $(\, k = 1 \;\wedge\; A(1) = 0\,) \;\to\; \forall j\, (\, 1 \leq j \leq k \;\to\; A(j) = 0\,)$
- $lehena(2) \;\to\; \exists x\, (\, lehena(x)\,)$
- $(\, 5 \leq n \wedge A(5) > 0\,) \;\to\; \exists j\, (\, 1 \leq j \leq n \;\wedge\; A(j) > 0\,)$
- $(\, 1 \leq i \leq n \wedge A(i) > 0\,) \;\to\; \exists j\, (\, 1 \leq j \leq n \;\wedge\; A(j) > 0\,)$
- $\forall i\, (\, 1 \leq i \leq n \;\to\; bikoitia(A(i))\,) \;\to\; (\, 7 \leq n \;\to\; bikoitia(A(7))\,)$
- $\forall j\, (\, 1 \leq j \leq n \;\to\; A(j) > 0\,) \;\to\; (\, 1 \leq i \leq n \;\to\; A(i) > 0\,)$

2.6. PROGRAMEN AURRE-ONDOETAKO ESPEZIFIKAZIO FORMALA

Liburu honetan, *aurre-ondoetako* formatua erabiliko dugu programak espezifikatzeko. P programa baten aurre-ondoetako espezifikazioa φ eta ψ izendatuko ditugun bi asertzioren bidez ematen da. φ asertzioari *aurreko baldintza* (edo *aurrebaldintza*) esaten zaio, eta ψ asertzioari *ondoko baldintza* (edo *postbaldintza*). Programa programazio-lengoaia batean idatzita egongo da, eta asertzioak, aldiz, lehen mailako logikan idatzita egongo dira. Hoare-ren hirukote bezala ezagutzen den

$$\{\, \varphi \,\}\ \mathtt{P}\ \{\, \psi \,\}$$

erako baieztapenak P programa φ betetzen den egoera batean hasten bada, orduan, bukatzen bada, ψ betetzen den egoera batean bukatuko dela dio. P programaren abiapuntu izan daitezkeela onartzen dugun egoera guztiak errepresentatzen ditu aurrebaldintzak. Beraz, ahal den formularik *ahulena* definitzea komeni da, hau da, programaren edozein konputazio zuzena izan dadin datuek bete beharreko *gutxieneko baldintza*. Aldiz, postbaldintzak datuen eta emaitzen arteko erlazioa adierazten du eta emaitza onargarri guztiak errepresentatzen dituen ahalik eta formula *gogorrena* definitu behar da, betiere sarrerako datuek aurrebaldintza bete behar dutela kontuan hartuta. Esate baterako, bi zenbaki arrunten (zenbaki negatiboak ez dira onartuko beraz) zatidura osoa eta hondarra kalkulatzen dituen programa baten espezifikazioa honela defini daiteke:

$\{\, y > 0 \wedge x \geq 0 \,\}$
```
begin
  z := 0;
  h := x;
  while h >= y loop
    z := z+1;
    h := h-y;
  end loop;
end;
```
$\{\, x = z * y + h \wedge y > h \geq 0 \wedge z \geq 0 \,\}$

Aurreko baldintza aski ahula da datu onargarri guztiak onartzeko; bakarrik zatitzailea zero izatea eragozten du. Postbaldintza aski gogorra da onartezinak liratekeen emaitzak baztertzeko. Aurrebaldintza bezala ($y > 1 \wedge x \geq 0$) erabili izan bagenu, gogorregia izango litzateke, programak edozein zenbaki arrunt eta 1 zenbakiaren arteko zatiketa osoa ere kalkulatzen duelako.

Bestetik, aurrebaldintza bezala ($\neg(y=0) \wedge x \geq 0$) erabili izan bagenu, ahulegia izango litzateke, aurrebaldintzak zatitzaile negatiboak onartu arren, programak ez baitu ondo kalkulatzen zatitzailea zenbaki negatiboa denean. Postbaldintzari dagokionez, ($x = z*y+h$) formula ahulegia izango litzateke; oso erraza da postbaldintza betetzen duen baina sarrerako datuen zatidura osoa ondo adierazten ez duen egoera bat aurkitzea. Esate baterako, $\{(x,31),(y,7),(c,3),(r,10)\}$ egoera. Azkenik, ($x = z*y+h \wedge h = y-1$) asertzioa gogorregia izango litzateke; programak sortuko lituzkeen egoera batzuek (adibidez, $\{(x,31),(y,7),(c,4),(r,3)\}$) ez lukete postbaldintza hori beteko.

2.7. ARIKETAK: PROGRAMEN AURRE-ONDOETAKO ESPEZIFIKAZIO FORMALA

1. Eman aurre-ondoetako espezifikazioa honako ekintza bakoitzarentzat:

   1.1. $B(1..n)$ bektorea $A(1..n)$ bektoreko osagaien permutazioa den ala ez erabaki.

   1.2. $A(1..n)$ bektoreko osagaiak goranzko ordenan ipini.

   1.3. $A(1..n)$ bektore batek zerra forma duen ala ez erabaki.

   $A(1..n)$ bektoreak zerra forma du, baldin eta soilik baldin jarraian dauden edozein hiru elementu hartuta, lehenengo bien ordena (gorakorra/beherakorra) ondorengo bien ordenaren aurkakoa bada.

   Adibidez,

   $A(1..7) = (5,2,7,4,8,3,6)$ zerra formako bektorea da,
   $A(1..5) = (3,4,2,1,6)$ ez da zerra formakoa.

2. Idatzi aurre-ondoetako espezifikazio formala honako bikote hauek osatzen dituzten ekintzentzat. Ekintza-bikote bakoitzean, ekintza batek aldi berean sarrera eta irteera den parametro bat du gutxienez, eta beste ekintzak ez du sarrera-irteera parametrorik. Ekintzen deskripzio informal horietan guztiz zehaztu gabe egon daitezkeen alderdiek sor ditzaketen zalantzak ebatzi programaren funtzionalitatea ahal den gutxiena murriztuz.

   2.1.
   - Bi zenbaki emanda, bien arteko biderkadura kalkulatu.
   - Bi zenbaki emanda, aldatu datuetako bat (aldatu gabe gelditu beharko duen) beste datuarekin biderkatuz.

2.2.
- Lehena ez den zenbaki bat emanda, haren faktore lehen txikiena itzuli.
- Lehena ez den zenbaki bat emanda, haren faktore lehen txikienarekin ordezkatu.

2.3.
- x eta y zenbakiak eta b balio boolearra emanda, x lehena eta b egiazkoa, edo y lehena eta b faltsua al diren erabaki.
- x eta y zenbakiak eta b balio boolearra emanda, b aldagaian bertan honako proposizio honen egia-balioa itzuli: «x lehena da eta b-ren hasierako balioa egiazkoa, edo y lehena da eta b-ren hasierako balioa faltsua».

2.8. BIBLIOGRAFIA-OHARRAK

Konputazio-egoeren multzoak formula logikoen bidez, hots, asertzioen bidez, errepresentatzeko ideia konputazio-zientzien hastapenetatik dator. Asertzioak (bereziki, aurreko eta ondoko baldintzak) erabiltzea programen zuzentasunaz arrazoitzeko beharrarekin estuki lotuta dago. Erreferentziarik goiztiarrenak [46, 89] 1940ko hamarkadaren bukaeran agertu ziren. Geroztik, 60ko hamarkadaren hondarrean, R. W. Floyd [40] eta C. A. R. Hoare [52] autore ezagunen lanak argitaratu zirelarik, programen zuzentasunaz formalki arrazoitzeko euskarri gisa lehen mailako logikan idatziriko asertzioen erabilerak benetako indarra eta zabalkundea hartu zuen. [25] lanean asertzioen jatorriari eta bilakaerari buruzko narrazio historiko bikaina aurki daiteke.

Programen egiaztapena lantzen duen edozein liburuk eskaintzen dio atal bat lehen mailako logikaren aurkezpenari eta programen konputazio-egoerak errepresentatzeko duen erabilerari. Liburu horien artean, adibide interesgarriak dituzten honako irakurgai osagarri hauek nabarmenduko genituzke, besteak beste: [2, 2.2. atala], [9, 1. eta 2. kapituluak], [13, I. kapitulua], [36, 2. kapitulua], [48, 1., 2., 3. eta 4. kapituluak], [65, 2. kapitulua] eta [68, 1. kapitulua].

Eiffel programazio-lengoaia diseinatu zuenean B. Meyer-ek [71, 72] *kontratu* deitu zien aurre-ondoetako espezifikazioei. Eiffel, objektuei orientatutako lengoaia, aitzindaria izan zen asertzioen erabileran. Gaur egun, programazio-lengoaia askok (adibidez, Euclid, Gypsy, Alphard, etab.) berenganatu dituzte asertzioak, eta, programen exekuzioan zehar edo arrazoitzaile automatikoen bidez, estatikoki, asertzioak betetzen direla froga daiteke. Halaber, fidagarriagoak diren programak idazteko helburuarekin, asertzioen era bateko edo besteko tratamendua ahalbidetzen duten aurreprozesadore gero eta sendoagoz hornitzen dituzte lengoaiarik erabilienak ere (adibidez, Basic, Ada, C, C++, C♯, Java, TCL eta abar).

2.9. ARIKETAK: PROGRAMEN ESPEZIFIKAZIO FORMALA

1. Adierazi zein diren formula zuzenak baieztapen bakoitzarentzat (bat baino gehiago izan daiteke zuzena). Horretaz gain, egokitu baieztapen horietako bakoitzari parametro bezala formulako aldagai libreak dituen predikatu bat (lehenengo baieztapenean dagoeneko predikatua agertzen da):

   1.1. $lehena(x) \equiv x$ zenbaki lehena da.

   a) $x > 1 \wedge \forall i\,(\,1 < i < x \rightarrow x \,\%\, i \neq 0\,)$ [ ]

   b) $x > 1 \wedge \exists i\,(\,1 < i < x \wedge x \,\%\, i \neq 0\,)$ [ ]

   c) $\forall i\,(\,1 < i < x \rightarrow x \,\%\, i \neq 0\,)$ [ ]

   1.2. _____ $\equiv A(1..n)$ bektoreko i-garren elementuaren ondoren dauden guztiak zeroak dira.

   a) $1 \leq i \leq n \wedge \forall j\,(\,i < j \leq n \rightarrow A(j) = 0\,)$ [ ]

   b) $0 < i \leq n \wedge \neg\exists j\,(\,i < j \leq n \wedge A(j) \neq 0\,)$ [ ]

   c) $\exists i\,(1 \leq i \leq n \wedge \forall j\,(\,i < j \leq n \rightarrow A(j) = 0\,)\,)$ [ ]

   1.3. _____ $\equiv A(1..n)$ bektorean x baldin badago, bakarrik $i..j$ sekzioan egongo da.

   a) $1 \leq i \leq j \leq n \wedge (\,\exists k\,(\,i \leq k \leq j \wedge x = A(k)\,) \vee \forall k\,(\,i \leq k \leq j \rightarrow x \neq A(k)\,)\,)$ [ ]

   b) $\exists k\,(\,i \leq k \leq j \wedge x = A(k)\,) \vee \forall k\,(\,1 \leq k \leq n \rightarrow x \neq A(k)\,)$ [ ]

   c) $\neg\exists k\,(\,1 \leq k < i \wedge x = A(k)\,) \wedge \neg\exists k\,(\,j < k \leq n \wedge x = A(k)\,)$ [ ]

   1.4. _____ $\equiv A(1..n)$ bektorean elkarren ondokoak diren bi elementuren arteko diferentziarik handiena (balio absolutuan) k da.

   a) $\exists i\,(\,1 < i \leq n \wedge |A(i) - A(i-1)| = k\,) \wedge \forall j\,(\,1 < j \leq n \rightarrow |A(j) - A(j-1)| \leq k\,)$ [ ]

   b) $\exists i\,(\,1 < i \leq n \wedge |A(i) - A(i-1)| = k\,) \wedge \forall j\,(\,1 < j \leq n \wedge i \neq j \rightarrow |A(j) - A(j-1)| \leq k\,)$ [ ]

   c) $\exists i\,(\,1 < i \leq n \wedge |A(i) - A(i-1)| = k\,) \wedge \forall j\,(\,1 < j \leq n \wedge i \neq j \rightarrow |A(j) - A(j-1)| < k\,)$ [ ]

2. Bete hutsuneak (____) honako formula hauetan:

   2.1. $A(1..n)$ bektoreko $A(i..j)$ sekzioan y aldiz agertzen da x.

   $$agerpen\_kopurua(x, A(1..n), i, j, y) \equiv$$

   $$1 \leq i \leq j \leq n \wedge \_\_ = \mathcal{N}k\ (\ i \leq k \leq j\ \wedge\ A(k) = \_\_\ )$$

   2.2. $B(1..n)$ bektorea $A(1..n)$ bektorearen permutazioa da.

   $$permutazioa(A(1..n), B(1..n)) \equiv$$

   $$\forall i\ (\ 1 \leq i \leq n\ \rightarrow\ \mathcal{N}j\ (\ 1 \leq j \leq n\ \wedge\ A(i) = \_\_\ )\ = \mathcal{N}k\ (\ 1 \leq k \leq n\ \wedge\ A(i) = \_\_\ )\ )$$

   2.3. $A(1..n)$ bektoreko elementu bakoitza bi aldiz agertzen da.

   $$\forall i\ (\ 1 \leq i \leq n\ \rightarrow\ agerpen\_kopurua(\_\_\_\_\_)\ )$$

   2.4. $A(1..n)$ bektoreak ez dauka errepikatutako elementurik.

   $$\forall i\ (\ 1 \leq i \leq n\ \rightarrow\ agerpen\_kopurua(\_\_\_\_\_)\ )$$

3. Idatzi euskaraz honako formula hauen esanahia:

   3.1. $\exists i\ (\ i > 0\ \wedge\ x/i = z\ )$

   3.2. $\forall i\ (\ 1 \leq i \leq n\ \rightarrow\ A(i)\ \%\ 2 = 0\ )$

   3.3. $\forall i\ (\ 10 \leq i \leq 20\ \rightarrow\ A(5) \neq A(i)\ )$

   3.4. $\exists i\ (\ 1 \leq i \leq 6\ \wedge\ \forall j\ (\ 6 < j \leq n\ \rightarrow\ A(i) \geq A(j)\ )\ )\ \wedge\ 6 \leq n$

   3.5. $A(1..n) = (a_1, ..., a_n)\ \wedge\ a_1 = a_n$

4. Adierazi honako baieztapen hauek lehen mailako formulen bidez eta egokitu formula horietako bakoitzari parametro bezala formulako aldagai libreak dituen predikatu bat:

   4.1. x da $A(1..n)$ bektoreko elementu txikiena.

   4.2. $A(1..n)$ bektoreko $A(i..j)$ sekzioan elementu handiena x da.

   4.3. Elkarren jarraian dauden k zeroz osatutako sekzio bakarra du $A(1..n)$ bektoreak.

4.4. i posizioak $A(1..n)$ bektorea bitan banatzen du: $A(i)$ baino txikiagoak diren balioak ezkerretik daude eta $A(i)$ baino handiagoak edo berdinak diren balioak eskuinetik daude.

4.5. $A(1..n)$ bektoreko elementu bikoiti guztiak beraien ondorengo elementu guztien baturaren berdinak dira.

4.6. $A(1..n)$ bektorean batekoz (bakarrik batekoz) osatutako sekuentziarik luzeena i posizioan hasten da, eta sekuentzia horren luzera y da.

4.7. $A(1..n)$ bektoreak bakarrik digituak ditu eta x zenbaki arrunta errepresentatzen du. Adibidez, $A(1..5) = (7, 3, 9, 8, 1)$ bektoreak 73981 zenbakia errepresentatzen du.

4.8. $A(1..n)$ bektoreak bakarrik digituak ditu eta kapikua den zenbaki bat errepresentatzen du.

4.9. $A(1..n)$ bektorea $B(1..m)$-ren azpibektorea da.

4.10. k da $A(1..n)$ bektorean hertsiki gorakorra den hasierako sekziorik luzeenaren luzera.

4.11. $A(1..n)$ bektorean elkarren jarraian dauden zeroz osatutako bikoteen kopurua z da.

4.12. $A(1..n)$ bektorearen osagaiak 2ren berreturak dira (ez dute elkarren jarraiak izan beharrik).

4.13. $A(1..n)$ bektorean elkarren ondoko azken bi osagai berdinen indizeak k eta $k+1$ dira.

4.14. $A(1..n)$ bektoreak k posizioa baino lehenago ez du errepikatutako osagairik.

4.15. $A(1..n)$ bektorean x-ren agerpen guztiak hasieran daude (ezkerrean) eta elkarren jarraian.

4.16. $A(1..n)$ bektoreak gutxienez elkarren ondoan dauden bi osagai desberdin ditu.

4.17. $A(1..n)$ bektoreak bakarrik elkarren ondoan dauden bi osagai desberdin ditu.

5. 2.9. atal honetan eta 2.3. atalean definitutako predikatuak erabili honako baieztapen hauek formalizatzeko:

5.1. x eta z elkarren segidako zenbaki lehenak dira. (2.9. atal honetako 1.1. ariketa erabiliz)

5.2. $T(1..n)$ bektoreko elementu handiena k posizioan dago. (2.3. ataleko 1.1. ariketa erabiliz)

5.3. $B(1..n)$ bektorea $A(1..n)$ bektoreko elementuak goranzko ordenan ipiniz lortzen den bektorea da. (2.9. atal honetako 2.2. ariketa erabiliz eta 2.3. ataleko 1.4. ariketa erabiliz)

5.4. $B(1..n)$ bektoreko osagai bakoitzak $A(1..n)$ bektorean posizio berean dagoen osagaia A-n zenbat aldiz agertzen den adierazten du. (2.9. atal honetako 2.1. ariketa erabiliz)

5.5. $A(1..n)$ bektorean gehien agertzen den elementua x da. (2.9. atal honetako 2.1. ariketa erabiliz)

5.6. $A(1..n)$ bektorean ondoz ondokoak diren n zenbaki lehen daude eta gainera goranzko ordenan agertzen dira (ez daukate lehenengo n zenbaki lehenak izan beharrik). (2.9. atal honetako 5.1. ariketa erabiliz)

5.7. $C(1..m)$ bektoreko elementu txikienaren indizea i da. (2.3. ataleko 4.4 ariketa erabiliz)

5.8. $C(1..m)$ bektoreak ez du batura bezala 0 ematen duen sekziorik. (2.3. ataleko 2.3. ariketa erabiliz)

6. Zenbaki arrunt baten *faktore (biderkagai) lehenak* zenbaki hori zatitzen duten zenbaki lehenak dira. Bestalde, p zenbakia n-ren faktore lehena baldin bada, p^b balioa n-ren zatitzailea izatearen baldintza betetzen duen b berretzaile handienari p-ren n-rekiko *anizkoiztasun-balioa* deitzen zaio. Zenbaki baten *faktorizazioa* honela ematen da: zenbaki horren faktore lehen guztiez osatutako zerrenda eta zerrenda horretako osagai bakoitzari dagokion anizkoiztasun-balioaren bidez. Aritmetikaren oinarrizko teoremak dio zenbaki arrunt guztiek faktorizazio bakarra dutela.

   Adibideak:

   - 6 zenbakiaren faktore lehenak 2 eta 3 dira. Faktore horien 6rekiko anizkoiztasun-balioa 1 da ($6 = 2^1 * 3^1$); izan ere, 1 baino handiagoa den beste edozein b berretzaile hartuz, 2^b eta 3^b ez dira 6ren zatitzaileak izango.
   - 5 zenbakiak faktore lehen bakarra du: 5 zenbakia bera (lehena delako). Faktore bakar horren anizkoiztasun-balioa 1 da.
   - 200 zenbakiak bi faktore lehen ditu: 2 eta 5. Faktore horien anizkoiztasun-balioak 3 eta 2 dira hurrenez hurren ($200 = 2^3 * 5^2$).
   - 2, 4, 8, 16, ... zenbakiek faktore lehen bakarra dute: 2 ($2 = 2^1$, $4 = 2^2$, $8 = 2^3$, $16 = 2^4$, ...).

Idatzi honako propietate hauek adierazten dituzten formulak lehen mailako logika erabiliz. Horretarako, *lehena*(n) predikatua eta (komeni denean) aurretik definitutako beste predikatuak ere erabil daitezke:

6.1. p zenbakia n-ren faktore lehena da.

6.2. x eta y zenbakiek ez dute faktore lehenik partekatzen.

6.3. $A(1..n)$ bektoreak x zenbakiaren faktorizazioa errepresentatzen du ($x = 200$ baldin bada, $A = (2, 2, 2, 5, 5)$ izan daiteke).

6.4. $A(1..n)$ bektoreko $A(1..k)$ sekzioan x zenbakiaren faktore lehen guztiak daude (errepikapenik gabe), eta $B(1..n)$ bektoreko $B(1..k)$ sekzioan faktore lehen horien anizkoiztasun-balioak daude (hau da, $B(i)$ elementua $A(i)$ faktore lehenaren anizkoiztasun-balioa da).

6.5. x zenbakiaren faktore lehen guztien batuketa z da (errepikapenak ere kontuan hartuz). Batuketa horretan, faktore lehen bakoitzaren errepikapen kopurua bere anizkoiztasun-balioaren berdina da (esate baterako, $x = 200$ baldin bada, $z = 2+2+2+5+5 = 16$ izango da).

7. Idatzi aurre-ondoetako espezifikazio formala honako bikote hauek osatzen dituzten ekintzentzat. Ekintza-bikote bakoitzean, ekintza batek aldi berean sarrera eta irteera den parametro bat du gutxienez, eta beste ekintzak ez du sarrera-irteera parametrorik. Ekintzen deskripzio informal horietan guztiz zehaztu gabe egon daitezkeen alderdiek sor ditzaketen zalantzak ebatzi programaren funtzionalitatea ahal den gutxiena murriztuz.

7.1.
- Sarrerako bektorearen alderantzizkoa itzuli bektorean bertan.
- Sarrerako bektorearen osagaiak beste bektore batean kopiatu alderantzizko ordenan.

7.2.
- Sarrerako bektorea berrordenatu, bertako x-ren agerpen denak bektorearen bukaeran (eskuinean) ipiniz.
- Sarrerako bektorearen osagaiak beste bektore batean kopiatu, sarrerako bektore horretan agertzen diren x denak irteerako bektorearen bukaeran ipiniz.

7.3.
- Sarrerako bektorean x balioa zenbat aldiz agertzen den kontatu.
- Sarrerako bektoreko azken posizioan utzi x balioa bektorean bertan zenbat aldiz agertzen den.

8. Idatzi honako programa hauen aurre-ondoetako espezifikazio formala (lehenengoak zenbaki arrunt baten faktoriala kalkulatzen du, eta bigarrenak bi zenbaki arrunten zatitzaile komunetako handiena):

8.1.
```
f := 1;
t := x;
while t >= 1 loop
  f := f*t;
  t := t-1;
end loop;
```

8.2.
```
z := x;
while y /= 0 loop
  z := x % y;
  x := y;
  y := z;
end loop;
```

3. Programa iteratiboen egiaztapena

Softwarea garatzeko erabiltzen diren metodoen helburu garrantzitsuenetakoa da bermatzea sortzen diren programek aurretik ezarritako helburuak betetzen dituztela, hau da, programak zuzenak edo errorerik gabekoak direla. Zuzentasuna da programek bete behar duten ezaugarri nagusia, funtsezkoa baita ezaugarri hori. Programazio-erroreak aurkitzeko hainbat teknika daude. Teknika horiek bi taldetan sailka daitezke: alde batetik probatan oinarritzen diren teknikak; beste aldetik, frogapen formaletan (matematikoetan) oinarritzen diren teknikak. Probatan oinarritutako teknikek datuen kasuistika zabala erabiltzen dute programen portaera aztertzeko. Egiaztapenaren kalitatea erabilitako proba-bankuaren estalduraren araberakoa izango da. Programa bat zuzena dela bermatzeko, sarrerako parametroek har ditzaketen balio guztiak eduki beharko lituzke proba-bankuak, baina hori ezinezkoa da balio horiez osatutako multzoa infinitua baldin bada, gehienetan gertatzen den bezala. Horregatik, aztertzen ari garen programak erroreak badituela froga daiteke metodo hauek erabiliz; ez, ordea, programa hori errorerik gabea denik. Aldiz, frogapen formal batek programa zuzena dela ziurtatzen du (esperotako portaeraren deskribapen formal batekiko). Onura horren truke, programa baten zuzentasun-frogapena formalizatzeko erabili behar diren garapen matematikoak nahiko neketsuak izan ohi dira oro har, eta, gainera, erroreak ere sor daitezke. Zailtasun horiek arintzeko, egindako frogapen formalen zuzentasuna automatikoki aztertzen duten software-tresna ugari sortu dira (ikusi 3.15. atala).

Kapitulu honetan egiaztapen formalerako teknika batean sakonki barneratuko gara adibide ugari emanez: Hoare-ren sistema formala. Hain zuzen ere, sekuentziala den edozein programazio-lengoaia agintzaileren muina formalizatzen duen axioma eta erregelen multzoa landuko dugu. Hasteko, 3.1. atalean, programen zuzentasunari buruzko kontzeptu eta notazioak finkatuko dira. Ondoren, 3.2. atalean, sistema formalaren kontzeptua orokorrean aurkeztuko da lehenengo, eta gero Hoareren sistema formalaren erabilgarritasuna eta helburuak zehaztuko dira. Hurrengo ataletan programazio-lengoaia

agintzaile eta sekuentzialen oinarrizko aginduen zuzentasuna landuko da: esleipena (3.3. atala), konposaketa sekuentziala (3.5. atala), baldintzazko aginduak (3.7. atala) eta iterazioak (3.9. atala). Lau atal horietako bakoitzaren ondoren landutako aginduari buruzko ariketez osatutako atala dator (3.4., 3.6., 3.8. eta 3.10. atalak hain zuzen ere). Adibideetan Hoareren sistema formala erabiliz eskuz egindako zuzentasun-frogen garapen zehatzak aurkeztuko dira (ariketetan egiteko eskatuko da). Horrelako frogak eskuz egitea oso neketsua eta astuna izan daiteke, baita kapitulu honetan lantzen diren bezalako programa laburretarako ere. Hala eta guztiz, programazio agintzailearen oinarriak sendotzeko balio du, baita programa bati buruz era matematikoan arrazoitzeko gaitasuna garatzeko ere. Gero, 3.11. eta 3.12. ataletan, asertzioen bidez programak dokumentatzeko teknika aurkeztuko da eta ariketa batzuk proposatuko dira. Iterazioen bukaera frogatzeko metodoa 3.13. atalean landuko da eta metodo horri buruzko ariketak 3.14. atalean planteatuko dira. Egiaztapen formala sakonago aztertzeko, arlo honetako historiari eta azken aurrerapenei buruzko erreferentziak eta irakurgai osagarriak ematen dira 3.15. atalean. Bukatzeko, ariketa orokor batzuk proposatzen dira 3.16. atalean.

3.1. PROGRAMEN ZUZENTASUNA

Programa baten zuzentasuna programaren ustezko portaera ezartzen duen deskribapen (formal) batekiko erabaki ohi da. Hortaz, programa bat espezifikazio batekiko zuzena izan daiteke, baina beste espezifikazio batekiko ez. Bi alderdi bereizi behar dira: zein den programa exekutatuz lortu nahi den emaitza eta zein den benetan lortzen den emaitza. Programa zuzena izateko emaitza horiek bat etorri beharko dute. Liburu honetan, aurreko kapituluan ikusitako *aurre-ondoetako espezifikazio*en bidez adieraziko da programa exekutatuz lortu nahi den emaitza.

Gainera, bi zuzentasun mota bereiziko ditugu: *zuzentasun partziala* eta *zuzentasun osoa*.

Izan bitez P programa (edo agindu-segida) eta (φ, ψ) espezifikazioa, φ aurrebaldintza eta ψ postbaldintza izanda. P programa (φ, ψ) espezifikazioarekiko *partzialki zuzena* izango da, baldin eta soilik baldin φ betetzen duen egoera batean hasiz gero ψ betetzen duen egoera batean bukatzen bada, betiere P-ren konputazioa bukatzen bada. P programa (φ, ψ) espezifikazioarekiko *partzialki zuzena* dela adierazteko honako hau idatziko dugu:

$$\{\, \varphi \,\}\ \mathtt{P}\ \{\, \psi \,\}$$

Zuzentasun partziala erabakitzerakoan ez da aztertuko aurrebaldintza betetzen duen edozein egoeratatik hasita programa bukatzen den ala ez. Bukaera

duten konputazioak postbaldintza betetzen duen egoera batean bukatuko direla baino ez du adierazten zuzentasun partzialak. Zuzentasun partzialak aurrebaldintza betetzen duen egoera batetik hasi eta bukatzen ez diren konputazioei buruz ez du ezer esaten. Gerta daiteke P-ren konputazio batzuk bukatzea eta beste batzuk ez, betiere hasierako egoeraren arabera. Beraz, φ aurrebaldintza funtsezkoa da P bukatzen den ala ez erabakitzeko. φ aurrebaldintza betetzen duen egoera batetik hasten den P-ren edozein konputazio urrats kopuru finituan *bukatzen* bada, orduan P programa φ *aurrebaldintzarekiko bukatu egiten dela* esango dugu.

Zuzentasun partzialak gehi programaren bukaerak programa baten *zuzentasun osoa* bermatzen dute. Hau da, P programa (φ,ψ) espezifikazioarekiko *guztiz zuzena* izango da (φ,ψ)-rekiko partzialki zuzena baldin bada eta φ-rekiko bukatzen bada. P programaren zuzentasun osoa edo, beste era batean esanda, P programa guztiz zuzena dela honela adieraziko da:

$$\{\ \varphi\ \}\ [\ \mathtt{P}\ ]\ \{\ \psi\ \}$$

Beraz, P programa guztiz zuzena baldin bada (φ,ψ) espezifikazioarekiko, φ betetzen duen egoera batean hasten den P-ren edozein konputazio ψ betetzen den egoera batean bukatuko da. Beraz, P programa bat (φ,ψ) espezifikazioarekiko guztiz zuzena izango da, espezifikazio horrekiko partzialki zuzena baldin bada, eta, gainera, φ aurrebaldintzarekiko bukatzen bada. Beste hitz batzuekin esanda, zuzentasun osoari buruzko $\{\ \varphi\ \}\ [\ \mathtt{P}\ ]\ \{\ \psi\ \}$ baieztapena frogatzeko, alde batetik zuzentasun partzialari buruzko $\{\ \varphi\ \}\ \mathtt{P}\ \{\ \psi\ \}$ baieztapena frogatu beharko dugu, eta, bestetik, P programa φ-rekiko bukatu egiten dela frogatu beharko dugu.

3.2. HOARE-REN SISTEMA FORMALA

Atal honetan, hasteko, sistema formalen kontzeptua azalduko dugu eta, ondoren, Hoareren sistema formala aurkeztuko dugu. Hoareren sistema formalaren helburua programei buruzko zuzentasun partzialeko baieztapenak frogatzea da. Hurrengo ataletan, Hoareren sistema formala sakonki deskribatuko da.

Sistema formal batek honako bi eratako osagaiak izango ditu:

- Axiomak: egiazkotzat jotzen diren oinarrizko propietateak.
- Inferentzi erregelak: aurretik ditugun propietateetatik propietate berriak ondorioztatzea ahalbidetzen dute. Erregelak era jakin batean adierazten dira. Esate baterako, P propietatea P_1, P_2, …, P_n propietateetatik ondorioztatzen dela honela adierazten da:

$$\frac{P_1,\ P_2,\ \ldots,\ P_n}{P}$$

Marraren gainean dauden P_1, P_2, ..., P_n propietateak *premisak* dira, eta ondorioztatzen den P propietatea azpian agertzen da. Beraz, inferentzi erregelak honela ulertu behar dira: P_1, P_2, ..., P_n propietateak ondorioztatu (edo frogatu) baldin badaitezke, orduan P propietatea ere ondoriozta daiteke.

Sistema formal batean, *frogapenak* propietateen segidak dira. Izan bedi hurrengo propietateen segida:

$$P_1,\ P_2,\ \ldots,\ P_{i-1},\ P_i,\ P_{i+1},\ \ldots,\ P_{k-1},\ P_k$$

Segida horretan P_k frogatu nahi dugun propietatea da eta P_i $(1 \leq i \leq k)$ bakoitza honako bi eratako elementu bat da:

- axioma bat, edo
- inferentzi erregela baten bitartez aurreko P_1, ...,P_{i-1} propietateetatik ondorioztatzen den propietatea.

Hoareren sistema formalak zuzentasun partzialari buruzko baieztapenak frogatzeko balio du. Hau da, $\{\ \varphi\ \}$ `P` $\{\ \psi\ \}$ erako adierazpenak frogatzeko balio du. Baieztapen horiei *Hoareren hirukote* ere deitzen zaie. Era horretako hirukote bakoitzak egiazkoa edo faltsua izan daitekeen zuzentasun partzialeko baieztapen edo propietate bat adierazten du (aurreko atalean ikusi den bezala). Adibidez, argi dago $\{\ x = 2\ \}$ `x := x+1;` $\{\ x = 3\ \}$ baieztapena edo propietatea egiazkoa dela, eta $\{\ x = 2\ \}$ `x := x+1;` $\{\ x = 1\ \}$ faltsua dela. Hau da, $\{\ x = 2\ \}$ baldintza betetzen duen egoera batean hasten diren `x := x+1;` programaren konputazio guztiak $\{\ x = 3\ \}$ baldintza betetzen duen egoera batean bukatuko dira, baina ez dira $\{\ x = 1\ \}$ baldintza betetzen duen egoera batean bukatuko. Hoareren sistema formala axioma eta erregelez osatutako multzo bat da eta sistema hori erabiliz egiazkoak diren hirukote edo zuzentasun partzialeko baieztapenen frogak eraiki daitezke. Baieztapen faltsuak, ordea, ez dira frogagarriak. Adibidez, Hoareren sistema formala erabiliz honako baieztapen hau froga daiteke:

$\{\ x = a \wedge y = b\ \}$
```
  lag := x;
  x := y;
  y := lag;
```
$\{\ x = b \wedge y = a\ \}$

Baina beste baieztapen hau ezin da frogatu:

$\{\, True \,\}$
```
  lag := x;
  x := y;
  y := lag;
```
$\{\, x = y \,\}$

Azken adibide horretan oinarrituko gara argibide garrantzitsu batzuk ematekо. Hirukote bat faltsua dela frogatzeko kontraadibide bat ematearekin nahikoa da. Kontraadibidea aurrebaldintza betetzen den egoera batean hasi eta postbaldintza betetzen ez den egoera batean bukatzen den konputazio bat izango da. Azken hirukotearen kontraadibide bat $\{(x,2),(y,7)\}$ egoeran hasten den konputazioa da. Egoera horretan $True$ aurrebaldintza bete egiten da (edozein egoerak $True$ formula betetzen baitu). Konputazioa hurrengo egoeretatik igaroko da:

$$\{\, (x,2),\, (y,7),\, (lag,2) \,\}$$
$$\{\, (x,7),\, (y,7),\, (lag,2) \,\}$$
$$\{\, (x,7),\, (y,2),\, (lag,2) \,\}$$

Argi dago azken egoerak $x = y$ postbaldintza ez duela betetzen. Bestalde, ez konputazio horrek ezta konputazio multzo finitu batek ere ezin dute frogatu honako hirukote hau egiazkoa denik:

$\{\, x = a \wedge y = b \,\}$
```
  lag := x;
  x := y;
  y := lag;
```
$\{\, x = b \wedge y = a \,\}$

Izan ere, onargarriak diren hasierako egoerez osatutako multzoa infinitua da. Azken hirukote horretan x eta y aldagaien hasierako balio onargarri guztiak adierazteko a eta b erabiltzen dira. Konputazio-egoeren segidekin arrazoitu behar denean, asertzioen bidez aldagai baten lehengo balioari buruzko propietateak formalizatu behar izatea espezifikazio-arazo ezaguna da. Adibide honetan, azken konputazio-egoeran x aldagaiak y-ren hasierako balioa duela, eta y aldagaiak x-ren hasierako balioa duela adierazi behar du aurre-ondoetako espezifikazioak. Kontuan izan $x = y$ postbaldintza erabiltzen badugu, azken egoeran x eta y aldagaiek balio bera dutela adierazten ari garela. Hainbat teknika proposatu izan dira espezifikazio-arazo hori ekiditeko. Aurreko adibidean a eta b-rekin egin dugun bezalaxe, liburu honetan programetan agertzen ez diren izen berriak erabiltzeko teknikari jarraituko diogu. Beraz, postbaldintzan erabiliko diren hasierako balioei izen bat

esleituko zaie aurrebaldintzan, eta, horrela, postbaldintzan hasierako eta bukaerako balioak bereizi ahal izango ditugu. Aurreko hirukoteak adierazten duen zuzentasun partzialeko baieztapena honako hau da:

> «Hasierako egoeran x-k a balioa (edozein) baldin badu eta y-k b balioa (edozein) baldin badu, orduan bukaerako egoeran x-k b balioa izango du, eta y-k a balioa izango du».

Aurreko hiru aginduez osatutako programaren portaera era horretan espezifikatzea nahiko zentzuzkoa da.

Espezifikazio-arazo horren deskribapena eta soluzioa ilustratzeko, sinpleagoa den adibide bat erabiliko dugu. Honako baieztapen hau

$\{ True \}$
```
  x := x+1;
```
$\{ x = x + 1 \}$

faltsua da, $x = x + 1$ ekuazioa ezin baita inola ere bete. Zer gertatzen da kasu honetan? Arazoa esleipenaren bi aldeetan x aldagai bera erabiltzetik dator: ezkerreko aldean helburu-aldagai bezala eta eskuineko aldean azpitermino bezala. Hala ere, nahiz eta aldagai bera izan, asertzioan x aldagaiaren bi balio desberdin aipatu nahi dira: lehenengoa, edo ezkerrekoa, esleipenaren bidez x-k hartzen duen balioa da (x-ren balio berria), eta bigarrena, eskuinekoa, esleipena exekutatu aurretik x-k zuen balioa da (x-ren lehengo balioa). Beraz, asertzioan x-ren bi balio horiek bereizi behar dira. Aurreko adibidean egin dugun bezala, aurrebaldintzan x-ren hasierako balioari a izena egokituko diogu, gero postbaldintzan x-ren balio berria eta lehengo balioa (edo hasierako balioa) erabili ahal izateko:

$\{ x = a \}$
```
  x := x+1;
```
$\{ x = a + 1 \}$

Programa batean aldagairen bat sarrerako datu gisa eta emaitza gisa erabiltzen denean, teknika hori erabiliko dugu beti programari buruzko propietateak adierazteko.

3.3. ESLEIPENA

Edozein programazio-lengoaia agintzailetako agindu mota oinarrizkoena esleipena da. Paradigma agintzailean, aldagaien balio-aldaketa esleipenen bidez gauzatzen da. Oro har, esleipen-agindu batek termino baten balioa aldagai bati esleitzen dio, aurretik aldagaiak zuen balioa ezabatuz. Esleipenak honako era honetan adieraziko ditugu:

```
x := t;
```

Hor x aldagaia izango da eta t terminoa izango da. 2.2.1. azpiatalean ikusi dugun bezala, terminoak sinpleak edo konposatuak izan daitezke. Beraz, t terminoa aldagai bat, konstante bat, edo eragileak eta (terminoak diren) eragigaiak erabiliz osatutako espresio bat izan daiteke.

Hoareren sistema formalean esleipenarekin lotutako propietateak edo baieztapenak frogatzeko *Esleipenaren Axioma* erabili behar dugu. Formalki, esleipenaren axiomaren definizioa honako hau da:

$$\textbf{(EA)} \qquad \boxed{\{\, def(t) \wedge \varphi_x^t \,\}\ \texttt{x := t;}\ \{\, \varphi \,\}}$$

Izan bedi `x := t;` agindua, t terminoa *definituta* baldin badago, φ_x^t betetzen duen egoera batean hasten den edozein konputazio φ betetzen duen egoera batean bukatuko dela adierazten du esleipenaren axiomak. Beste hitz batzuekin esanda: aurrebaldintzak t definituta dagoela ziurtatzen badu ($def(t)$), t terminoak aurrebaldintzan betetzen dituen propietateak (φ_x^t) beteko ditu x aldagaiak postbaldintzan (φ).

Oro har, terminoak eta formulak ondo definituta egotea ezinbesteko baldintza da programen konputazio-egoerei buruzko propietateak adierazi ahal izateko. Horrek esan nahi du beraien ebaluazioak errorerik ez duela sortu behar, 2. kapituluan azaltzen den bezala. Hemendik aurrera, $def(t)$ adierazpenak zera adieraziko du: t terminoa ondo definituta egoteko, hau da, t terminoaren ebaluazioak errorerik ez sortzeko bete beharreko baldintza betetzen duten egoera guztiak errepresentatzen dituen formula. Kasu berezia da $True$ formulak adierazten duena, $def(t)$ espresioak t edozein konputazio-egoeratan ondo definituta dagoela adieraziz. Liburu honetan kontuan hartuko ditugun bi errore mota nagusiak zeroz zatitzea eta bektoreetako eremuz kanpoko indizeak dira[1]. Zehazki, zeroz zatitzeak ($s/0$) eta zeroz zatitzearen hondarrak ($s \,\%\, 0$) errorea sortuko dute eta, $A(1..n)$ bektorea emanda, $A(i)$ erako adierazpenek errorea sortuko dute i balioa A bektorearen eremutik kanpo baldin badago, hau da, $i < 1$ edo $i > n$ baldin bada. Jakina, termino sinple guztiak (aldagaiak eta konstanteak) beti ondo definituta egongo dira. Gainera, erroreak barreiatu egiten dira. Hau da, t termino konposatuaren azpitermino batek errorea sortzen badu, orduan t terminoak ere errorea sortuko du. Adibidez, 10 terminoa ondo definituta dago edozein konputazio-egoeratan, $10/x$ eta $10 \,\%\, x$ terminoak ondo definituta egongo dira x zero ez bada, eta $(2*x)+(10 \,\%\, (y+z))$ terminoa ondo definituta egongo da $y+z$ zero ez bada:

1. Aldiz, zenbaki osoak inplementatzen dituen datu-motaren eremuz kanpoko balioek eragindako erroreak ez ditugu kontuan hartuko, beraz batuketa, kenketa eta biderketa eragiketek ez dute inoiz errorerik eragingo.

$$\begin{aligned} def(10) &\equiv True \\ def(10/x) &\equiv x \neq 0 \\ def(10 \,\%\, x) &\equiv x \neq 0 \\ def((2*x)+(10 \,\%\, (y+z))) &\equiv y+z \neq 0 \end{aligned}$$

Era berean, $A(1..n)$ bektorea emanda:

$$\begin{aligned} def(A(1)) &\equiv True \\ def(A(i)) &\equiv 1 \leq i \leq n \\ def(A(i+j)) &\equiv 1 \leq i+j \leq n \\ def(A(i/j)) &\equiv j \neq 0 \wedge 1 \leq i/j \leq n \\ def(x \,\%\, y + A(i/j)) &\equiv y \neq 0 \wedge j \neq 0 \wedge 1 \leq i/j \leq n \end{aligned}$$

Esleipenaren axiomaren formulazioa eskema orokor bat da. Hau da, edozein x aldagai, t termino eta φ formula hartuz, eskema horren instantziak edo formulazio konkretuak egin daitezke. Horrela osatutako adierazpen konkretuak dira, izatez, esleipenaren axiomak, hau da, Hoareren sistema formalean egiazkoak diren oinarrizko baieztapenak. Adibidez,

$\{\, x+1=3 \,\}$
```
  x := x+1;
```
$\{\, x=3 \,\}$

hirukotea esleipenaren axioma bat da; izan ere, $x+1$ ondo definituta dago ($def(x+1) \equiv True$) eta $(x=3)^{x+1}_{x}$ ordezkapenak $x+1=3$ adierazten du[2]. Jakina, $x+1=3$ eta $x=2$ formulek konputazio-egoeren multzo bera adierazten dute, aurrebaldintza bera adierazten dute-eta. Beraz,

$\{\, x=2 \,\}$
```
  x := x+1;
```
$\{\, x=3 \,\}$

hirukotea ere esleipenaren axioma bat da, eta, ondorioz, baieztapena egiazkoa dela ziurtatzen duen urrats bakarreko frogapen formala.

Hurrengo baieztapena ere esleipenaren axiomaren bidez froga daiteke:

$\{\, 1 \leq i \leq n \wedge A(i) > 0 \,\}$
```
  x := A(i);
```
$\{\, x > 0 \,\}$

Baieztapen horretan $\varphi \equiv (\, x > 0 \,)$, $\varphi^{A(i)}_{x} \equiv (\, A(i) > 0 \,)$ eta $def(A(i)) \equiv (\, 1 \leq i \leq n \,)$ izango lirateke.

2. Kontuan hartu $x+1=3$ formula $True \wedge x+1=3$ formularen sinplifikazioa dela.

Esleipenaren axiomak esleipen-aginduei buruz eta agindu horien eraginei buruz arrazoitzeko aukera ematen digu. Hala ere, esleipenei buruz egindako baieztapen zuzen guztiak ezin dira esleipenaren axioma bakarrik erabiliz frogatu. Esate baterako, honako baieztapena

$$\{\, z \neq 0 \wedge y/z \neq 1 \,\}$$
```
  x := y/z;
```
$$\{\, x \neq 1 \,\}$$

esleipenaren axioma bakarrik erabiliz frogagarria da. Baina beste baieztapen hau hartzen badugu:

$$\{\, z \neq 0 \wedge y = 0 \,\}$$
```
  x := y/z;
```
$$\{\, x \neq 1 \,\}$$

baieztapen hori frogatzeko esleipenaren axioma erabiltzea ez da nahikoa, nahiz eta baieztapena zuzena izan. Horregatik, Hoareren sistema formalak aurreko adibideko baieztapena bezalakoak frogatzea ahalbidetzen duten eta, beraz, esleipenaren axiomaren gabeziak osatzen dituzten inferentzi erregelak ditu. Aurreko adibide horretan, $z \neq 0 \wedge y = 0$ baieztapenaren aurrebaldintza eta esleipenaren axioma erabiliz lortzen den $z \neq 0 \wedge y/z \neq 1$ formula baliokideak ez izateak eragiten du arazoa. Hor aurrebaldintza esleipenaren axioma erabiliz lortzen den formula baino gogorragoa da. Jarraian, (EA) esleipenaren axiomaz gain oraintxe aurkeztuko dugun (ODE) erregela ere erabiltzen duen frogapen bat emango da aurreko hirukote horrentzat. Frogapen hori eman baino lehen, aipa dezagun beste hirukote batzuetan postbaldintza behar baino ahulagoa izatea ere gerta daitekeela. Aurrebaldintza gogorregia edo/eta postbaldintza ahulegia duten propietateak frogatzeko, Hoareren sistema formaleko *Ondorioaren Erregela* erabil daiteke. Erregela hori honela definituko genuke formalki:

(ODE)
$$\frac{\varphi \rightarrow \varphi_1,\ \{\, \varphi_1 \,\}\ \mathtt{P}\ \{\, \psi_1 \,\},\ \psi_1 \rightarrow \psi}{\{\, \varphi \,\}\ \mathtt{P}\ \{\, \psi \,\}}$$

Erregela honek baditu oso erabiliak diren bi aldaera edo kasu partikular. Aldaera horietan ψ_1 eta ψ formulak edo φ_1 eta φ formulak berdinak dira:

$$\frac{\varphi \rightarrow \varphi_1,\ \{\, \varphi_1 \,\}\ \mathtt{P}\ \{\, \psi \,\}}{\{\, \varphi \,\}\ \mathtt{P}\ \{\, \psi \,\}} \qquad \frac{\{\, \varphi \,\}\ \mathtt{P}\ \{\, \psi_1 \,\},\ \psi_1 \rightarrow \psi}{\{\, \varphi \,\}\ \mathtt{P}\ \{\, \psi \,\}}$$

Ondorioaren erregelari esker, hirukote bat frogatzerakoan baieztapen horretako φ aurrebaldintza baino ahulagoa den φ_1 beste aurrebaldintza bat erabil

daiteke eta ψ postbaldintza baino gogorragoa den ψ_1 beste postbaldintza bat ere erabil daiteke.

Esleipenaren axioma eta ondorioaren erregela erabiliz, urrats bat baino gehiago dituzten frogapenak lortuko dira. Oro har, frogapen bat axiomen eta inferentzi erregelen aplikazioetatik lortzen den propietate-segida bat da, propietate horiek Hoareren hirukoteak edo $\alpha \to \beta$ erako inplikazio logikoak izan daitezkeelarik. Segidako azken propietatea frogatu nahi dugun Hoareren hirukotea izango da.

Ondoren, $\{\ z \neq 0 \ \wedge\ y = 0\ \}$ `x := y/z;` $\{\ x \neq 1\ \}$ hirukotearen frogapen formala emango dugu:

1. $(\ z \neq 0 \ \wedge\ y = 0\ ) \ \to\ (\ z \neq 0 \ \wedge\ y/z \neq 1\ )$
2. $\{\ z \neq 0 \ \wedge\ y/z \neq 1\ \}$
   `x := y/z;` **(EA)**
   $\{\ x \neq 1\ \}$
3. $\{\ z \neq 0 \ \wedge\ y = 0\ \}$
   `x := y/z;` 1, 2 eta **(ODE)**
   $\{\ x \neq 1\ \}$

Inferentzi erregela bateko premisen agerpen-ordenak ez du garrantzirik. Hori dela-eta, aurreko adibideko 1. eta 2. propietateak alderantzizko ordenan ipin daitezke.

Atal honetako gainerakoan, esleipenaren axioma eta ondorioaren erregelaren erabilera erakusteko helburuarekin, egiazkoak diren hirukote batzuen frogapenak aurkeztuko ditugu.

Izan bitez zenbaki osozko $A(1..n)$ taula eta *positiboa* izeneko predikatua:

$$\mathit{positiboa}(A(1..n)) \ \equiv\ \forall i\ (\ 1 \leq i \leq n \ \to\ A(i) > 0\ )$$

Honako baieztapen hau

$\{\ 1 \leq j \leq n \ \wedge\ \mathit{positiboa}(A(1..n))\ \}$
`x := A(j);`
$\{\ x > 0\ \}$

zuzena dela honela froga daiteke:

1. $\{\ 1 \leq j \leq n \ \wedge\ A(j) > 0\ \}$
   `x := A(j);` **(EA)**
   $\{\ x > 0\ \}$
2. $(\ 1 \leq j \leq n \ \wedge\ \mathit{positiboa}(A(1..n))\ ) \ \to\ (\ 1 \leq j \leq n \ \wedge\ A(j) > 0\ )$

3. $\{\ 1 \leq j \leq n \wedge \mathit{positiboa}(A(1..n))\ \}$
   `x := A(j);` 1, 2 eta **(ODE)**
   $\{\ x > 0\ \}$

Frogapen horretan esleipenaren axiomaren aplikaziotik lortutako aurrebaldintza hasierako aurrebaldintza baino ahulagoa da. Horregatik, ondorioaren erregela erabili beharra dago.

Beste baieztapen hau ere aurrekoen antzera froga daiteke:

$\{\ 1 \leq i < n\ \}$
`i := i+1;`
$\{\ 1 \leq i \leq n\ \}$

Hona hemen haren zuzentasunaren froga:

1. $\{\ 1 \leq i+1 \leq n\ \}$
   `i := i+1;` **(EA)**
   $\{\ 1 \leq i \leq n\ \}$
2. $(\ 1 \leq i < n\ ) \rightarrow (\ 1 \leq i+1 \leq n\ )$
3. $\{\ 1 \leq i < n\ \}$
   `i := i+1;` 1, 2 eta **(ODE)**
   $\{\ 1 \leq i \leq n\ \}$

Kasu honetan, bigarren urratseko inplikazio logikoa betetzen denez, lehenengo urratseko hirukoteko $(\ 1 \leq i+1 \leq n\ )$ aurrebaldintza frogatu nahi den hirukoteko $(\ 1 \leq i < n\ )$ aurrebaldintzaz ordezka daiteke, eta hori da, hain zuzen ere, ondorioaren erregela erabiliz hirugarren urratsean egin dena. Kontuan hartu $(\ 1 \leq i+1 \leq n\ )$ eta $(\ 1 \leq i < n\ )$ formulak ez direla baliokideak; izan ere, $(\ 1 \leq i+1 \leq n\ )$ formula $(\ 0 \leq i < n\ )$ formularen baliokidea da.

Honako baieztapen honen frogapena ere aurrekoen antzekoa da:

$\{\ z = 1 \wedge x > 0\ \}$
`y := 0;`
$\{\ z = x^y \wedge y \geq 0\ \}$

Hona hemen frogapena:

1. $\{\ z = x^0 \wedge 0 \geq 0\ \} \equiv \{\ z = x^0\ \}$
   `y := 0;` **(EA)**
   $\{\ z = x^y \wedge y \geq 0\ \}$
2. $(\ z = 1 \wedge x > 0\ ) \rightarrow (\ z = x^0\ )$
3. $\{\ z = 1 \wedge x > 0\ \}$
   `y := 0;` 1, 2 eta **(ODE)**
   $\{\ z = x^y \wedge y \geq 0\ \}$

Ondoren aztertuko dugun frogapenean, aurrekoetan ez bezala, ondorioaren erregelaren kasu orokorra erabiliko da (eta ez haren aldaerak). Honako baieztapen hau frogatuko dugu:

$$\{\, b \wedge \mathit{positiboa}(A(1..i)) \wedge 1 \leq i < n \wedge A(i+1) < 0 \,\}$$
`  b := false;`
$$\{\, b \leftrightarrow \mathit{positiboa}(A(1..i+1)) \,\}$$

Baieztapen horretan b aldagai boolearra da eta *positiboa* lehenago definitu dugun predikatua:

1. $(b \wedge \mathit{positiboa}(A(1..i)) \wedge 1 \leq i < n \wedge A(i+1) < 0)$
   $\rightarrow (1 \leq i < n \wedge A(i+1) < 0) \rightarrow (\neg \mathit{positiboa}(A(1..i+1)))$
   $\rightarrow (\neg \mathit{False} \wedge \neg \mathit{positiboa}(A(1..i+1)))$
2. $\{\, \neg \mathit{False} \wedge \neg \mathit{positiboa}(A(1..i+1)) \,\}$
   `b := false;` **(EA)**
   $\{\, \neg b \wedge \neg \mathit{positiboa}(A(1..i+1)) \,\}$
3. $(\neg b \wedge \neg \mathit{positiboa}(A(1..i+1))) \rightarrow (\neg b \leftrightarrow \neg \mathit{positiboa}(A(1..i+1)))$
   $\rightarrow (b \leftrightarrow \mathit{positiboa}(A(1..i+1)))$
4. $\{\, b \wedge \mathit{positiboa}(A(1..i)) \wedge 1 \leq i < n \wedge A(i+1) < 0 \,\}$
   `b := false;` 1, 2, 3 eta **(ODE)**
   $\{\, b \leftrightarrow \mathit{positiboa}(A(1..i+1)) \,\}$

Frogapen horretako lehenengo urratsa hiru inplikazio kateatzen dituen eta $\varphi \rightarrow \alpha \rightarrow \beta \rightarrow \varphi_1$ egitura duen espresioa da, baina ondorioaren erregelan $\varphi \rightarrow \varphi_1$ formula erabiltzen da. Hor α eta β tarteko formulak $\varphi \rightarrow \varphi_1$ inplikazioa argitzeko ipini dira. Gauza bera egin da hirugarren urratseko inplikazio logikoarekin ere, baina tarteko formula bakar bat erabiliz.

Aurreko hirukote bera ondorioaren erregelaren lehenengo aldaera erabiliz ere froga daiteke:

1. $(b \wedge \mathit{positiboa}(A(1..i)) \wedge 1 \leq i < n \wedge A(i+1) < 0)$
   $\rightarrow (1 \leq i < n \wedge A(i+1) < 0)$
   $\rightarrow (\neg \mathit{positiboa}(A(1..i+1)))$
   $\rightarrow (\mathit{False} \leftrightarrow \mathit{positiboa}(A(1..i+1)))$
2. $\{\, \mathit{False} \leftrightarrow \mathit{positiboa}(A(1..i+1)) \,\}$
   `b := false;` **(EA)**
   $\{\, b \leftrightarrow \mathit{positiboa}(A(1..i+1)) \,\}$
3. $\{\, b \wedge \mathit{positiboa}(A(1..i)) \wedge 1 \leq i < n \wedge A(i+1) < 0 \,\}$
   `b := false;` 1, 2 eta **(ODE)**
   $\{\, b \leftrightarrow \mathit{positiboa}(A(1..i+1)) \,\}$

3.4. ARIKETAK: ESLEIPENA

1. Aukeratu hirukote zuzena osatuko luketen postbaldintzak, $A(1..n)$ zenbaki osoz eratutako bektorea izanda:

$\{\ 1 \leq n \land \forall i\ (\ 1 \leq i < n\ \rightarrow\ A(i) = A(n)\ )\ \}$

```
y := 1;
```

i) $\{\ \forall i\ (\ 1 \leq i \leq n\ \rightarrow\ A(i) = y\ )\ \}$ [ ]

ii) $\{\ \forall i\ (\ 2 \leq i \leq n\ \rightarrow\ A(i) = y * A(i-1)\ )\ \}$ [ ]

iii) $\{\ \forall i\ (\ 1 \leq i \leq n\ \rightarrow\ y * A(i) = y\ )\ \}$ [ ]

2. Osatu honako hirukote hauek postbaldintza egokiekin ({ __ }):

2.1. $\{\ \exists i\ (\ x \leq i \leq x+5\ \land\ A(i) > 0\ )\ \}$

```
x := x+2;
```

{ ________________________________ }

2.2. $\{\ bikoitia(k)\ \land\ y * z^k = p\ \}$

```
k := k/2;
z := z*z;
```

{ ________________________________ }

3. Bete hutsuneak (____) honako hirukote honi dagokion frogapenean:

$\{\ bikoitia(k)\ \land\ y * z^k = p\ \}$

```
k := k/2;
```

$\{\ y * z^{2*k} = p\ \}$

Frogapena:

1. $(\ bikoitia(k)\ \land\ y * z^k = p\ )\ \rightarrow\ (\ $ ________________)

2. { ________________ }
   `k := k/2;` **(EA)**
   $\{\ y * z^{k*2} = p\ \}$

3. $\{\ bikoitia(k)\ \land\ y * z^k = p\ \}$
   `k := k/2;` 1, 2 eta **(ODE)**
   $\{\ y * z^{k*2} = p\ \}$

4. Egiaztatu era formalean honako baieztapen hauek:

4.1. $\{\ n \geq 1\ \land\ i = 0\ \}$

```
zerorik_ez := true;
```

$\{\ 0 \leq i \leq n\ \land\ (\ zerorik\_ez \leftrightarrow \forall k\ (\ 1 \leq k \leq i\ \rightarrow\ A(k) \neq 0\ )\ )\ \}$

4.2. $\{\, i = j^k \wedge i < w \,\}$
`i := i*j;`
$\{\, i = j^{k+1} \,\}$

3.5. KONPOSAKETA SEKUENTZIALA

Orain arte, frogatu ahal izan ditugun baieztapenak esleipen-agindu bakarrekoak ziren. Atal honetan, agindu batez baino gehiagoz osatutako agindusegidei buruzko propietateak frogatzea ahalbidetzen duen inferentzi erregela aurkeztuko dugu: *konposaketaren erregela*.

Formulaziorik orokorrenean, n ($n >= 2$ izanda) programaz osatutako P_1, P_2, ..., P_n segida bat hartuta, konposaketaren erregelaren definizioa honako hau da:

(KPE)
$$\frac{\{\, \varphi \,\}\ \mathtt{P_1}\ \{\, \varphi_1 \,\},\ \{\, \varphi_1 \,\}\ \mathtt{P_2}\ \{\, \varphi_2 \,\},\ \ldots\ ,\ \{\, \varphi_{n-1} \,\}\ \mathtt{P_n}\ \{\, \psi \,\}}{\{\, \varphi \,\}\ \mathtt{P_1}\ \mathtt{P_2}\ \ldots\ \mathtt{P_n}\ \{\, \psi \,\}}$$

Erregela horrek badu P_1 eta P_2 programez edo aginduez osatutako segidarako formulazio sinpleagoa ere:

$$\frac{\{\, \varphi \,\}\ \mathtt{P_1}\ \{\, \varphi_1 \,\},\ \{\, \varphi_1 \,\}\ \mathtt{P_2}\ \{\, \psi \,\}}{\{\, \varphi \,\}\ \mathtt{P_1}\ \mathtt{P_2}\ \{\, \psi \,\}}$$

Bi programa edo bi aginduko segidarako formulazioan, konposaketaren erregelak adierazten du $\{\, \varphi \,\}\ \mathtt{P_1}\ \mathtt{P_2}\ \{\, \psi \,\}$ erako baieztapenak frogatu ahal izateko tarteko φ_1 asertzio bat erabili behar dela. Tarteko asertzio hori erabiliz, hasierako propietatetik propietate horretako bi programa edo agindu horietako bakoitzarentzat propietate berri bat osatuko da. Lehenengo propietatea $\{\, \varphi \,\}\ \mathtt{P_1}\ \{\, \varphi_1 \,\}$ izango da eta propietate horretan postbaldintza φ_1 izango da eta aurrebaldintza hasierako propietatearen aurrebaldintza bera. Bigarren baieztapena edo propietatea $\{\, \varphi_1 \,\}\ \mathtt{P_2}\ \{\, \psi \,\}$ izango da, eta bigarren propietate horretan aurrebaldintza φ_1 izango da eta postbaldintza hasierako propietatearen postbaldintza izango da. Beraz, bi programen edo aginduen arteko zubi-lana egiten du φ_1 tarteko asertzioak. Kontuan hartu P_1 edota P_2 programak konposatuak izan daitezkeela. Beraz, konposaketaren erregela erabiliz P_1 (edota P_2) programari buruzko baieztapena frogatzeko bi baieztapen berri osa daitezke, eta horrela jarrai daiteke deskonposaketa-prozesuarekin agindu bakarreko programak lortu arte eta, ondorioz, konposaketaren erregela orokorrarekin urrats bakar batean egin daitekeenaren baliokidea dena eginez. Konposaketaren erregelaren formulazio orokorrean, n agindu edo programa

dituen hasierako propietatea frogatzeko n propietate frogatzen dira $n-1$ tarteko asertzio erabiliz.

Honako propietate hau frogatzeko konposaketaren erregela erabiliko dugu:

$\{\, x = a \wedge y = b \,\}$
```
  x := x+y;
  y := x-y;
  x := x-y;
```
$\{\, x = b \wedge y = a \,\}$

Propietatean hiru agindu agertzen direnez, bi tarteko asertzio beharko ditugu. Tarteko asertzio horiek lortzeko, behetik gorako edo goitik beherako estrategiak erabil daitezke. Tarteko asertzioak aukeratutako estrategiaren araberakoak izango dira:

	Tarteko asertzioak	
	Goitik beherako estrategia	Behetik gorako estrategia
$\{\, x = a \wedge y = b \,\}$		
`x := x+y;`	$\varphi_1 \equiv x = a+b \wedge y = b$	$\varphi'_1 \equiv y = b \wedge x-y = a$
`y := x-y;`	$\varphi_2 \equiv x = a+b \wedge y = a$	$\varphi'_2 \equiv x-y = b \wedge y = a$
`x := x-y;`		
$\{\, x = b \wedge y = a \,\}$		

Goitik beherako estrategiari jarraituz eta estrategia horri dagozkion asertzioak erabiliz, propietatearen honako frogapen hau lor daiteke:

1. $(\, x = a \wedge y = b \,) \rightarrow (\, x+y = a+b \wedge y = b \,)$
2. $\{\, x+y = a+b \wedge y = b \,\}$
   `x := x+y;` **(EA)**
   $\{\, \underbrace{x = a+b \wedge y = b}_{\varphi_1} \,\}$
3. $(\, \underbrace{x = a+b \wedge y = b}_{\varphi_1} \,) \rightarrow (\, x = a+b \wedge x-y = a \,)$
4. $\{\, x = a \wedge y = b \,\}$
   `x := x+y;` 1, 2, 3 eta **(ODE)**
   $\{\, x = a+b \wedge x-y = a \,\}$
5. $\{\, x = a+b \wedge x-y = a \,\}$
   `y := x-y;` **(EA)**
   $\{\, \underbrace{x = a+b \wedge y = a}_{\varphi_2} \,\}$

6. $(\underbrace{x = a+b \wedge y = a}_{\varphi_2}) \rightarrow (x - y = b \wedge y = a)$
7. $\{ x - y = b \wedge y = a \}$
   `x := x-y;` **(EA)**
   $\{ x = b \wedge y = a \}$
8. $\{ \underbrace{x = a+b \wedge y = a}_{\varphi_2} \}$
   `y := x-y;` 6, 7 eta **(ODE)**
   $\{ x = b \wedge y = a \}$
9. $\{ x = a \wedge y = b \}$
   `x := x+y;`
   `y := x-y;`
   `x := x-y;` 4, 5, 8 eta **(KPE)**
   $\{ x = b \wedge y = a \}$

Baina behetik gorako estrategia erabiltzen bada, lau urratseko honako frogapen hau osa daiteke:

1. $\{ \underbrace{x - y = b \wedge y = a}_{\varphi'_2} \}$
   `x := x-y;` **(EA)**
   $\{ x = b \wedge y = a \}$
2. $\{ x - (x - y) = b \wedge x - y = a \} \equiv \{ \underbrace{y = b \wedge x - y = a}_{\varphi'_1} \}$
   `y := x-y;` **(EA)**
   $\{ \underbrace{x - y = b \wedge y = a}_{\varphi'_2} \}$
3. $\{ y = b \wedge x + y - y = a \} \equiv \{ y = b \wedge x = a \} \equiv \{ x = a \wedge y = b \}$
   `x := x+y;` **(EA)**
   $\{ \underbrace{y = b \wedge x - y = a}_{\varphi'_1} \}$
4. $\{ x = a \wedge y = b \}$
   `x := x+y;`
   `y := x-y;`
   `x := x-y;` 1, 2, 3 eta **(KPE)**
   $\{ x = b \wedge y = a \}$

Bigarren frogapen horretan, tarteko asertzioak diseinatzeko esleipenaren axioma erabili da, eta, ondorioz, frogapena laburragoa da.

3.6. ARIKETAK: KONPOSAKETA SEKUENTZIALA

1. Bete hutsuneak (____) honako hirukote honi dagokion frogapenean:

   $\{\ batura = g\ \}$
   ```
   g := g+1;
   batura := batura+g;
   ```
   $\{\ batura = 2*g-1\ \}$

   Frogapena:

   1. $(\ batura = g\ ) \rightarrow (\ \_\_\_\_\_\ )$
   2. $\{\ \underline{batura = g+1-1}\ \}$
      `g := g+1;` **(EA)**
      $\{\ batura = g-1\ \}$
   3. $\{\ batura = g\ \}$
      `g := g+1;` 1, 2 eta **(ODE)**
      $\{\ batura = g-1\ \}$
   4. $(\ batura = g-1\ ) \rightarrow (\ batura+g-g = g-1\ )$
      $\rightarrow (\ \_\_\_\_\_\ )$
   5. $\{\ \_\_\_\_\_\ \}$
      `batura := batura+g;` **(EA)**
      $\{\ batura = 2*g-1\ \}$
   6. $\{\ batura = g-1\ \}$
      `batura := batura+g;` 4, 5 eta **(ODE)**
      $\{\ batura = 2*g-1\ \}$
   7. $\{\ batura = g\ \}$
      ```
      g := g+1;
      batura := batura+g;
      ```
      3, 6 eta **(KPE)**
      $\{\ batura = 2*g-1\ \}$

2. Frogatu, kontraadibide bat emanez, honako baieztapen hau ez dela zuzena:

   $\{\ y*z^k = p\ \}$
   ```
   k := k/2;
   z := z*z;
   ```
   $\{\ y*z^k = p\ \}$

3. Honako bi baieztapen hauetatik zuzena denarentzat, eman dagokion zuzentasun-froga, eta, zuzena ez denarentzat, eman kontraadibide bat zuzena ez dela erakusteko:

(A) $\{\, 4*x = 5^{k+1} - 1 \,\}$
```
k := k+1;
x := x + 5**k;
```
$\{\, 4*x = 5^{k+1} - 1 \,\}$

(B) $\{\, x = 5^{k+1} \,\}$
```
k := k+1;
x := x + 5**k;
```
$\{\, x = 5^{k+1} \,\}$

(A) baieztapena zuzena da eta (B) ez da zuzena [ ]

(B) baieztapena zuzena da eta (A) ez da zuzena [ ]

3.7. BALDINTZAZKO AGINDUAK

Atal honetan, baldintzazko aginduei buruzko baieztapenen frogapena aztertuko da.

Baldintzazko *if-then-else* erako aginduen egitura honako hau izan ohi da:

```
if B then
  P1
else
  P2
end if;
```

Eta era horretako aginduen semantika operazionala honela defini daiteke: hasteko B baldintza ebaluatuko da, eta ebaluazio horren emaitzaren arabera P_1 (*then* blokea) edo P_2 (*else* blokea) agindu-segida exekutatuko da. Hau da, B baldintza egiazkoa baldin bada, orduan P_1 exekutatuko da, eta B baldintza faltsua baldin bada, P_2 exekutatuko da.

Beraz, honako hirukote honek

$\{\, \varphi \,\}$
```
if B then
  P1
else
  P2
end if;
```
$\{\, \psi \,\}$

φ betetzen den egoera batean hasten den edozein konputazio (bukatzen bada) ψ betetzen den egoera batean bukatuko dela adierazten du, bidean P_1 edo P_2 agindu-segida exekutatuz. Beraz, gerta daitezkeen bi aukerak hartu beharko

ditugu kontuan, hau da, B betetzea edo ez betetzea. B baldintza betetzen bada, orduan P_1 agindu-segida exekutatuko da, eta, beraz, frogatu behar den propietatea honako hau izango da:

$$\{\, \varphi \wedge B \,\}\ \mathtt{P_1}\ \{\, \psi \,\}$$

Bestela, B baldintza ez bada betetzen, orduan P_2 exekutatuko da, eta beste baieztapen hau frogatu beharko da:

$$\{\, \varphi \wedge \neg B \,\}\ \mathtt{P_2}\ \{\, \psi \,\}$$

Gainera, B baldintzaren ebaluazioa egokia dela bermatu behar da. Hau da, B-ren ebaluazioak errorerik ez duela sortzen. Horretarako, $def(B)$ ere betetzen dela frogatu beharko da. Esleipenaren axiomaren formulazioan def funtzioa terminoekin erabili dugu, baina oraingo honetan def funtzioa formulekin erabiltzen da. Hori horrela izanda, def funtzioaren definizio berria aurretik genuen definizioaren orokortzetzat har daiteke. Izan ere, B baldintza (edo formula) definituta egongo da, baldin eta soilik baldin B baldintzan agertzen diren termino guztiak definituta badaude.

Edonola ere, B baldintzaren eta $def(B)$ formularen ebaluazioak loturarik gabekoak direla azpimarratu behar da. Hortaz, bi kontzeptuak ez dira nahastu behar: gauza bat da baldintza ondo definituta egotea (kasu horretan $def(B)$ espresioa $True$ formula izango da), eta beste gauza bat baldintza egiazkoa edo faltsua izatea. Are gehiago, B baldintzak ez dauka egoera guztietan definituta egon beharrik; izan ere, φ betetzen den egoera batetik hasitako baldintzazko aginduaren edozein konputaziori buruzko propietate bat frogatu nahi da. Beraz, φ formula betetzen den edozein egoeratan B baldintza definituta dagoela frogatzearekin nahikoa da, eta, horretarako, $\varphi \rightarrow def(B)$ formula frogatu behar da.

Aurrekoa kontuan hartuz, *if-then-else* egitura duten baldintzazko aginduei buruzko baieztapenak frogatzeko inferentzi erregela berri bat defini daiteke. Erregela hori *baldintzaren erregela* izendatuko dugu eta honela formaliza daiteke:

(BDE)

$$\frac{\varphi \rightarrow def(B),\ \{\, \varphi \wedge B \,\}\ \mathtt{P_1}\ \{\, \psi \,\},\ \{\, \varphi \wedge \neg B \,\}\ \mathtt{P_2}\ \{\, \psi \,\}}{\{\, \varphi \,\}\ \underline{\mathtt{if}}\ \mathtt{B}\ \underline{\mathtt{then}}\ \mathtt{P_1}\ \underline{\mathtt{else}}\ \mathtt{P_2}\ \underline{\mathtt{end\ if}};\ \{\, \psi \,\}}$$

Adibidez, hurrengo baieztapena betetzen dela froga daiteke baldintzaren erregela erabiliz:

$\{\, True \,\}$
```
if x >= y then
  z := x-y;
else
  z := y-x;
end if;
```
$\{\, z = |x-y| \,\}$

Ondoren, aurreko baieztapenaren frogapen formal eta zehatz bat proposatzen da:

1. $True \rightarrow \underbrace{True}_{def(x \geq y)}$
2. $(\, True \wedge x \geq y \,) \rightarrow (\, x - y = |x-y| \,)$
3. $\{\, x - y = |x-y| \,\}$
   `z := x-y;` **(EA)**
   $\{\, z = |x-y| \,\}$
4. $\{\, True \wedge x \geq y \,\} \equiv \{\, x \geq y \,\}$
   `z := x-y;` 2, 3 eta **(ODE)**
   $\{\, z = |x-y| \,\}$
5. $(\, True \wedge x < y \,) \rightarrow (\, y - x = |x-y| \,)$
6. $\{\, y - x = |x-y| \,\}$
   `z := y-x;` **(EA)**
   $\{\, z = |x-y| \,\}$
7. $\{\, True \wedge x < y \,\} \equiv \{\, x < y \,\}$
   `z := y-x;` 5, 6 eta **(ODE)**
   $\{\, z = |x-y| \,\}$
8. $\{\, True \,\}$
   ```
   if x >= y then
     z := x-y;
   else
     z := y-x;
   end if;
   ```
   1, 4, 7 eta **(BDE)**
   $\{\, z = |x-y| \,\}$

Baldintzazko agindu batzuek *else* kasurik ez dute, eta *if-then* egitura dutela esaten da:

```
if B then
  P
end if;
```

Egitura hori duten baldintzazko aginduen semantika operazionala honako hau da: B baldintza betetzen bada, orduan P programa edo agindu-segida exekutatuko da. Baina B betetzen ez bada, baldintzazko agindua agindurik exekutatu gabe bukatuko da eta baldintzazko agindu horren ondoren dagoen aginduan jarraituko da. Beraz, *if-then* egitura duten baldintzazko aginduei buruzko baieztapenetan

$\{ \varphi \}$
```
  if B then
    P
  end if;
```
$\{ \psi \}$

φ formula betetzen duten baina B baldintza betetzen ez duten egoera guztietan, ψ formulak egiazkoa izan beharko du, hau da, honako inplikazio honek egiazkoa izan beharko du:

$$(\varphi \wedge \neg B) \rightarrow \psi$$

Formula horrek *then* blokeko agindu-segida exekutatzen ez duten konputazioak errepresentatzen ditu, bestela esanda, *else* kasutik jarraituko luketenak. Aurreko formula hori erabiliz, *if-then* egitura duten baldintzazko aginduetarako baldintzaren erregela honela definitzen da:

(BDE)
$$\frac{\varphi \rightarrow def(B),\ \{ \varphi \wedge B \}\ \texttt{P}\ \{ \psi \},\ (\varphi \wedge \neg B) \rightarrow \psi}{\{ \varphi \}\ \texttt{if B then P end if;}\ \{ \psi \}}$$

Baldintzaren erregelaren formulazio berri horrek honako propietate hau frogatzea ahalbidetzen du:

$\{ x = a \}$
```
  if x < 0 then
    x := -x;
  end if;
```
$\{ x = |a| \}$

Frogapena:

1. $(x = a) \rightarrow \underbrace{True}_{def(x<0)}$
2. $(x = a \wedge x < 0) \rightarrow (-x = |a|)$
3. $\{ -x = |a| \}$
   `x := -x;`
   $\{ x = |a| \}$ **(EA)**

4. $\{\, x = a \wedge x < 0 \,\}$
   `x := -x;` 1, 2 eta **(ODE)**
   $\{\, x = |a| \,\}$
5. $(\, x = a \wedge x \geq 0 \,) \rightarrow (\, x = |a| \,)$
6. $\{\, x = a \,\}$
   `if x < 0 then`
   `  x := -x;`
   `end if;` 1, 4, 5 eta **(BDE)**
   $\{\, x = |a| \,\}$

if-then-else eta *if-then* egiturez gain, baldintzazko aginduek beste egitura batzuk ere izan ditzakete, gainera egitura horiek ikusi ditugunak baino konplexuagoak ere izan daitezke. Adibidez, adarkatze anizkoitza izan dezakete, honako agindu honetan bezala:

```
if B1 then
  P1
elsif B2 then
  P2
else
  P3
end if;
```

Agindu horren kasuan, B_1 betetzen bada, P_1 exekutatuko da, baina B_1 betetzen ez bada, orduan B_2 baldintza ebaluatu, eta B_2 egiazkoa bada, P_2 exekutatuko da. Aldiz, B_1 eta B_2 betetzen ez badira, P_3 exekutatuko da. Kontuan izan B_1 betetzen bada, B_2 baldintza ez dela ebaluatuko. Aurreko egitura duten baldintzazko aginduei buruzko baieztapenak frogatzeko honako baldintzazko erregela hau erabil daiteke:

$$\frac{\begin{array}{c}\varphi \rightarrow def(B_1),\ \{\, \varphi \wedge B_1 \,\}\ \mathtt{P_1}\ \{\, \psi \,\},\\ (\, \varphi \wedge \neg B_1 \,) \rightarrow def(B_2),\ \{\, \varphi \wedge \neg B_1 \wedge B_2 \,\}\ \mathtt{P_2}\ \{\, \psi \,\},\\ \{\, \varphi \wedge \neg B_1 \wedge \neg B_2 \,\}\ \mathtt{P_3}\ \{\, \psi \,\}\end{array}}{\{\, \varphi \,\}\ \mathtt{if\ B_1\ then\ P_1\ elsif\ B_2\ then\ P_2\ else\ P_3\ end\ if;}\ \{\, \psi \,\}}$$

Orain arte atal honetan aztertutako aginduek erakutsi digutenez, baldintzazko aginduei buruzko propietateak frogatzea ahalbidetzen duten erregelen formulazioak baldintzazko agindu horien egiturara egokituta daude. Hau da, aginduen egitura aztertuz erabaki behar dugu zein propietate frogatu behar diren, baldintzazko agindu osoari dagokion propietatea frogatzeko. Esan daiteke, hortaz, ez dagoela baldintzazko erregela bakarra, baizik eta baldintzazko aginduaren egiturak ezartzen duela inferentzi erregelak nolakoa izan

behar duen, esleipenaren axioma eta esleipenekin gertatzen den bezalaxe. Baina, bestalde, badira edozein baldintzazko aginduri buruzko propietateak frogatzean beti egiaztatu behar diren bi propietate: lehenengoa, edozein adar edo bidetatik joanda ere bukaeran postbaldintza beteko dela, eta, bigarrena, ebaluatuko diren baldintza guztiak definituta daudela.

Atal honi bukaera emateko, honako baieztapen hau frogatuko dugu:

$\{ True \}$
```
if y <= z then
  if x <= y then
    m := x;
  else
    m := y;
  end if;
elsif x <= z then
  m := x;
else
  m := z;
end if;
```
$\{ m = txikiena(x,y,z) \}$

Frogapen honetan, $def(B)$ eta $def(t)$ formulei buruzko urratsak ez dira adieraziko, formula horiek guztiak $True$ baitira.

1. $(x \leq y \leq z) \rightarrow (x = txikiena(x,y,z))$
2. $\{ x = txikiena(x,y,z) \}$
   `m := x;` **(EA)**
   $\{ m = txikiena(x,y,z) \}$
3. $\{ x \leq y \leq z \}$
   `m := x;` 1, 2 eta **(ODE)**
   $\{ m = txikiena(x,y,z) \}$
4. $(y \leq z \wedge y < x) \rightarrow (y = txikiena(x,y,z))$
5. $\{ y = txikiena(x,y,z) \}$
   `m := y;` **(EA)**
   $\{ m = txikiena(x,y,z) \}$
6. $\{ y \leq z \wedge y < x \}$
   `m := y;` 4, 5 eta **(ODE)**
   $\{ m = txikiena(x,y,z) \}$
7. $(x \leq z < y) \rightarrow (x = txikiena(x,y,z))$

8. $\{\ x = txikiena(x,y,z)\ \}$
   `m := x;` **(EA)**
   $\{\ m = txikiena(x,y,z)\ \}$
9. $\{\ x \leq z < y\ \}$
   `m := x;` 7, 8 eta **(ODE)**
   $\{\ m = txikiena(x,y,z)\ \}$
10. $(\ y > z \wedge x > z\ ) \rightarrow (\ z = txikiena(x,y,z)\ )$
11. $\{\ z = txikiena(x,y,z)\ \}$
    `m := z;` **(EA)**
    $\{\ m = txikiena(x,y,z)\ \}$
12. $\{\ y > z \wedge x > z\ \}$
    `m := z;` 10, 11 eta **(ODE)**
    $\{\ m = txikiena(x,y,z)\ \}$
13. $\{\ True\ \}$

```
if y <= z then
  if x <= y then
    m := x;
  else
    m := y;
  end if;
elsif x <= z then
  m := x;
else
  m := z;
end if;
```

3, 6, 9, 12 eta **(BDE)**

$\{\ m = txikiena(x,y,z)\ \}$

3.11. atalean, frogapen hau erabiliko dugu programen dokumentazioa ilustratzeko.

3.8. ARIKETAK: BALDINTZAZKO AGINDUAK

1. Osatu honako baieztapen hau postbaldintza egokia emanez ($\{\ \_\_\ \}$):

$\{\ 1 \leq i \leq n \wedge z = \mathcal{N}k\ (\ 1 \leq k < i \wedge A(k) > B(k)\ )\ \}$

```
if A(i) > B(i) then
  z := z+1;
end if;
```

{ _____ }

2. Bete hutsuneak (____) honako hirukote honi dagokion frogapenean:

$\{\ 1 \le i < n \wedge \neg dago \wedge \neg badago(A(1..i), x)\ \}$
```
  i := i+1;
  if A(i) = x then
    dago := true;
  end if;
```
$\{\ 1 \le i \le n \wedge (\ dago \leftrightarrow badago(A(1..i), x)\ )\ \}$

Frogapena:

1. $(\ 1 \le i < n \wedge \neg dago \wedge \neg badago(A(1..i), x)\ )$
   $\rightarrow\ (\ 1 \le i+1-1 < n \wedge \neg dago \wedge$
   $\neg badago(A(1..i+1-1), x)\ )$
2. $\{\ 1 \le i+1-1 < n \wedge \neg dago \wedge \neg badago(A(1..i+1-1), x)\ \}$
   `i := i+1;` **(EA)**
   $\{$ _____ $\}$
3. $(\ 1 \le i-1 < n \wedge \neg dago \wedge \neg badago(A(1..i-1), x)\ )$
   $\rightarrow\ (\ 1 \le i \le n \wedge \neg dago \wedge \neg badago(A(1..i-1), x)\ )$
4. $\{\ 1 \le i < n \wedge \neg dago \wedge \neg badago(A(1..i), x)\ \}$
   `i := i+1;` 1, 2, 3 eta **(ODE)**
   $\{$ _____ $\}$
5. $(\ 1 \le i \le n \wedge \neg dago \wedge \neg badago(A(1..i-1), x) \wedge A(i) = x\ )$
   $\rightarrow\ (\ 1 \le i \le n \wedge badago(A(1..i), x)\ )$
   $\rightarrow\ (\ 1 \le i \le n \wedge (\ true \leftrightarrow badago(A(1..i), x)\ )\ )$
6. $\{$ _____ $\}$
   `dago := true;` **(EA)**
   $\{$ _____ $\}$
7. $\{\ 1 \le i \le n \wedge \neg dago \wedge \neg badago(A(1..i-1), x) \wedge A(i) = x\ \}$
   `dago := true;` 5, 6 eta **(ODE)**
   $\{\ 1 \le i \le n \wedge (\ dago \leftrightarrow badago(A(1..i), x)\ )\ \}$
8. (_____)
   $\rightarrow\ (\ 1 \le i \le n \wedge \neg dago \wedge \neg badago(A(1..i), x)\ )$
   $\rightarrow\ (\ 1 \le i \le n \wedge (\ dago \leftrightarrow badago(A(1..i), x)\ )\ )$
9. $(\ 1 \le i \le n \wedge \neg dago \wedge \neg badago(A(1..i-1), x)\ )$
   $\rightarrow\ (\ \_\_\_\_\_\ )\ \rightarrow\ def(A(i) = x)$

10. $\{\ 1 \le i \le n \wedge \neg dago \wedge \neg badago(A(1..i-1), x)\ \}$
```
if A(i) = x then
  dago := true;
end if;
```
$\{\ 1 \le i \le n \wedge (\ dago \leftrightarrow badago(A(1..i), x)\ )\ \}$

11. $\{\ 1 \le i < n \wedge \neg dago \wedge \neg badago(A(1..i), x)\ \}$
```
i := i+1;
if A(i) = x then
  dago := true;
end if;
```
4, 10 eta **(KPE)**
$\{\ 1 \le i \le n \wedge (\ dago \leftrightarrow badago(A(1..i), x)\ )\ \}$

3. Egiaztatu era formalean honako baieztapen hauek:

3.1. $\{\ 1 < i \le n \wedge handiena(A(1..i-1), m)\ \}$
```
if m < A(i) then
  m := A(i);
end if;
i := i+1;
```
$\{\ 1 < i \le n+1 \wedge handiena(A(1..i-1), m)\ \}$

3.2. $\{\ m = i * j + k + 1\ \}$
```
if j = k+1 then
  i := i+1;
  k := 0;
else
  k := k+1;
end if;
```
$\{\ m = i * j + k\ \}$

4. Honako bi baieztapen hauetatik zuzena denarentzat, eman dagokion zuzentasun-froga, eta, zuzena ez denarentzat, eman kontraadibide bat zuzena ez dela erakusteko:

(A) $\{\ z = \mathcal{N}j\ (\ 1 \le j < i \wedge A(j) > B(j)\ )\ \wedge\ 1 \le i < n\ \}$
```
if A(i) > B(i) then
  z := z+1;
end if;
i := i+1;
```
$\{\ z = \mathcal{N}j\ (\ 1 \le j < i \wedge A(j) > B(j)\ )\ \wedge\ 1 \le i \le n\ \}$

(B) $\{\, z = \mathcal{N}j\ (\ 1 \le j < i \wedge A(j) > B(j)\ ) \wedge 1 \le i < n \,\}$

```
  i := i+1;
  if A(i) > B(i) then
    z := z+1;
  end if;
```

$\{\, z = \mathcal{N}j\ (\ 1 \le j < i \wedge A(j) > B(j)\ ) \wedge 1 \le i \le n \,\}$

(A) baieztapena zuzena da eta (B) ez da zuzena [ ]

(B) baieztapena zuzena da eta (A) ez da zuzena [ ]

3.9. INBARIANTEAK ETA ITERAZIOEN ZUZENTASUN PARTZIALA

Orain arte frogatu ditugun zuzentasun partzialeko propietateak behin bakarrik exekutatzen diren agindu-segidei buruzkoak izan dira. Hala ere, agindu iteratiboak ere badaudenez, oso ohikoa da agindu-segida batzuk behin baino gehiagotan exekutatzea. Atal honetan, *while* aginduei buruzko zuzentasun partzialeko propietateak landuko ditugu. Oro har, *while* aginduek honako egitura hau dute:

```
while B loop
  P
end loop;
```

Era horretako aginduen semantika operazionala honako hau da: B baldintza betetzen bada, orduan P agindu-segida exekutatuko da, eta P-ren exekuzioa bukatu ondoren B baldintza berriro aztertuko da. Eta, horrela, behin eta berriz errepikatuko da prozesua B baldintza betetzen den bitartean, hau da, B baldintza faltsua izan arte.

Era horretan, iterazioa osatzen duten agindu-segida errepikatuz burutzen du agindu iteratiboak bere egitekoa. Iterazioa zentzuzkoa bada, aldaketak aldaketa, bada egindakoa islatzen duen eta prozesuan zehar kontserbatzen den propietate bat. Adibidez, honako baieztapen honek x aldagaian x eta y aldagaien batura kalkulatuko dela adierazten du:

$\{\, x = a \wedge y = b \ge 0 \,\}$

```
  while y /= 0 loop
    x := x+1;
    y := y-1;
  end loop;
```

$\{\, x = a + b \,\}$

Batuketa urratsez urrats egiten da: iterazio bakoitzean bat gehitzen zaio x-ri eta bat kentzen zaio y-ri. Hori horrela, begizta bakoitzean exekutatzen den agindu-segidak honako propietate hau kontserbatzen du:

$$x + y = a + b \wedge b \geq y \geq 0$$

Beste baieztapen honetan

$\{ x \geq 1 \wedge y = 1 \}$
```
  while 2*y <= x loop
    y := 2*y;
  end loop;
```
$\{ y = handiena\{k \mid ber2(k) \wedge k \leq x\} \}$

x baino txikiagoa edo berdina den 2ren berretura handiena y aldagaian itzuliko dela adierazten da. Postbaldintzari dagokionez, *ber*2 predikatuaren balioa egiazkoa izango da berari argumentu gisa emandako balioa 2ren berretura denean. Hasieran, y aldagaia 1 balioarekin hasieratu da, eta iterazio bakoitzean haren balioa bikoiztu egiten da. Beraz, iterazioak kontserbatzen duen propietatea honako hau da:

$$ber2(y) \wedge y \leq x$$

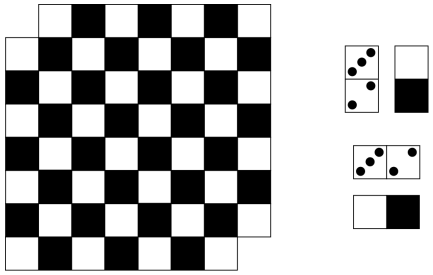
Hirugarren adibide honetan

$\{ n \geq 1 \wedge i = 0 \wedge b = 0 \}$
```
  while i /= n loop
    i := i+1;
    b := b+A(i);
  end loop;
```
$\{ b = \sum_{j=1}^{n} A(j) \}$

iterazioak zenbaki osozko $A(1..n)$ bektoreko elementuen batura b aldagaian kalkulatuko duela adierazten da. Horretarako, iterazio bakoitzean bektoreko elementu bat batzen zaio b aldagaiari. Portaera hori adierazten duen eta iterazioak kontserbatzen duen propietatea honako hau da:

$$b = \sum_{j=1}^{i} A(j) \wedge 0 \leq i \leq n$$

Ikusi ditugun hiru adibide horietan, iterazioaren gorputzeko P agindu-segidak aldagai batzuk aldatzen ditu, baina balio berriek kontserbatu egiten dute aurrekoek betetako propietatea. Hau da, iterazioz iterazio propietatea kontserbatu egiten da. *while* aginduko gorputzak kontserbatzen duen propietate horri *inbariante* esaten zaio. Garbi utzi behar da inbariantea ez dela



3.1. irudia. Dominoaren adibidea.

P-k kontserbatzen duen edozein propietate, baizik eta iterazioaren semantika adierazten duena, hain zuzen ere. Azkeneko adibidera itzuliz, hau da, $A(1..n)$ taulako elementuen batuketa kalkulatzen duen iteraziora itzuliz, $(0 \leq i \leq n)$ propietatea ere kontserbatzen da. Hala ere, azken propietate horrek iterazioaren eginkizunari buruz ezer askorik adierazten ez duenez, ezin dezakegu eman inbariantetzat, iterazioak $A(1..n)$ bektoreko elementuen batuketa itzultzen duela frogatzeko.

Zuzentasun partzialeko propietateak frogatzeko orduan, inbariantea *while* bateko B baldintza ebaluatu aurretik betetzen den asertzioa da:

$\{\, \varphi \,\}$
```
while  { INB }
 B
loop
  P
end loop;
```
$\{\, \psi \,\}$

Hau da, inbariantea da iterazio bakoitza exekutatzen hasi aurretik betetzen den asertzioa, besteak beste bai konputazioa lehenengo aldiz puntu horretatik igarotzen denean (P-ko agindurik oraindik exekutatu ez denean) eta bai azkeneko aldiz igarotzen denean ere (B baldintzaren ebaluazioak faltsua dela adierazten duenean, eta, ondorioz, *while* aginduaren exekuzioa bukatzen denean). B baldintza betetzen denean P-ren exekuzioak inbariantea betearazten duenez, inbariantea kontserbatu egiten da. Eta horrela izango da *while* aginduaren exekuzioa bukatu arte. Hau da, B baldintzak egiazkoa izateari utzi arte.

Laburbilduz, inbariantea iterazioko gorputzean aldatutako aldagaiak erabiliz iterazioaren semantika adierazten duen eta iterazioko gorputzak kontserbatzen duen formula bat da.

Inbariantearen kontzeptua hobeto ulertzeko eta prozesu iteratiboez arrazoitzeko duen erabilgarritasuna erakusteko asmoz joko bat erabiliko dugu adibide bezala. Eman dezagun xakeko taula bati ezkerreko ertzeko goiko laukia eta eskuineko ertzeko beheko laukia kendu zaizkiola (ikusi 3.1. irudia). Hona hemen erantzun beharreko galdera: dominoko fitxek bi laukitxo estaltzen badituzte eta modu horizontalean nahiz bertikalean jar badaitezke taulan, taula guztiz estali al daiteke dominoko fitxekin? Galdera honi erantzuteko problemak soluziorik ba al duen ala ez pentsatu behar dugu. Azter dezagun egoera: xakeko taula estandarretan 32 laukitxo beltz eta 32 laukitxo zuri daude. Adibide honetako taulan, ordea, 30 laukitxo beltz baino ez daude:

$$zenbatzuri = 32$$
$$zenbatbeltz = 30$$

Hau da, laukitxo zuriak laukitxo beltzak baino bi gehiago dira. Guztira, 62 laukitxo daude, eta, beraz, 31 dominoko fitxa beharko lirateke xakeko taula guztia estaltzeko. Gainera, dominoko fitxa bat jartzen dugun bakoitzean, laukitxo zuri bat eta laukitxo beltz bat estaltzen dira, fitxak ezin direlako diagonalean jarri. Hortaz, lehenengo fitxa jarri ondoren, estali gabeko 31 laukitxo zuri eta 29 laukitxo beltz geldituko dira:

$$zenbatzuri = 32 - 1 = 31$$
$$zenbatbeltz = 30 - 1 = 29$$

Horrek esan nahi du oraindik estali gabeko laukitxo zuriak estali gabeko laukitxo beltzak baino bi gehiago direla. Hurrengo fitxa jartzen dugunean ere erlazio hori mantendu egingo da: estali gabeko 31 laukitxo zuri eta 29 laukitxo beltz. Eta gauza bera gertatuko da hurrengo urratsetan ere. Fitxa beltzen eta zurien kopuruen arteko erlazio hori mantendu egiten da fitxak jarri ahala, eta, horrenbestez, gure problemaren inbariantetzat har daiteke:

$$\mathbf{INB} \equiv (zenbatzuri - 2 = zenbatbeltz)$$

Dominoko 30 fitxa jarri ondoren, laukitxo beltz guztiak estalita daude, eta, beraz, bi laukitxo zuri estali gabe geldituko dira:

$$zenbatzuri = 2$$
$$zenbatbeltz = 0$$

Ezinezkoa denez dominoko fitxa bat bi laukitxo zuritan jartzea, oso erraz ikus daiteke problema honek soluziorik ez duela. Ondorioz, hasierako galderaren erantzuna ezezkoa da.

Xakeko taularen adibidea erabiliz, jarraitu beharreko prozesu errepikakorraren inbariantea zehazteak galdera baten erantzun justifikatua aurkitzea errazten duela ikusi dugu. Fitxa beltzen eta zurien kopuruen arteko erlazioak ondo deskribatu du prozesu horretan zer gertatzen zen, eta bidea eman dio erantzun argi bati: hasieran laukitxo zuriak beltzak baino bi gehiago dira, fitxa bat jartzen den bakoitzean bi kopuru horien arteko aldeari eutsi egiten zaio, eta fitxa gehiago ezin denean jarri estali gabeko bi laukitxo zuri eta zero laukitxo beltz gelditzen dira. Beraz, ezin dira laukitxo guztiak estali.

Adibide horretako ideiari jarraituz, iterazioei buruzko propietateak froga daitezke, partikularki *while* motako aginduak. Helburu horrekin *while-aren erregela* honela definitzen da:

(WHE)

$$\frac{\varphi \rightarrow INB,\ INB \rightarrow def(B),\quad \{\, INB \wedge B \,\}\ \texttt{P}\ \{\, INB \,\},\ (\, INB \wedge \neg B \,) \rightarrow \psi}{\{\, \varphi \,\}\ \underline{\texttt{while}}\ \texttt{B}\ \underline{\texttt{loop}}\ \texttt{P}\ \underline{\texttt{end loop}};\ \{\, \psi \,\}}$$

Erregela horrek *while* aginduei buruzko zuzentasun partzialeko propietateak frogatzeko honako lau propietate hauek frogatzea eskatzen du:

- Inbarianteak φ aurrebaldintzaren ondorioa izan behar du. Beste era batean esanda, konputazioa lehenengo aldiz iteraziora iristen denean inbarianteak egiazkoa izan behar du.
- Inbariantea betetzen denean B baldintza definituta dago.
- B baldintza betetzen duen egoera batetik hasten den P agindu-segidaren exekuzioak inbariantea kontserbatzen du.
- Inbariantea betetzen bada eta B baldintza betetzen ez bada, orduan ψ postbaldintza beteko da.

Lehenengo hiru puntuei esker, iterazio bakoitzaren hasieran B baldintza definituta egongo da eta inbariantea bete egingo da.

Kontuan izan erregela horren bidez zuzentasun partzialeko baieztapen bat frogatzen dela. Hau da, ez dugu frogatzen B-ren ebaluazioak noizbait faltsua itzuliko duenik, iterazioaren exekuzioa bukaraziz. Laugarren premisaren kasuan, agindu iteratiboa bukatzen bada ψ postbaldintza beteko dela frogatzen da, baina ez da frogatzen agindu iteratiboa bukatuko den ala ez. 3.13. atalean iterazioen bukaeraren problema landuko dugu, eta, horrela, iterazioei buruzko zuzentasun osoko propietateak frogatu ahal izango ditugu.

Ikus dezagun orain iterazio bati buruzko zuzentasun partzialeko propietate bat frogatzeko *while*-aren erregela nola erabiltzen den. Horretarako, x

eta y aldagaien hasierako balioen batuketa x aldagaian itzultzen duen adibidea erabiliko dugu. Egiaztapena bideratzeko lehen urratsa inbariante bat definitzea da (xakeko taularen probleman egin dugun bezala):

$\{\, \varphi \equiv x = a \wedge y = b \geq 0 \,\}$
`while` $\{\, INB \equiv x + y = a + b \wedge 0 \leq y \leq b \,\}$
`y /= 0`
`loop`
`x := x+1;`
`y := y-1;`
`end loop;`
$\{\, \psi \equiv x = a + b \,\}$

Proposatutako inbariantean oinarrituta eta *while*-aren erregela ardatz hartuta, frogapena honela egin daiteke:

1. $(\underbrace{x = a \wedge y = b \geq 0}_{\varphi}) \rightarrow (\underbrace{x + y = a + b \wedge 0 \leq y \leq b}_{INB})$
2. $(\underbrace{x + y = a + b \wedge 0 \leq y \leq b}_{INB}) \rightarrow \underbrace{True}_{def(y \neq 0)}$
3. $(\underbrace{x + y = a + b \wedge 0 \leq y \leq b}_{INB} \wedge \underbrace{y \neq 0}_{B})$
   $\rightarrow ((x+1) + (y-1) = a + b \wedge 0 \leq y - 1 \leq b)$
4. $\{\, (x+1) + (y-1) = a + b \wedge 0 \leq y - 1 \leq b \,\}$
   `x := x+1;` **(EA)**
   $\{\, x + (y-1) = a + b \wedge 0 \leq y - 1 \leq b \,\}$
5. $\{\, x + (y-1) = a + b \wedge 0 \leq y - 1 \leq b \,\}$
   `y := y-1;` **(EA)**
   $\{\, x + y = a + b \wedge 0 \leq y \leq b \,\}$
6. $\{\, (x+1) + (y-1) = a + b \wedge 0 \leq y - 1 \leq b \,\}$
   `x := x+1;`
   `y := y-1;` 4, 5 eta **(KPE)**
   $\{\, x + y = a + b \wedge 0 \leq y \leq b \,\}$
7. $\{\, \underbrace{x + y = a + b \wedge 0 \leq y \leq b}_{INB} \wedge \underbrace{y \neq 0}_{B} \,\}$
   `x := x+1;`
   `y := y-1;` 3, 6 eta **(ODE)**
   $\{\, \underbrace{x + y = a + b \wedge 0 \leq y \leq b}_{INB} \,\}$

8. $(\underbrace{x+y=a+b \wedge 0 \leq y \leq b}_{INB} \wedge \underbrace{y=0}_{\neg B}) \rightarrow (\underbrace{x=a+b}_{\psi})$

9. $\{ x=a \wedge y=b \geq 0 \}$
   `while` $\{ x+y=a+b \wedge 0 \leq y \leq b \}$
   ```
   y /= 0
   loop
     x := x+1;
     y := y-1;
   end loop;
   ```
   1, 2, 7, 8 eta **(WHE)**
   $\{ x=a+b \}$

Jarraian, x baino txikiagoa edo berdina den 2ren berretura handiena itzultzen duen programari buruzko zuzentasun partzialeko hurrengo propietatea era berean froga daitekeela erakutsiko dugu:

$\{ \varphi \equiv x \geq 1 \wedge y=1 \}$
`while` $\{ INB \equiv ber2(y) \wedge 1 \leq y \leq x \}$
```
2*y <= x
loop
  y := 2*y;
end loop;
```
$\{ \psi \equiv y = handiena\{k \mid ber2(k) \wedge k \leq x\} \}$

Frogapen formala honako hau da:

1. $(\underbrace{x \geq 1 \wedge y=1}_{\varphi}) \rightarrow (\underbrace{ber2(y) \wedge 1 \leq y \leq x}_{INB})$
2. $(\underbrace{ber2(y) \wedge 1 \leq y \leq x}_{INB}) \rightarrow \underbrace{True}_{def(2*y \leq x)}$
3. $(\underbrace{ber2(y) \wedge 1 \leq y \leq x}_{INB} \wedge \underbrace{2*y \leq x}_{B}) \rightarrow (ber2(2*y) \wedge 1 \leq 2*y \leq x)$
4. $\{ ber2(2*y) \wedge 1 \leq 2*y \leq x \}$
   ```
   y := 2*y;
   ```
   (EA)
   $\{ \underbrace{ber2(y) \wedge 1 \leq y \leq x}_{INB} \}$
5. $\{ \underbrace{ber2(y) \wedge 1 \leq y \leq x}_{INB} \wedge \underbrace{2*y \leq x}_{B} \}$
   ```
   y := 2*y;
   ```
   3, 4 eta **(ODE)**
   $\{ \underbrace{ber2(y) \wedge 1 \leq y \leq x}_{INB} \}$

6. $(\underbrace{ber2(y) \wedge 1 \leq y \leq x}_{INB} \wedge \underbrace{2*y > x}_{\neg B}) \rightarrow$
$(\underbrace{y = handiena\{k \mid ber2(k) \wedge k \leq x\}}_{\psi})$

7. $\{ x \geq 1 \wedge y = 1 \}$
```
while 2*y <= x loop
  y := 2*y;
end loop;
```
1, 2, 5, 6 eta **(WHE)**
$\{ y = handiena\{k \mid ber2(k) \wedge k \leq x\} \}$

3.10. ARIKETAK: INBARIANTEAK ETA ITERAZIOEN ZUZENTASUN PARTZIALA

1. Aukeratu inbariante zuzena honako baieztapen honentzat. Programak x elementua $A(1..n)$ bektorean agertzen den ala ez erabakitzen du, erantzuna *dago* aldagai boolearrean utziz.

$\{ n \geq 1 \}$
```
i := 0;
dago := false;
while  { INB }
 not dago and i < n
loop
  i := i+1;
  if A(i) = x then
    dago := true;
  end if;
end loop;
```
$\{ dago \leftrightarrow \exists j \, (\, 1 \leq j \leq n \wedge A(j) = x \,) \}$

(*a*) **INB** $\equiv \, (\, dago \leftrightarrow \exists j \, (\, 1 \leq j \leq i \wedge A(j) = x \,) \,) \wedge 0 \leq i \leq n$ [ ]

(*b*) **INB** $\equiv \, (\, dago \wedge \exists j \, (\, 1 \leq j \leq i \wedge A(j) = x \,) \,) \wedge 0 \leq i \leq n$ [ ]

2. Frogatu honako programa honen zuzentasun partziala. Programak, x negatiboa ez izanda, x eta y zenbaki osoen biderkadura kalkulatzen du.

```
z := 0;
while x /= 0 loop
  z := z+y;
  x := x-1;
end loop;
```

3.11. ASERTZIOAK ETA PROGRAMEN DOKUMENTAZIOA

Programen zuzentasuna frogatzeko ez ezik, programak dokumentatzeko ere erabil daitezke asertzioak, programen exekuzioaren propietate nagusiak adierazten baitituzte: hau da, programek egin beharreko lana nola egiten duten eta beraien helburua zergatik betetzen duten erakusten duen arrazoibide logikoaren oinarriak dira. Programazio-lengoaia batzuek ez dituzte asertzioak prozesatzen, baina kasu horietan ere asertzioak idaztea oso onuragarria da; izan ere, asertzioak programen mantentze-fasean oso lagungarriak eta zehatzak izan daitezkeen iruzkintzat har daitezke. Hala ere, askoz komenigarriagoa da asertzioak programazio-lengoaiaren barruko adierazpen gisa erabiltzea eta sistemak automatikoki egiaztatzea. Programazio-lengoaia batzuetan asertzioak programa exekutatzean egiaztatzen dira (adibidez, Eiffel), eta, beste batzuetan, programa konpilatzean (adibidez, Dafny). Asertzioen bidezko programen dokumentazioa zuzentasun-frogaren eskematzat edo laburpentzat har daiteke. Hain zuzen ere, frogapenaren urratsak programatzaileek idatzitako asertzioetatik abiatuta egiaztatzen dira Dafny bezalako lengoaietan. Programen dimentsio eta konplexutasuna handitzen diren heinean, eskema edo laburpen horiek erabiltzea beharrezkoagoa egiten da, baina betiere eskema horiek justifikatzeko automatikoki edo eskuz egindako frogapen formal bat behar dela kontuan hartuz.

Adibidez, asertzioak erabiltzen dira hurrengo programa dokumentatzeko:

$\{\, True \,\}$
```
  if y <= z then
    if x <= y then
```
$\{\, x \leq y \leq z \,\}$
```
      m := x;
    else
```
$\{\, y \leq z \wedge y < x \,\}$
```
      m := y;
    end if;
  elsif x <= z then
```
$\{\, x \leq z < y \,\}$
```
    m := x;
  else
```
$\{\, y > z \wedge x > z \,\}$
```
    m := z;
  end if;
```
$\{\, m = txikiena(x, y, z) \,\}$

Asertzio horiek programaren zuzentasunaren adierazle oso egokiak dira. 3.7. atalaren bukaeran baieztapen horren zuzentasun partzialaren frogapena agertzen da.

Ondoren, beste bi adibide aztertuko ditugu. Adibide horietan bi programa iteratibo dokumentatuko dira asertzioak erabiliz, inbarianteak barne. Programa horien zuzentasun partzialeko propietateen edo baieztapenen frogapenak ere proposatzen dira, asertzioak frogapenetatik atera direla erakutsiz.

Lehenengo adibidean, programak $A(1..n)$ bektorean agertzen diren zeroen kopurua kalkulatzen du zk aldagaian:

$\{\, n \geq 1 \,\}$
```
  i := 0;
  zk := 0;
  while
```
$\{\, INB \equiv zk = \mathcal{N}j\,(\,1 \leq j \leq i \wedge A(j) = 0\,) \wedge 0 \leq i \leq n \,\}$
```
   i < n
  loop
    i := i+1;
    if A(i) = 0 then
      zk := zk+1;
    end if;
  end loop;
```
$\{\, zk = \mathcal{N}j\,(\,1 \leq j \leq n \wedge A(j) = 0\,) \,\}$

Hasteko, programaren aurre-ondoetako espezifikazioa eta iterazioaren inbariantea adierazten dira. Erraz ikus daitekeenez, lehenengo bi aginduetan i eta zk aldagaiak zerorekin hasieratu ondoren inbariantea bete egiten da. Gainera, i aldagaiaren balioa n baino handiagoa edo berdina denean postbaldintza beteko dela bermatzen du inbarianteak. Aurreko bi propietate hauen frogapen zehatza jakin-mina duten irakurleentzat uzten da. Ondoren, i aldagaia n baino txikiagoa denean iterazioaren gorputzak inbariantea kontserbatu egiten duela frogatuko dugu. Hau da, hurrengo baieztapena frogatuko da:

$\{\, zk = \mathcal{N}j\,(\,1 \leq j \leq i \wedge A(j) = 0\,) \wedge 0 \leq i < n \,\}$
```
  i := i+1;
  if A(i) = 0 then
    zk := zk+1;
  end if;
```
$\{\, zk = \mathcal{N}j\,(\,1 \leq j \leq i \wedge A(j) = 0\,) \wedge 0 \leq i \leq n \,\}$

Iterazioaren gorputza bi aginduz osatuta dago: lehenengoa, esleipen bat da eta, bigarrena, baldintzazko agindu bat. Beraz, baieztapen hori frogatzeko,

batetik, esleipenaren axioma (beharrezkoa baldin bada ondorioaren erregelarekin batera) eta baldintzaren erregela erabili beharko ditugu, eta, ondoren, konposaketaren erregela. Esleipena exekutatu ondoren ateratzen den postbaldintza konposaketaren erregelak erabiltzen duen tarteko asertzioa izango da, baita baldintzazko aginduaren aurrebaldintza ere. Gainera, baldintzazko aginduak *if-then* egitura duenez, hurrengo bi propietate frogatu beharko dira: baldintzazko aginduaren baldintza egiazkoa bada, *then* atala exekutatu ondoren postbaldintza betetzen dela; bestela, baldintzaren ukapenaren eta tarteko asertzioaren konjuntzioak postbaldintza inplikatzen duela. Laburbilduz, aurreko zuzentasun partzialeko propietatearen frogapena hurrengo hiru asertzioen bidez adierazten diren hiru propietateetan datza:

1. Tarteko asertzioa.

2. *then* atalaren aurrebaldintza (baldintzazko aginduaren baldintza egiazkoa den kasua).

3. Baldintzazko aginduaren baldintza faltsua denean betetzen den inplikazio logikoa.

Eta aurreko asertzio horiek erabiliz, iterazioaren gorputza dokumenta daiteke:

$$
\begin{array}{l}
\{\, zk = \mathcal{N}j\ (\, 1 \le j \le i \,\wedge\, A(j) = 0\,)\ \wedge\ 0 \le i < n\,\} \\
\quad \texttt{i := i+1;} \\
\quad \{\, zk = \mathcal{N}j\ (\, 1 \le j \le i-1 \,\wedge\, A(j) = 0\,)\ \wedge\ 1 \le i \le n\,\} \\
\quad \underline{\texttt{if}}\ \texttt{A(i) = 0}\ \underline{\texttt{then}} \\
\qquad \{\, zk+1 = \mathcal{N}k\ (\, 1 \le j \le i \,\wedge\, A(j) = 0\,)\ \wedge\ 0 \le i \le n\,\} \\
\qquad \texttt{zk := zk+1;} \\
\quad \texttt{// } \underline{\texttt{else}}\ \texttt{(ez dago)} \\
\qquad \{\, (\, zk = \mathcal{N}j\ (\, 1 \le j \le i-1 \,\wedge\, A(j) = 0\,)\ \wedge\ 1 \le i \le n \,\wedge\, A(i) \ne 0\,) \\
\qquad\qquad \rightarrow\ (\, zk = \mathcal{N}j\ (\, 1 \le j \le i \,\wedge\, A(j) = 0\,)\ \wedge\ 0 \le i \le n)\,\} \\
\quad \underline{\texttt{end if}}\texttt{;} \\
\{\, zk = \mathcal{N}j\ (\, 1 \le j \le i \,\wedge\, A(j) = 0\,)\ \wedge\ 0 \le i \le n\,\}
\end{array}
$$

Eskema horrek frogapenaren baieztapen garrantzitsuenetariko bat adierazten du, hau da, iterazioaren baldintza betetzen denean iterazioaren gorputzak inbariantea kontserbatu egiten duela. Hala ere, asertzioak ez dira ausaz asmatu, baizik eta Hoareren sistema formalean oinarritzen dira:

1. $(zk = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 0 \le i < n\ )$
   $\rightarrow\ (\ zk = \mathcal{N}j\ (\ 1 \le j \le i+1-1 \wedge A(j) = 0\ ) \wedge$
   $1 \le i+1 \le n\ )$
2. $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i+1-1 \wedge A(j) = 0\ ) \wedge 1 \le i+1 \le n\ \}$
   `i := i+1;` **(EA)**
   $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i-1 \wedge A(j) = 0\ ) \wedge 1 \le i \le n\ \}$
3. $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 0 \le i < n\ \}$
   `i := i+1;` 1, 2 eta **(ODE)**
   $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i-1 \wedge A(j) = 0\ ) \wedge 1 \le i \le n\ \}$
4. $(zk = \mathcal{N}j\ (\ 1 \le j \le i-1 \wedge A(j) = 0\ ) \wedge 1 \le i \le n\ )$
   $\rightarrow\ (\ 1 \le i \le n\ ) \equiv def(A(i) = 0)$
5. $(zk = \mathcal{N}j\ (\ 1 \le j \le i-1 \wedge A(j) = 0\ ) \wedge 1 \le i \le n \wedge A(i) = 0\ )$
   $\rightarrow\ (\ zk+1 = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 0 \le i \le n\ )$
6. $\{\ zk+1 = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 0 \le i \le n\ \}$
   `zk := zk+1;` **(EA)**
   $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 0 \le i \le n\ \}$
7. $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i-1 \wedge A(j) = 0\ ) \wedge 1 \le i \le n \wedge A(i) = 0\ \}$
   `zk := zk+1;` 5, 6 eta **(ODE)**
   $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 1 \le i \le n\ \}$
8. $(zk = \mathcal{N}j\ (\ 1 \le j \le i-1 \wedge A(j) = 0\ ) \wedge 1 \le i \le n \wedge A(i) \neq 0\ )$
   $\rightarrow\ (\ zk = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 0 \le i \le n\ )$
9. $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i-1 \wedge A(j) = 0\ ) \wedge 1 \le i \le n\ \}$
   ```
   if A(i) = 0 then
     zk := zk+1;
   end if;
   ```
   4, 7, 8 eta **(BDE)**
   $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 0 \le i \le n\ \}$
10. $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 0 \le i < n\ \}$
    ```
    i := i+1 then
    if A(i) = 0 then
      zk := zk+1;
    end if;
    ```
    3, 9 eta **(KPE)**
    $\{\ zk = \mathcal{N}j\ (\ 1 \le j \le i \wedge A(j) = 0\ ) \wedge 0 \le i \le n\ \}$

Azkenik, programa osoaren dokumentazioa honako hau da:

$\{\, n \geq 1 \,\}$
```
  i := 0;
  zk := 0;
  while
```
$\{\, INB \equiv zk = \mathcal{N}j\,(\,1 \leq j \leq i \wedge A(j) = 0\,) \wedge 0 \leq i \leq n \,\}$
```
   i < n
  loop
    i := i+1;
```
$\{\, zk = \mathcal{N}j\,(\,1 \leq j \leq i-1 \wedge A(j) = 0\,) \wedge 1 \leq i \leq n \,\}$
```
    if A(i) = 0 then
```
$\{\, zk+1 = \mathcal{N}j\,(\,1 \leq j \leq i \wedge A(j) = 0\,) \wedge 0 \leq i \leq n \,\}$
```
      zk := zk+1;
    // else (ez dago)
```
$\{\, (\, zk = \mathcal{N}j\,(\,1 \leq j \leq i-1 \wedge A(j) = 0\,) \wedge 1 \leq i \leq n \wedge A(i) \neq 0\,)$
$\rightarrow\, (\, zk = \mathcal{N}j\,(\,1 \leq j \leq i \wedge A(j) = 0\,) \wedge 0 \leq i \leq n\,)\,\}$
```
    end if;
  end loop;
```
$\{\, zk = \mathcal{N}j\,(\,1 \leq j \leq n \wedge A(j) = 0\,)\,\}$

Bigarren adibidean, zuzentasun partzialeko baieztapena frogatzeko esleipenaren axioma eta orain arte aurkeztu ditugun lau inferentzi erregelak (ondorioarena, konposaketarena, baldintzarena eta *while*-arena) erabili behar dira:

$\{\, \varphi \equiv x \geq 1 \,\}$
```
  i := 1;
  b := 0;
  while
   i <= x/2
  loop
    if x % i = 0 then
      b := b+i;
    end if;
    i := i+1;
  end loop;
  q := (x=b);
```
$\{\, \psi \equiv q \leftrightarrow x = \sum_{\substack{1 \leq k < x \,\wedge \\ x \,\%\, k = 0}} k \,\}$

Programa horrek positiboa den x zenbaki osoa *betea* den ala ez erabakitzen du q aldagai boolearrean. Kontuan hartu positiboa den x zenbaki oso bat

betea izango dela, zenbaki horren berezko zatitzaileen (zatitzaile positibo guztiak x zenbakia bera izan ezik) batura x baldin bada. Ondoren, zuzentasun partzialeko propietatearen frogapenetik ateratako asertzioen bidez programa dokumentatuko da:

$\{\, \varphi \equiv x \geq 1 \,\}$
```
  i := 1;
```
$\{\, \varphi_1 \equiv x \geq 1 \wedge i = 1 \,\}$
```
  b := 0;
```
$\{\, \varphi_2 \equiv x \geq 1 \wedge i = 1 \wedge b = 0 \,\}$
```
  while
```
$\{\, INB \equiv 1 \leq i \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i \wedge \\ x \,\%\, k = 0}} k \,\}$
```
   i <= x/2
  loop
    if x % i = 0 then
```
$\{\, \varphi_3 \equiv 1 \leq i+1 \leq (x/2)+1 \wedge b+i = \sum_{\substack{1 \leq k < i+1 \wedge \\ x \,\%\, k = 0}} k \,\}$
```
      b := b+i;
    // else (ez dago)
```
$\{\, (\, INB \wedge i \leq x/2 \wedge x \,\%\, i \neq 0 \,) \rightarrow (\, 1 \leq i+1 \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i+1 \wedge \\ x \,\%\, k = 0}} k \,) \,\}$
```
    end if;
```
$\{\, \varphi_4 \equiv 1 \leq i+1 \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i+1 \wedge \\ x \,\%\, k = 0}} k \,\}$
```
    i := i+1;
  end loop;
```
$\{\, \varphi_6 \equiv b = \sum_{\substack{1 \leq k < x \wedge \\ x \,\%\, k = 0}} k \,\}$
```
  q := (x=b);
```
$\{\, \psi \equiv q \leftrightarrow x = \sum_{\substack{1 \leq k < x \wedge \\ x \,\%\, k = 0}} k \,\}$

Ikusten denez, tarteko asertzioak φ_j erako izenekin adierazi dira jarraian emango den frogapena errazago ulertzeko. Frogapenean, i eta b hasieratu ondoren betetzen diren φ_1 eta φ_2 asertzioak goitik beheranzko estrategia erabiliz sortu dira. Frogarekin jarraituz, *while* aginduaren aurrebaldintza φ_2 asertzioa izango da. Hor *while* aginduaren baldintza betetzen denean iterazioak inbariantea kontserbatu egiten duela frogatzeko, *then* blokearen aurrebaldintza den φ_3 formula eta *if* aginduaren baldintza betetzen ez deneko (`else (ez dago)`) kasuari dagokion inplikazioa hartu behar dira kontuan. Baldintzazko aginduaren eta i aldagaiaren balioa eguneratzen duen esleipenaren artean betetzen den φ_4 asertzioa ere ipini da. Bukatzeko, φ_6 asertzioa *while*-aren postbaldintzatzat eta q aldagaiari egiten zaion esleipenaren aurrebaldintzatzat hartu da. Hor φ_5 asertzioa ere erabil daiteke. Era berean, φ_1

eta φ_2 asertzioen ordez, inbariantetik abiatuta goranzko estrategiari jarraituz lor daitezkeen asertzioak ere erabil ditzakegu. Horrela eginez gero, frogapena zertxobait desberdina izango litzateke. Hurrengo frogapenean goitik beheranzko estrategiari jarraitu zaio:

1. $\{\, x \geq 1 \,\}$
   `i := 1;` **(EA)**
   $\{\, \underbrace{x \geq 1 \wedge i = 1}_{\varphi_1} \,\}$
2. $\{\, \underbrace{x \geq 1 \wedge i = 1}_{\varphi_1} \,\}$
   `b := 0;` **(EA)**
   $\{\, \underbrace{x \geq 1 \wedge i = 1 \wedge b = 0}_{\varphi_2} \,\}$
3. $(\, \underbrace{x \geq 1 \wedge i = 1 \wedge b = 0}_{\varphi_2} \,) \rightarrow (\, 1 \leq i \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i \wedge \\ x \,\%\, k = 0}} k \,)$
4. $(\, 1 \leq i \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i \wedge \\ x \,\%\, k = 0}} k \,) \rightarrow \underbrace{True}_{def(i \leq x/2)}$
5. $(\, 1 \leq i \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i \wedge \\ x \,\%\, k = 0}} k \wedge i \leq x/2 \,) \rightarrow \underbrace{(\, i \neq 0 \,)}_{def(x \,\%\, i = 0)}$
6. $(\, 1 \leq i \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i \wedge \\ x \,\%\, k = 0}} k \wedge i \leq x/2 \wedge x \,\%\, i = 0 \,)$
   $\rightarrow \underbrace{(\, 1 \leq i+1 \leq (x/2)+1 \wedge b+i = \sum_{\substack{1 \leq k < i+1 \wedge \\ x \,\%\, k = 0}} k \,)}_{\varphi_3}$
7. $\underbrace{\{\, 1 \leq i+1 \leq (x/2)+1 \wedge b+i = \sum_{\substack{1 \leq k < i+1 \wedge \\ x \,\%\, k = 0}} k \,\}}_{\varphi_3}$
   `b := b+i;` **(EA)**
   $\underbrace{\{\, 1 \leq i+1 \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i+1 \wedge \\ x \,\%\, k = 0}} k \,\}}_{\varphi_5}$
8. $\{\, 1 \leq i \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i \wedge \\ x \,\%\, k = 0}} k \wedge i \leq x/2 \wedge x \,\%\, i = 0 \,\}$
   `b := b+i;` 6, 7 eta **(ODE)**
   $\underbrace{\{\, 1 \leq i+1 \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i+1 \wedge \\ x \,\%\, k = 0}} k \,\}}_{\varphi_5}$

9. $(1 \le i \le (x/2)+1 \wedge b = \sum_{\substack{1 \le k < i \wedge \\ x \,\%\, k = 0}} k \wedge i \le x/2 \wedge x \,\%\, i \neq 0)$
$\rightarrow \underbrace{(1 \le i+1 \le (x/2)+1 \wedge b = \sum_{\substack{1 \le k < i+1 \wedge \\ x \,\%\, k = 0}} k)}_{\varphi_5}$

10. $\{ 1 \le i \le (x/2)+1 \wedge b = \sum_{\substack{1 \le k < i \wedge \\ x \,\%\, k = 0}} k \wedge i \le x/2 \}$
```
if x % i = 0 then
  b := b+i;
end if;
```
5, 8, 9 eta **(BDE)**
$\underbrace{\{ 1 \le i+1 \le (x/2)+1 \wedge b = \sum_{\substack{1 \le k < i+1 \wedge \\ x \,\%\, k = 0}} k \}}_{\varphi_5}$

11. $\underbrace{\{ 1 \le i+1 \le (x/2)+1 \wedge b = \sum_{\substack{1 \le k < i+1 \wedge \\ x \,\%\, k = 0}} k \}}_{\varphi_5}$
```
i := i+1;
```
(EA)
$\underbrace{\{ 1 \le i \le (x/2)+1 \wedge b = \sum_{\substack{1 \le k < i \wedge \\ x \,\%\, k = 0}} k \}}_{INB}$

12. $\{ 1 \le i \le (x/2)+1 \wedge b = \sum_{\substack{1 \le k < i \wedge \\ x \,\%\, k = 0}} k \wedge i \le x/2 \}$
```
if x % i = 0 then
  b := b+i;
end if;
i := i+1;
```
10, 11 eta **(KPE)**
$\{ 1 \le i \le (x/2)+1 \wedge b = \sum_{\substack{1 \le k < i \wedge \\ x \,\%\, k = 0}} k \}$

13. $(1 \le i \le (x/2)+1 \wedge b = \sum_{\substack{1 \le k < i \wedge \\ x \,\%\, k = 0}} k \wedge i > x/2) \rightarrow \underbrace{(b = \sum_{\substack{1 \le k < x \wedge \\ x \,\%\, k = 0}} k)}_{\varphi_6}$

14. $\{ \underbrace{x \ge 1 \wedge i = 1 \wedge b = 0}_{\varphi_2} \}$
```
while i <= x/2 loop
  if x % i = 0 then
    b := b+i;
  end if;
  i := i+1;
end loop;
```
3, 4, 12, 13 eta **(WHE)**
$\underbrace{\{ b = \sum_{\substack{1 \le k < x \wedge \\ x \,\%\, k = 0}} k \}}_{\varphi_6}$

15. $\underbrace{(\, b = \sum_{\substack{1 \le k < x \ \wedge \\ x \,\%\, k = 0}} k \,)}_{\varphi_6} \rightarrow (\, (\, x = b \,) \leftrightarrow x = \sum_{\substack{1 \le k < x \ \wedge \\ x \,\%\, k = 0}} k \,)$

16. $\{\, (\, x = b \,) \leftrightarrow x = \sum_{\substack{1 \le k < x \ \wedge \\ x \,\%\, k = 0}} k \,\}$

```
q := (x=b);
```

(EA)

$\{\, q \leftrightarrow x = \sum_{\substack{1 \le k < x \ \wedge \\ x \,\%\, k = 0}} k \,\}$

17. $\underbrace{\{\, b = \sum_{\substack{1 \le k < x \ \wedge \\ x \,\%\, k = 0}} k \,\}}_{\varphi_6}$

```
q := (x=b);
```

15, 16 eta **(ODE)**

$\underbrace{\{\, q \leftrightarrow x = \sum_{\substack{1 \le k < x \ \wedge \\ x \,\%\, k = 0}} k \,\}}_{\psi}$

18. $\{\, \underbrace{x \ge 1}_{\varphi} \,\}$

```
i := 1;
b := 0;
while  { 1 ≤ i ≤ (x/2)+1 ∧ b = Σ_{1≤k<i ∧ x % k=0} k }
 i <= x/2
loop
  if x % i = 0 then
    b := b+i;
  end if;
  i := i+1;
end loop;
q := (x=b);
```

1, 2, 14, 17 eta **(KPE)**

$\{\, q \leftrightarrow x = \sum_{\substack{1 \le k < x \ \wedge \\ x \,\%\, k = 0}} k \,\}$

3.12. ARIKETAK: ASERTZIOAK ETA PROGRAMEN DOKUMENTAZIOA

Dokumentatu zehazten diren asertzioekin eta inbarianteekin honako programa iteratibo hauek:

1. Programa honek $A(1..n)$ bektorean bakoitiak diren osagaien kopurua eta bikoitiak diren osagaien kopurua berdinak diren ala ez erabakiko du.

{ **Aurre** ≡ _____ }
```
i := 0;
w := 0;
z := 0;
while
```
{ **INB** ≡ _____ }
```
  i < n
loop
  i := i+1;
```
{ φ_1 ≡ _____ }
```
  if ( A(i) % 2 = 0 ) then
```
{ φ_2 ≡ _____ }
```
    w := w+1;
  else
```
{ φ_3 ≡ _____ }
```
    z := z+1;
  end if;
end loop;
```
{ φ_4 ≡ _____ }
```
r := (w=z);
```
{ **Post** ≡ _____ }

2. Programa honek 2 zenbakiaren 0 eta n-ren arteko berreturen batura kalkulatuko du b aldagaian; alegia, 2^0, 2^1, …, 2^n segidako elementuen batura kalkulatuko du.

{ **Aurre** ≡ _____ }
```
i := 0;
p := 1;
b := 1;
while
```
{ **INB** ≡ _____ }
```
  i < n
loop
```
{ φ_1 ≡ _____ }
```
  i := i+1;
```
{ φ_2 ≡ _____ }
```
  p := p*2;
```
{ φ_3 ≡ _____ }
```
  b := b+p;
end loop;
```
{ **Post** ≡ _____ }

3.13. BORNE-ADIERAZPENAK ETA ITERAZIOEN BUKAERA

3.9. atalean Hoareren sistema formala erabili dugu agindu iteratiboei buruzko zuzentasun partzialeko baieztapenak frogatzeko. Baina, 3.1. atalean ikusi dugunez, zuzentasun osoa frogatzeko zuzentasun partziala frogatzeaz gain, iterazioak urrats kopuru finitu baten ondoren bukatu egiten direla frogatu behar da. Iterazioen konputazioa prozesu errepikakorra izan ohi da, eta prozesu hori bukatzea edo ez bukatzea urrats kopuru finitu baten ondoren baldintza jakin bat betetzearen menpe egongo da. Kontuan izatekoa da, bestetik, bukaeraren propietatea hasierako egoerei (aurreko baldintzari) estuki lotuta dagoela. Ondorioz, iterazio baten exekuzioa nola abiatzen den aztertzea funtsezkoa da bukatuko den ala ez erabakitzeko. Hori horrela izanda, P agindu-segida baten bukaera-propietatea aztertzean, P agindu-segida horrez gain, aurrebaldintza ere kontuan hartuko dugu. Zehazki, P agindu-segida eta φ aurrebaldintza emanda, P-ren exekuzioa φ formularekiko bukatu egiten dela esango dugu, baldin eta soilik baldin φ betetzen den egoera batean hasten den P-ren edozein konputazio bukatu egiten bada. Inbariantea ere aurrebaldintzari lotuta dago, aurrebaldintza baita iterazioaren exekuzioa hasi aurretik inbariantea beteko dela bermatzen duena. Jarraian, aurrebaldintzak eta inbarianteak programen bukaera-frogapenetan duten eragina erakusteko adibideak erabiliko ditugu. Gure lehenengo adibidea honako iterazio hau da:

P_1 =
```
while i /= 0 loop
   i := i-1;
end loop;
```

Nabaria da $\{\, True \,\}\ P_1\ \{\, i = 0 \,\}$ zuzentasun partzialeko baieztapena betetzen dela. $True$ aurrebaldintzarekiko P_1-en bukaera-propietatea, ordea, ez da betetzen, $True$ ahulegia baita: $True$ aurrebaldintzak i aldagaiak balio negatiboak izatea onartzen du, baina i negatiboa baldin bada, iterazioa ez da bukatuko, amaierarik gabe ziklatuko du. Beraz, $\{\, True \,\}\ [\ P_1\ ]\ \{\, i = 0 \,\}$ zuzentasun osoko baieztapena ezin da frogatu. Aldiz, $i \geq 0$ aurrebaldintza hartuz gero, $\{\, i \geq 0 \,\}\ P_1\ \{\, i = 0 \,\}$ zuzentasun partzialeko baieztapena eta $i \geq 0$ formularekiko P_1-en bukaera-propietatea bete egiten dira. Ondorioz, $\{\, i \geq 0 \,\}\ [\ P_1\ ]\ \{\, i = 0 \,\}$ zuzentasun osoko baieztapena froga daiteke. Kontuan izan aurrebaldintza $i \geq 0$ baldin bada, $i \geq 0$ bera inbariantetzat ere har daitekeela. Baina aurrebaldintza $True$ baldin bada, ezin da $i \geq 0$ inbariantetzat hartu, $True$ aurrebaldintzak ez baitu hasierako egoeran $i \geq 0$ betetzen denik bermatzen.

Gure bigarren adibidea lehenengoaren aldaera bat da:

```
P2  =  while i /= 0 loop
         i := i-2;
       end loop;
```

Aurreko adibidean bezala, $\{\, True \,\}\ P_2\ \{\, i = 0 \,\}$ zuzentasun partzialeko baieztapena bete arren, P_2-ren bukaera ezin da frogatu $True$ aurrebaldintzatzat hartuz. Baina, orain aurrebaldintzatzat $i \geq 0$ hartuta ere, P_2 bukatuko denik ezin da frogatu, $i \geq 0$ aurrebaldintza ahulegia baita: i-ren hasierako balioa bakoitia denean i ez da inoiz 0 izatera iritsiko. Bestalde, $\{\, i \geq 0 \wedge \mathit{bikoitia}(i) \,\}$ aurrebaldintzarekiko P_2 programaren bukaera-propietatea froga daiteke, eta aurrebaldintza bera inbariantetzat ere har daiteke. Erraz ikus daiteke bai $\mathit{bikoitia}(i)$ propietatea eta bai $i \geq 0$ propietatea ere, kontserbatu egiten direla iteraziotik iteraziora. Beraz, $\{\, i \geq 0 \wedge \mathit{bikoitia}(i) \,\}\ [\ P_2\ ]\ \{\, i = 0 \,\}$ zuzentasun osoko baieztapena bete egiten da. Aurreko ataletan, $\{\, i \geq 0 \wedge \mathit{bikoitia}(i) \,\}\ P_2\ \{\, i = 0 \,\}$ bezalako zuzentasun partzialeko baieztapenen frogapena aztertu da. Ondoren, φ aurrebaldintza eta P programa emanda, φ-rekiko P-ren bukaera-propietatea nola frogatzen den aztertuko dugu. Azalduko dugun metodoari jarraituz, P_1 programa $i \geq 0$ aurrebaldintzarekiko bukatu egiten dela frogatuko dugu, baita $i \geq 0 \wedge \mathit{bikoitia}(i)$ aurrebaldintzarekiko P_2 bukatu egiten dela ere.

Iterazio baten aurrebaldintza inbariantearen kasu partikular bat baino ez da, iterazioaren exekuzioa hasi aurretik betetzen den kasu partikularra hain zuzen ere. Beraz, proposatzen den metodoan iterazioaren bukaera inbariantearekiko frogatuko da.

Honako egitura hau duen *while* iterazio bat emanda,

```
while  { INB }
 B
loop
  P
end loop;
```

iterazio hori noizbait bukatzeko, $INB \wedge B$ betetzen den egoera batean hasita, P agindu-segidak B faltsu egin beharko du aldi kopuru finituan exekutatu ondoren. Hori gertatu ahal izateko, P agindu-segidak B baldintzako aldagairen bat aldatzea beharrezkoa izan arren, ez da nahikoa: aldaketa kopuru finitu baten ondoren B baldintza ez dela beteko bermatu behar du aldaketa horrek. Horretarako, $(\mathbb{N}, \leq)$, ondo oinarritutako ordena duena, hartuko dugu ardatz. Hau da, $\leq$ eragilea $\mathbb{N}$-ren gainean definitutako *ordena* da, erreflexiboa, antisimetrikoa eta iragankorra baita. Gainera, ordena hori *ondo oinarritua* da (hertsiki) beheranzko kate infiniturik ez duelako. Bestela esanda, hertsiki beheranzko edozein kate zero edo handiagoa den elementu batean

bukatzen da:

$$80 > \ldots > 23 > \ldots > 14 > \ldots > 3 > \ldots > 0$$

Ondorioz, balio bezala bakarrik zenbaki arruntak har ditzakeen edozein E espresiok aldi kopuru finituan har ditzake gero eta txikiagoak diren balioak. Eredu formal horretan oinarrituta, *while* aginduen bukaera-propietatearen frogapena B eta P-ko aldagaiak eta konstanteak erabiliz eratutako E *borne-adierazpenak*[3] sortzean datza. E borne-adierazpenak honako bi propietate hauek bete beharko ditu:

a) $(\, INB \wedge B \,) \rightarrow E \in \mathbb{N}$ (hau da, $E \geq 0$)

b) $\{\, INB \wedge B \wedge E = z \,\}$ `P` $\{\, E < z \,\}$.

Hor z izenak INB, B eta P-n agertzen ez den izen bat izan behar du.

Ikusten denez, E espresioak iterazioaren barnean balio bezala zenbaki arruntak hartuko dituela ziurtatzen du a) propietateak, eta, bestalde, E espresioak gero eta txikiagoak diren balioak hartuko dituela bermatzen duen Hoareren hirukotea formalizatzeko z izen berria erabiltzen du b) propietateak. Beraz, metodoa honela justifikatzen da: demagun z_i balioa E adierazpenak i-garren urratsean hartzen duen balioa dela ($E = z_i \in \mathbb{N}$), honako bi propietate hauek beteko dira:

- $z_1, z_2, \ldots, z_i, \ldots \in \mathbb{N}$
- $z_1 > z_2 > \ldots > z_i > \ldots$

Ondorioz, $INB \wedge B$ formula ezin daiteke mugagabeki bete, bestela hertsiki beherakorra den kate infinitu bat sortuko bailitzateke $\mathbb{N}$-ko balioak erabiliz, eta hori ezinezkoa da. Inbariantea iterazio bakoitzean kontserbatu egiten dela frogatu dugunez (infinituki ere bai) eta iterazioren batean $INB \wedge B$ konjuntzioa faltsua izatera iritsiko denez, konjuntzio horren B osagaia izango da noizbait faltsua izatera behartuta egongo dena, $\neg B$ betez.

Horrenbestez, aipatutako a) eta b) propietateak frogatuz gero (E adierazpenak zenbaki arrunten multzoko balioak bakarrik hartzen dituela eta iterazioz iterazio haren balioa hertsiki txikiagoa dela), honako egitura hau duen *while* agindua

```
while  { INB }
  B
loop
    P
end loop;
```

3. Izena iterazio kopuruaren goi-bornea adieraztetik dator.

urrats kopuru finitu baten ondoren bukatu egingo dela frogatuta geldituko da.

Hemendik aurrera programa iteratiboen bukaera-propietatea frogatzeko erabiltzen den borne-adierazpena dokumentazioan ipiniko dugu. Zehazki, borne-adierazpena *while* aginduaren barruan `loop` hitzaren ondoren ipiniko da, honako eskema honetan erakusten den bezala:

```
while  { INB }
 B
loop  { E }
  P
end loop;
```

Bukaera-propietatearen frogapenak errazagoak izan daitezen, Hoareren sistemako *kontserbazioaren axioma* oso erabilgarria gerta daiteke. *Kontserbazioaren axioma* honela definitzen da:

(KA)

$$\frac{\{\, def(P) \wedge \varphi \,\}\ \texttt{P}\ \{\, \varphi \,\}}{\texttt{P} \text{ agindu-segidak } \varphi \text{ formulako aldagairik aldatzen ez badu}}$$

Bukaera-propietatearen frogapen formaletan, axioma horrek zuzentasun partzialerako bakarrik behar diren xehetasunak alde batera uzten ditu, frogapenak errazagoak eginez. Axioma horretan, *def* funtzioa agindu-segidetara orokortu da: $def(P)$ formulak P agindu-segidan agertzen diren termino guztiak definituta egon daitezen bete beharreko propietate denak jasotzen ditu.

Ondoren, 3.9. eta 3.11. ataletan partzialki zuzenak direla frogatu ditugun adibideetara itzuliz, adibide horietako programa iteratiboen bukaera-propietatea frogatuko dugu formalki. Adibide horietan aurre-ondoetako espezifikazioak eta inbarianteak finkatuta genituen, eta orain ere formula horiexek hartuko ditugu. Hasteko, honako programa iteratibo honen bukaera-propietatea frogatuko dugu:

$\{\, \varphi \equiv x = a \wedge y = b \geq 0 \,\}$
```
  while  { INB ≡ x+y = a+b ∧ 0 ≤ y ≤ b }
   y /= 0
  loop
    x := x+1;
    y := y-1;
  end loop;
```
$\{\, \psi \equiv x = a+b \,\}$

Lehenengo urratsa, E borne-adierazpena zehaztea da. Horretarako, B baldintzak eta P agindu-segidak erabiltzen dituzten aldagaiei erreparatuko diegu: hau da, x eta y. Biak dira erabilgarri E borne-adierazpena definitzeko. Hala ere, erraz ikus daiteke x-ren balioa gero eta handiagoa dela, eta y-ren balioa gero eta txikiagoa. Badirudi, hortaz, y-rekin nahikoa dugula borne-adierazpena definitzeko. Oro har, E adierazpenak iterazioa bukatzeko gelditzen den urrats kopurua goitik mugatu behar du, baina kasu honetan E y hartuz, goitik mugatu ez ezik, era zehatzean adieraziko du exekuzioa bukatzeko gelditzen den iterazio kopurua. Horrenbestez:

$$E \equiv y$$

Bigarren urratsa, bukaera bermatzeko aurkeztutako a) eta b) propietateak bete egiten direla frogatzea da:

a) $(\underbrace{x+y=a+b \wedge 0 \leq y \leq b}_{INB} \wedge \underbrace{y \neq 0}_{B}) \rightarrow \underbrace{y}_{E} \in \mathbb{N}$

b) $\{\underbrace{x+y=a+b \wedge 0 \leq y \leq b}_{INB} \wedge \underbrace{y \neq 0}_{B} \wedge \underbrace{y}_{E} = z\}$

```
x := x+1;
y := y-1;
```

$\{\underbrace{y}_{E} < z\}$

Frogapena:

1. $(x+y=a+b \wedge 0 \leq y \leq b \wedge y \neq 0 \wedge y=z) \rightarrow (y-1<z)$
2. $\{y-1<z\}$
   `x := x+1;` **(KA)**
   $\{y-1<z\}$
3. $\{x+y=a+b \wedge 0 \leq y \leq b \wedge y \neq 0 \wedge y=z\}$
   `x := x+1;` 1, 2 eta **(ODE)**
   $\{y-1<z\}$
4. $\{y-1<z\}$
   `y := y-1;` **(EA)**
   $\{y<z\}$
5. $\{x+y=a+b \wedge 0 \leq y \leq b \wedge y \neq 0 \wedge y=z\}$
   `x := x+1;`
   `y := y-1;` 3, 4 eta **(KPE)**
   $\{y<z\}$

Beste iterazio honen bukaera-propietatea ere era berean frogatuko dugu:

$\{\, x \geq 1 \wedge y = 1 \,\}$
```
while  { ber2(y) ∧ 1 ≤ y ≤ x }
 2*y <= x
loop
   y := 2*y;
end loop;
```
$\{\, y = handiena\{k \mid ber2(k) \wedge k \leq x\} \,\}$

Hasteko, E borne-adierazpena definitu behar dugu. Kasu honetan, y aldagaia baino ez da agertzen P agindu-segidan, baina haren balioa gero eta handiagoa da iterazioak aurrera egin ahala. Beraz, y adierazpena ezin da borne-adierazpentzat hartu, bukaerarako gelditzen zaizkigun urratsen kopurua ez baitu mugatzen. Bestetik, $-y$ adierazpenak ere ez du balio, $-y$ adierazpenak gero eta balio txikiagoak hartu arren, balio negatiboak hartuko lituzkeelako, eta hori ez da onargarria. Nahiz eta iterazioaren gorputzean ez agertu, B baldintzan beste aldagai bat ere badugu: x. Iterazioak ez du x-ren balioa aldatzen, baina y aldagaiak hartuko dituen balioen goi-muga bat ezartzen du. Hortaz, honako borne-adierazpena finkatuko genuke:

$$E \equiv x - y$$

Metodoak zehaztutako bi propietateak frogatzea dagokigu orain:

a) $(\, ber2(y) \wedge 1 \leq y \leq x \wedge 2 * y \leq x \,) \rightarrow x - y \in \mathbb{N}$

b) $\{\, ber2(y) \wedge 1 \leq y \leq x \wedge 2 * y \leq x \wedge x - y = z \,\}$
```
   y := 2*y;
```
$\{\, x - y < z \,\}$

Frogapena:

1. $(\, ber2(y) \wedge 1 \leq y \leq x \wedge 2 * y \leq x \wedge x - y = z \,)$
   $\rightarrow (\, 1 \leq y \wedge x - (2 * y) \geq 0 \wedge x - y = z \geq 0 \wedge y < 2 * y \,)$
   $\rightarrow (\, x - (2 * y) < z \,)$
2. $\{\, x - (2 * y) < z \,\}$
   `y := 2*y;` **(EA)**
   $\{\, x - y < z \,\}$
3. $\{\, ber2(y) \wedge 1 \leq y \leq x \wedge 2 * y \leq x \wedge x - y = z \,\}$
   `y := 2*y;` 1, 2 eta **(ODE)**
   $\{\, x - y < z \,\}$

Kasu honetan E borne-adierazpena iterazio kopuruaren goi-muga da, iterazio bakoitzean unitate bat baino gehiago egiten duelako beherantz.

Ondoren, honako baieztapen honetan agertzen den iterazioaren bukaera frogatuko dugu:

$\{\, n \geq 1 \,\}$
```
i := 0
zk := 0
while  { 0 ≤ i ≤ n ∧ zk = #j ( 1 ≤ j ≤ i ∧ A(j) = 0 ) }
 i < n
loop
  i := i+1;
  if A(i) = 0 then
    zk := zk+1;
  end if;
end loop;
```
$\{\, zk = \#j \,(\, 1 \leq j \leq n \,\wedge\, A(j) = 0 \,) \,\}$

Baieztapen horretako programan, i eta zk aldagaiak 0 balioarekin hasieratuko dira eta haien balioa gero eta handiagoa izango da. Hortaz, aldagai horiek ezin dira zuzenean borne-adierazpentzat hartu. Baina iterazioaren baldintzan n konstantea agertzen da, eta haren balioa i eta zk aldagaiek har ditzaketen balioak baino handiagoa da. Ondorioz, E borne-adierazpena definitzeko bi aukera kontsidera ditzakegu: $n-i$ eta $n-zk$. Borne-adierazpenak iterazio bakoitzaren ondoren balio txikiagoa hartu behar duenez, $n-zk$ ez da egokia, bakarrik $A(i)$ zero denean txikitzen baita. Bestalde, $n-i$ adierazpenaren balioa iterazio bakoitzean txikitu egiten da, i handitu egiten baita. Gainera, $n-i$ adierazpenak exekuzioa bukatzeko zehazki zenbat iterazio gelditzen diren adierazten du. Beraz, $E \equiv n-i$. Bektoreen tratamendu sekuentziala egiten denean, ohikoa da era horretako borne-adierazpenak definitzea: bektoreko elementuak tratatu ahala, oraindik tratatu gabeko elementuen kopurua borne-adierazpentzat har daiteke. Hona hemen bukaeraren frogapena, proposatutako borne-adierazpena erabiliz:

a) $\quad (\, 0 \leq i \leq n \,\wedge\, zk = \#j \,(\, 1 \leq j \leq i \,\wedge\, A(j) = 0 \,) \,\wedge\, i < n \,) \,\rightarrow\, n - i \in \mathbb{N}$

b) Frogapena:

$$\begin{aligned} 1.\ (\, 0 \leq i \leq n \,\wedge\, zk = \#j \,(\, 1 \leq j \leq i \,\wedge\, A(j) = 0 \,) \,\wedge\, i < n \,\wedge\, & \\ n - i = z \,) & \\ \rightarrow\ (\, 1 \leq i+1 \leq n \,\wedge\, n - i = z > 0 \,) & \\ \rightarrow\ (\, 1 \leq i+1 \leq n \,\wedge\, n - (i+1) < z \,) & \end{aligned}$$

2. $\{\, 1 \le i+1 \le n \,\wedge\, n-(i+1) < z \,\}$
   `i := i+1;` **(EA)**
   $\{\, 1 \le i \le n \,\wedge\, n-i < z \,\}$
3. $\{\, 0 \le i \le n \,\wedge\, zk = \mathcal{N}j\,(\, 1 \le j \le i \,\wedge\, A(j) = 0\,) \,\wedge\, i < n \,\wedge\, n-i = z \,\}$
   `i := i+1;` 1, 2 eta **(ODE)**
   $\{\, 1 \le i \le n \,\wedge\, n-i < z \,\}$
4. $\{\, 1 \le i \le n \,\wedge\, n-i < z \,\}$
   ```
   if A(i) = 0 then
     zk := zk+1;
   end if;
   ```
   (KA)
   $\{\, n-i < z \,\}$
5. $\{\, 0 \le i \le n \,\wedge\, zk = \mathcal{N}j\,(\, 1 \le j \le i \,\wedge\, A(j) = 0\,) \,\wedge\, i < n \,\wedge\, n-i = z \,\}$
   ```
   i := i+1;
   if A(i) = 0 then
     zk := zk+1;
   end if;
   ```
   3, 4 eta **(KPE)**
   $\{\, n-i < z \,\}$

Frogapen horretako lehenengo urratsean $\alpha \to \beta \to \delta$ erako inplikazio-segida bat daukagu. Inplikazio-segida horretan β tarteko formula hirugarren urratsean erabilitako $\alpha \to \delta$ inplikazio logikoa hobeto ulertzeko ipini da.

Gure azken adibidean, zenbaki bat betea den ala ez erabakitzen duen programa bukatzen dela frogatuko dugu:

$\{\, x \ge 1 \,\}$
```
i := 1
b := 0
while
```
$\{\, 1 \le i \le (x/2)+1 \,\wedge\, b = \sum_{\substack{1 \le k < i \,\wedge \\ x \,\%\, k = 0}} k \,\}$
```
 i <= x/2
loop
  if x % i = 0 then
    b := b+i;
  end if;
  i := i+1;
end loop;
q := (x=b);
```
$\{\, q \leftrightarrow x = \sum_{\substack{1 \le k < x \,\wedge \\ x \,\%\, k = 0}} k \,\}$

Aurreko baieztapenean, i aldagaia 1 balioarekin hasieratu da eta iterazio bakoitzean haren balioa handituz doa $(x/2)+1$ baliora iritsi arte. Iterazio bakoitzean, i eta $(x/2)+1$ adierazpenen arteko aldea gero eta txikiagoa da, eta, gainera, *while* agindua bukatzeko gelditzen den iterazio kopurua zehazten du. Beraz, $(x/2)+1-i$ borne-adierazpentzat har daiteke. Jarraian agindu iteratiboaren bukaera-propietatea frogatuko da $E \equiv (x/2)+1-i$ erabiliz:

a) $$(1 \leq i \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i \wedge \\ x \,\%\, k = 0}} k \wedge i <= x/2) \rightarrow (x/2)+1-i \in \mathbb{N}$$

b) Frogapena:

1. $\{ (x/2)+1-(i+1) < z \} \equiv \{ (x/2)-i < z \}$
   ```
   i := i+1;
   ```
   (EA)
   $\{ (x/2)+1-i < z \}$
2. $\{ i \neq 0 \wedge (x/2)-i < z \}$
   ```
   if x % i = 0 then
     b := b+i;
   end if;
   ```
   (KA)
   $\{ (x/2)-i < z \}$
3. $\{ i \neq 0 \wedge (x/2)-i < z \}$
   ```
   if x % i = 0 then
     b := b+i;
   end if;
   i := i+1;
   ```
   1, 2 eta **(KPE)**
   $\{ (x/2)+1-i < z \}$
4. $$(1 \leq i \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i \wedge \\ x \,\%\, k = 0}} k \wedge i <= x/2 \wedge (x/2)+1-i = z)$$
   $$\rightarrow (i \neq 0 \wedge (x/2)+1-i = z > 0)$$
   $$\rightarrow (i \neq 0 \wedge (x/2)-i < z)$$
5. $$\{ 1 \leq i \leq (x/2)+1 \wedge b = \sum_{\substack{1 \leq k < i \wedge \\ x \,\%\, k = 0}} k \wedge i <= x/2 \wedge (x/2)+1-i = z \}$$
   ```
   if x % i = 0 then
     b := b+i;
   end if;
   i := i+1;
   ```
   3, 4 eta **(ODE)**
   $\{ (x/2)+1-i < z \}$

Aurreko adibide batzuetan gertatu den bezala, laugarren urratsean $\alpha \to \beta \to \delta$ erako inplikazio-segida bat daukagu. Hor β tarteko formula bosgarren urratsean erabilitako $\alpha \to \delta$ inplikazio logikoa errazago ulertzeko ipini da.

3.14. ARIKETAK: BORNE-ADIERAZPENAK ETA ITERAZIOEN BUKAERA

1. Eman borne-adierazpena eta egin bukaeraren frogapen formala 3.10. ataleko ariketa bakoitzean.

3.15. BIBLIOGRAFIA-OHARRAK

1949ko ekainaren 24an, A. Turing-ek *Checking a Large Routine* izeneko hitzaldian [89] (azpi-)programa bat matematikoki zuzena dela frogatzeko helburua iradoki zuen. Aldi berean, helburu hori lortzeko propietate batzuk betetzen direla frogatzearekin nahikoa dela ere iradoki zuen. Hamarkada bat igaro ondoren, konputagailuen programazioaren alorreko ikertzaile batzuek helburu horri berriro ekin zioten. 60ko hamarkadaren erdialdera, *programazio egituratu*aren agerpenarekin batera [31, 32], R. Floyd-en fluxu-diagramei (*flowcharts*) buruzko lanak [40] eta C. A. R. Hoareren *while*-programen lengoaiari buruzko lanak [52] programei buruz arrazoitzeko sistema formalaren kontzeptua sorrarazi zuten. Horrela, programen egiaztapena sortu zen. Programen egiaztapenak programen diseinuan, programazio-lengoaien semantikan eta programazio-lengoaien diseinuan eragin handia izan zuen [54, 66]. Besteak beste, K. Apt-en lanak [4, 5] aitzindariak izan ziren programen egiaztapenaren oinarriak definitzen. 70eko hamarkadan, programen egiaztapena automatizatzeko helburuarekin, asertzioak erabiltzen zituzten programazio-lengoaia esperimental batzuk agertu ziren. Programazio-lengoaia horien artean, Gypsy [3], Euclid [84], eta Alphard [94] aipa daitezke. Eiffel [71] izan zen asertzioak erabili zituen objektuei zuzendutako lehenengo programazio-lengoaia. Haren diseinua E. W. Dijkstra-ren lanetan [34, 33] oinarritzen da. Gaur egun, programen egiaztapen automatikorako tresna asko garatzen ari dira software-industrian gehien erabiltzen diren programazio-lengoaientzat. Adibidez, ezagunenak eta erabilienak direnen artean, hurrengoak aipa daitezke: ACL2 [59], Dafny [62], Key [16], KIV [14] eta Spark [15].

3.16. ARIKETAK: PROGRAMA ITERATIBOEN EGIAZTAPENA

3.16.1. Esleipena eta konposaketa sekuentziala

1. Idatzi hirukotea betearazten duen agindua (____;)[4]:

1.1. $\{\ 1 \le i \le n \ \wedge\ z = \sum_{k=1}^{i-1} A(k)\ \}$

$\_\_\_\_\_;$

$\{\ 1 \le i \le n \ \wedge\ z = \sum_{k=1}^{i} A(k)\ \}$

1.2. $\{\ handiena(A(1..i), m) \ \wedge\ 1 \le i < n \ \wedge\ A(i+1) > m\ \}$

$\_\_\_\_\_;$

$\{\ handiena(A(1..i+1), m) \ \wedge\ 1 \le i < n\ \}$

1.3. $\{\ \forall i\ (\ 1 \le i < muga \ \rightarrow\ A(i) = i^2\ ) \ \wedge\ 1 \le muga \le n\ \}$

$\_\_\_\_\_;$

$\{\ \forall i\ (\ 1 \le i \le muga \ \rightarrow\ A(i) = i^2\ ) \ \wedge\ 1 \le muga \le n\ \}$

2. Markatu zuzena den aukera:

2.1. Zein da baieztapena zuzena egiten duen postbaldintza?

$\{\ a = i\ \}$

```
i := i+1;
```

a) $\{\ a = i+1\ \}$ [ ]

b) $\{\ a = i-1\ \}$ [ ]

c) $\{\ i = i+1 \ \wedge\ a = i\ \}$ [ ]

2.2. Zein da baieztapen zuzena?

i) $\{\ i-1 > 0\ \}$ `i := i-1;` $\{\ i > 0\ \}$ [ ]

ii) $\{\ i > 0\ \}$ `i := i-1;` $\{\ i-1 > 0\ \}$ [ ]

4. $handiena(A(1..n), x) \equiv badago(A(1..n), x) \ \wedge\ \forall j\ (\ 1 \le j \le n \ \rightarrow\ x \ge A(j)\ )$ eta $badago(A(1..n), x) \equiv \exists i\ (\ 1 \le i \le n \ \wedge\ A(i) = x\ )$

3. Bete hutsuneak (____) honako baieztapen honi dagokion frogapenean:

$\{\ z = p^k\ \}$
```
  k := k+1;
  z := z*p;
```
$\{\ z = p^k\ \}$

Frogapena:

1. $(\ z = p^k\ ) \rightarrow (\ z = p^{k+1-1}\ )$
2. $\{\ z = p^{k+1-1}\ \}$
   `k := k+1;` **(EA)**
   $\{\ \_\_\_\_\_\ \}$
3. $\{\ z = p^k\ \}$
   `k := k+1;` 1, 2 eta **(ODE)**
   $\{\ \_\_\_\_\_\ \}$
4. $(\ z = p^{k-1}\ ) \rightarrow (\ z * p = p^{k-1} * p\ ) \rightarrow (\ \_\_\_\_\_\ )$
5. $\{\ \_\_\_\_\_\ \}$
   `z := z*p;` **(EA)**
   $\{\ z = p^k\ \}$
6. $\{\ \_\_\_\_\_\ \}$
   `z := z*p;` 4, 5 eta **(ODE)**
   $\{\ z = p^k\ \}$
7. $\{\ z = p^k\ \}$
   `k := k+1;`
   `z := z*p;` 3, 6 eta **(KPE)**
   $\{\ z = p^k\ \}$

4. Esan honako baieztapen hauek egiazkoak ala faltsuak diren, x edozein aldagai eta t edozein termino izanda. Baieztapen bakoitzeko, zuzena baldin bada, frogatu haren zuzentasuna, eta faltsua baldin bada, bilatu kontraadibide bat.

4.1. $\{\ False\ \}$
```
  x := t;
```
$\{\ \varphi\ \}$

4.2. $\{\ True\ \}$
```
  x := t;
```
$\{\ \varphi\ \}$

5. Egiaztatu era formalean honako baieztapen hauek:

5.1. $\{\, n \geq 1 \wedge A(1) \neq 0 \wedge zerorik\_ez \,\}$

```
  i := 1;
```

$\{\, 1 \leq i \leq n \wedge (\, zerorik\_ez \leftrightarrow \forall k \, (\, 1 \leq k \leq i \, \rightarrow \, A(k) \neq 0 \,) \,) \,\}$

5.2. $\{\, x \geq y \,\}$

```
  z := x;
```

$\{\, z = handiena(x, y) \,\}$

5.3. $\{\, 1 \leq i \leq n \wedge s = \sum_{k=1}^{i-1} A(k) \,\}$

```
  s := s+A(i);
  i := i+1;
```

$\{\, 1 \leq i \leq n+1 \wedge s = \sum_{k=1}^{i-1} A(k) \,\}$

6. Frogatu, kontraadibide bat emanez, honako baieztapen hau ez dela zuzena:

$\{\, y * z^k = p \,\}$

```
  k := k/2;
  z := z*z;
```

$\{\, y * z^k = p \,\}$

7. Honako baieztapen-bikote bakoitzean zuzena denarentzat, eman dagokion zuzentasun-froga, eta, zuzena ez denarentzat, eman kontraadibide bat zuzena ez dela erakusteko:

7.1. (A) $\{\, 1 \leq v \leq z \wedge x^z * y^v = w \,\}$

```
  z := z+1;
  w := w*x*y;
  v := v+1;
```

$\{\, 1 < v \leq z \wedge x^z * y^v = w \,\}$

(B) $\{\, 1 < v \leq z \wedge x^z * y^v = w \,\}$

```
  z := z-1;
  w := w*x*y;
  v := v-1;
```

$\{\, 1 \leq v \leq z \wedge x^z * y^v = w \,\}$

(A) baieztapena zuzena da eta (B) ez da zuzena [ ]

(B) baieztapena zuzena da eta (A) ez da zuzena [ ]

7.2. (A) $\{\, 1 < k < w \wedge z = 2^k * 4^w \,\}$

```
k := k-1;
z := z*8;
w := w+2;
```

$\{\, 1 \leq k < w \wedge z = 2^k * 4^w \,\}$

(B) $\{\, 1 \leq k < w \wedge z = 2^k * 4^w \,\}$

```
k := k+2;
z := z*8;
w := w+1;
```

$\{\, 1 < k \leq w \wedge z = 2^k * 4^w \,\}$

(A) baieztapena zuzena da eta (B) ez da zuzena [ ]

(B) baieztapena zuzena da eta (A) ez da zuzena [ ]

8. Frogatu honako baieztapen hauek, s_i espresioak Fibonacci-ren segidako i-garren terminoa adierazten duela kontuan hartuz: $s_0 = 0$, $s_1 = 1$ eta $k \geq 2$ denean $s_k = s_{k-1} + s_{k-2}$.

8.1. $\{\, 1 \leq i < n \wedge x = s_i \wedge y = s_{i+1} \wedge z = s_{i+2} \,\}$

```
x := y;
y := z;
z := x+y;
i := i+1;
```

$\{\, 1 \leq i \leq n \wedge x = s_i \wedge y = s_{i+1} \wedge z = s_{i+2} \,\}$

8.2. $\{\, \exists i \,(\, i \geq 0 \wedge u = s_i \wedge z = s_{i+1} \,) \,\}$

```
u := u+z;
z := u+z;
```

$\{\, \exists i \,(\, i \geq 0 \wedge u = s_i \wedge z = s_{i+1} \,) \,\}$

9. n-garren karratu betea, hau da n^2, lehenengo n zenbaki bakoitien batura da. Beste era batean esanda:

$$n^2 = \sum_{k=1}^{n} (2 * k - 1)$$

Propietate hori erabiliz, frogatu honako baieztapen hau:

$\{\, 1 \leq k < n \wedge x = k^2 \,\}$

```
k := k+1;
x := x + 2*k - 1;
```

$\{\, 1 \leq k \leq n \wedge x = k^2 \,\}$

3.16.2. Baldintzazko aginduak

1. Osatu honako baieztapen hau postbaldintza ({ __ }) egokia emanez:

$\{\, True \,\}$
```
if x > y then
  z := x;
else
  z := y;
end if;
```
{ _____ }

2. Idatzi honako hirukoteak betearazten dituzten aginduak (____):

2.1. $\{\, i = n \wedge \neg badago(A(1..n-1), x) \,\}$
```
if ___________ then
  _____________________;
else
  _____________________;
end if;
```
$\{\, dago \leftrightarrow badago(A(1..n), x) \,\}$

2.2. $\{\, 1 \leq i < n \wedge handiena(A(1..i), m) \,\}$
```
i := i+1;
if ___________ then
  _____________________;
end if;
```
$\{\, 1 \leq i \leq n \wedge handiena(A(1..i), m) \,\}$

3. Bete hutsuneak (____) honako baieztapen honi dagokion frogapenean:

$\{\, 1 \leq err < n \wedge 1 \leq zut \leq n \wedge (err-1) * n + zut = b \,\}$
```
if zut = n then
  err := err+1;
  zut := 1;
else
  zut := zut+1;
end if;
```
$\{\, 1 \leq err, zut \leq n \wedge (err-1) * n + zut = b+1 \,\}$

Frogapena:

1. $(1 \leq err < n \wedge 1 \leq zut \leq n \wedge (err-1)*n+zut = b \wedge$
   $zut = n)$
   $\rightarrow (1 \leq err < n \wedge (err-1)*n+n = b)$
   $\rightarrow (1 \leq err < n \wedge err*n = b)$
   $\rightarrow (1 \leq err+1-1 < n \wedge (err+1-1)*n = b)$
2. $\{$ _____ $\}$
   `err := err+1;` **(EA)**
   $\{ 1 \leq err-1 < n \wedge (err-1)*n = b \}$
3. $(1 \leq err-1 < n \wedge (err-1)*n = b)$
   $\rightarrow (1 \leq err \leq n \wedge (err-1)*n = b)$
4. $\{ 1 \leq err < n \wedge 1 \leq zut \leq n \wedge (err-1)*n+zut = b \wedge$
   $zut = n \}$
   `err := err+1;` 1, 2, 3 eta **(ODE)**
   $\{ 1 \leq err \leq n \wedge (err-1)*n = b \}$
5. $\{ 1 \leq err \leq n \wedge (err-1)*n = b \}$
   `zut := 1;` **(EA)**
   $\{$ _____ $\}$
6. $(1 \leq err \leq n \wedge (err-1)*n = b \wedge zut = 1)$
   $\rightarrow (1 \leq err \leq n \wedge (err-1)*n+1 = b+1 \wedge zut = 1)$
   $\rightarrow (1 \leq err, zut \leq n \wedge (err-1)*n+zut = b+1)$
7. $\{ 1 \leq err \leq n \wedge (err-1)*n = b \}$
   `zut := 1;` 5, 6 eta **(ODE)**
   $\{ 1 \leq err, zut \leq n \wedge (err-1)*n+zut = b+1 \}$
8. $\{ 1 \leq err < n \wedge 1 \leq zut \leq n \wedge (err-1)*n+zut = b \wedge$
   $zut = n \}$
   `err := err+1;`
   `zut := 1;` _____
   $\{ 1 \leq err, zut \leq n \wedge (err-1)*n+zut = b+1 \}$
9. $($ _____ $)$
   $\rightarrow (1 \leq err < n \wedge 1 \leq zut < n \wedge$
   $(err-1)*n+zut = b)$
   $\rightarrow (1 \leq err < n \wedge 1 \leq zut+1-1 < n \wedge$
   $(err-1)*n+zut+1-1 = b)$
10. $\{$ _____ $\}$
    `zut := zut+1;` **(EA)**
    $\{ 1 \leq err < n \wedge 1 \leq zut-1 < n \wedge (err-1)*n+zut-1 = b \}$

11. $(\ 1 \leq err < n \ \wedge\ 1 \leq zut - 1 < n \ \wedge\ (err-1) * n + zut - 1 = b\ )$
$\rightarrow\ (\ 1 \leq err < n \ \wedge\ 1 \leq zut \leq n \ \wedge$
$(err-1) * n + zut = b + 1\ )$
$\rightarrow\ (\ 1 \leq err, zut \leq n \ \wedge\ (err-1) * n + zut = b + 1\ )$

12. $\{\ 1 \leq err < n \ \wedge\ 1 \leq zut \leq n \ \wedge\ (err-1) * n + zut = b \ \wedge$
$zut \neq n\ \}$
`zut := zut+1;` 9, 10, 11 eta **(ODE)**
$\{\ 1 \leq err, zut \leq n \ \wedge\ (err-1) * n + zut = b + 1\ \}$

13. $(\ 1 \leq err < n \ \wedge\ 1 \leq zut \leq n \ \wedge\ (err-1) * n + zut = b\ )$
$\rightarrow\ def(zut = n)$

14. $\{\ 1 \leq err < n \ \wedge\ 1 \leq zut \leq n \ \wedge\ (err-1) * n + zut = b\ \}$

```
if zut = n then
  err := err+1;
  zut := 1;
else
  zut := zut+1;
end if;
```

8, 12, 13 eta **(BDE)**

$\{\ 1 \leq err, zut \leq n \ \wedge\ (err-1) * n + zut = b + 1\ \}$

4. B adierazpen boolearra, x edozein aldagai, t edozein termino eta φ edozein formula izanda, esan honako baieztapen hauek egiazkoak ala faltsuak diren. Baieztapen bakoitzeko, egiazkoa baldin bada, frogatu, eta, bestela, bilatu kontraadibide bat.

4.1. $\{\ def(B) \ \wedge\ \neg B\ \}$

```
if B then
  x:=t;
end if;
```

$\{\ \varphi\ \}$

4.2. $\{\ def(B) \ \wedge\ \neg B \ \wedge\ \varphi\ \}$

```
if B then
  x:=t;
end if;
```

$\{\ \varphi\ \}$

5. Egiaztatu era formalean honako baieztapen hauek:

5.1. $\{\, True \,\}$

```
if i <= j then
  if j < k then
    m := k;
  else
    m := j;
  end if;
else
  if i < k then
    m := k;
  else
    m := i;
  end if;
end if;
```

$\{\, m = handiena(i, j, k) \,\}$

5.2. $\{\, True \,\}$

```
if (x % 2) /= 0 then
  x := x+1;
end if;
```

$\{\, bikoitia(x) \,\}$

5.3. $\{\, \neg berdin \,\}$

```
if x = y then
  berdin := true;
end if;
```

$\{\, berdin \leftrightarrow x = y \,\}$

6. Izan bedi honako Hoareren hirukote hau:

$\{\, True \,\}$
```
if x = 0 then
  y := 3;
end if;
```
$\{\, \psi \,\}$

Aukeratu honako $\{\, \psi \,\}$ formula hauetatik hirukotea egiazkoa egiten duena:

$\{\, x = 0 \wedge y = 3 \,\}$ [ ]
$\{\, x = 0 \leftrightarrow y = 3 \,\}$ [ ]
$\{\, x = 0 \rightarrow y = 3 \,\}$ [ ]
$\{\, y = 3 \rightarrow x = 0 \,\}$ [ ]

7. Zuzenak al dira honako baieztapen hauek? Kasu bakoitzean frogatu zuzentasuna edo eman kontraadibidea.

7.1. $\{\, True \,\}$
```
if x < y then
  lag := x;
  x := y;
  y := lag;
elsif y < z then
  lag := y;
  y := z;
  z := lag;
else
  lag := x;
  x := z;
  z := lag;
end if;
```
$\{\, x \geq y \geq z \,\}$

7.2. $\{\, True \,\}$

```
if x < y then
  lag := x;
  x := y;
  y := lag;
end if;
if y < z then
  lag := y;
  y := z;
  z := lag;
end if;
if x < y then
  lag := x;
  x := y;
  y := lag;
end if;
```

$\{\, x \geq y \geq z \,\}$

8. Honako bi baieztapen hauetatik zuzena denarentzat, eman dagokion zuzentasun-froga, eta, zuzena ez denarentzat, eman kontraadibide bat zuzena ez dela erakusteko:

(A) $\{\, z = x + y \,\}$

```
y := y/2;
if y % 2 /= 0 then
  x := x+1;
end if;
x := x+y;
```

$\{\, z = x + y \,\}$

(B) $\{\, z = x + y \,\}$

```
if y % 2 /= 0 then
  x := x+1;
end if;
y := y/2;
x := x+y;
```

$\{\, z = x + y \,\}$

(A) zuzena da eta (B) ez da zuzena [ ]

(B) zuzena da eta (A) ez da zuzena [ ]

9. Asmatu zuzentasun partziala frogatzeko egokia den inferentzi erregela honako agindu hauentzat:

   9.1. B_i adierazpen boolearrak eta P_i programak izanda:

```
if B1 then
  P1;
elsif B2 then
  P2;
...
elsif Bn then
  Pn;
else
  Pn+1;
end if;
```

   9.2. Kasu-hautaketa duten aginduen aldaera bat da honako hau. Hor E datu-mota diskretu bateko espresioa da, b_i balioak datu-mota horretakoak dira (gainera beraien artean desberdinak dira) eta P_i elementuak programak dira.

```
case E is
  when b1 => P1;
    ...
  when bn => Pn;
  when others => Pn+1;
end case;
```

   Aginduaren funtzionamendua honako hau da: P_i exekutatuko da E espresioaren balioa b_i denean. E ebaluatzean lortzen den balioa $b_1, \ldots, b_n$ balioen artean ez badago, P_{n+1} exekutatuko da.

3.16.3. Iterazioak

1. Aukeratu inbariante zuzena aurre-ondoetako espezifikazioarekin datozen honako programa hauentzat:

   1.1. Programa honek *lehena* aldagai boolearrean x zenbaki arrunta lehena den ala ez erabakiko du.

$\{\, x \geq 2 \,\}$
```
d := 2;
while  { INB }
 x % d /= 0
loop
  d := d+1;
end loop;
lehena := (d = x);
```
$\{\, lehena \leftrightarrow \forall i \,(\, 1 < i < x \;\rightarrow\; x \,\%\, i \neq 0 \,) \,\}$

   (a) **INB** $\equiv\; 1 < d \leq x \;\wedge\; \forall i \,(\, 1 < i < d \;\rightarrow\; x \,\%\, i \neq 0 \,)$ [ ]

   (b) **INB** $\equiv\; 1 < d \leq x \;\leftrightarrow\; \forall i \,(\, 1 < i < d \;\rightarrow\; x \,\%\, i \neq 0 \,)$ [ ]

   (c) **INB** $\equiv\; 1 < d \leq x \;\wedge\; (\, lehena \leftrightarrow \forall i \,(\, 1 < i < d \;\rightarrow\; x \,\%\, i \neq 0 \,) \,)$ [ ]

   1.2. Programa honek x zenbaki arruntaren faktoriala kalkulatuko du.

$\{\, x \geq 0 \,\}$
```
f := 1;
t := x;
while  { INB }
 t >= 1
loop
  f := f*t;
  t := t-1;
end loop;
```
$\{\, f = x! \,\}$

   (a) **INB** $\equiv\; f = (t-1)! \;\wedge\; 0 \leq t \leq x$ [ ]

   (b) **INB** $\equiv\; f = x!/t! \;\wedge\; 0 \leq t \leq x$ [ ]

   (c) **INB** $\equiv\; f = x!/(t-1)! \;\wedge\; 0 \leq t \leq x$ [ ]

2. Eman honako iterazio hauen inbariantea eta borne-adierazpena:

   2.1. Honako programa honek $A(1..n)$ bektoreko balio txikiena kalkulatuko du m aldagaian.

```
{ n ≥ 1 }
  m := A(1);
  k := 1;
  while  { INB ≡ _____________________________ }
   k < n
  loop   { E ≡ _____________ }
    k := k+1;
    if A(k) < m then
      m := A(k);
    end if;
  end loop;
{ txikiena(A(1..n),m) }
```

   non:

$$txikiena(A(1..n),m) \equiv \exists i\,(\,1 \le i \le n \wedge A(i) = m\,) \wedge \forall j\,(\,1 \le j \le n \rightarrow A(j) \ge m\,)$$

   2.2. Honako programa honek $A(1..n)$ bektoreko elementuetatik erdia baino gehiago $B(1..n)$ bektoreko posizio berean daudenak baino handiagoak diren ala ez erabakiko du, erantzuna b aldagai boolearrean utziz.

```
{ n ≥ 1 }
  i := 1;
  z := 0;
  while  { INB ≡ _____________________________ }
   i <= n
  loop   { E ≡ _____________ }
    if A(i) > B(i) then
      z := z+1;
    end if;
    i := i+1;
  end loop;
  b := (z > n/2);
{ b ↔ 𝒩j ( 1 ≤ j ≤ n ∧ A(j) > B(j) ) > n/2 }
```

2.3. Honako programa honek $A(1..n)$ bektoreko elementuak atzekoz aurrera ipiniko ditu.

$\{\, n \geq 1 \wedge A = (a_1, \ldots, a_n) \,\}$
```
  k := 1;
  while  { INB ≡ _____ }
   k <= n/2
  loop   { E ≡ _____ }
    lag := A(k);
    A(k) := A(n-k+1);
    A(n-k+1) := lag;
    k := k+1;
  end loop;
```
$\{\, \forall i \,(\, 1 \leq i \leq n \;\rightarrow\; A(i) = a_{n-i+1} \,) \,\}$

3. Dokumentatu honako programa iteratibo hauek programa bakoitzean zehaztutako asertzioekin:

3.1. Honako programa honek $|x - y|$ balioa d aldagaian utziko du.

$\{$ **Aurre** $\equiv$ _____ $\}$
```
  d := 0;
  if x <= y then
    u := x;
    z := y;
  else
    u := y;
    z := x;
  end if;
```
$\{\, \varphi_1 \equiv$ _____ $\}$
```
  while  { INB ≡ _____ }
   u /= z
  loop   { E ≡ _____ }
```
$\{\, \varphi_2 \equiv$ _____ $\}$
```
    z := z-1;
```
$\{\, \varphi_3 \equiv$ _____ $\}$
```
    d := d+1;
  end loop;
```
$\{$ **Post** $\equiv d = |x - y| \,\}$

3.2. Honako programa honek $A(1..n)$ bektorea batura berdineko bi sekziotan zati daitekeen ala ez erabakiko du. Hala bada, i aldagaia izango da zatiketa non egin daitekeen adierazten duen indizea.

```
{ Aurre ≡ _____ }
  x := A(1);
  y := 0;
  k:= 2;
  while  { INB1 ≡ _____ }
   k <= n
  loop   { E1 ≡ _____ }
    y := y+A(k);
    k := k+1;
  end loop;
  i := 1;
  while  { INB2 ≡ _____ }
   x /= y and i < n
  loop   { E2 ≡ _____ }
    i := i+1;
    x := x+A(i);
    y := y-A(i);
  end loop;
  batuber := not (i = n);
{ Post ≡ _____ }
```

4. Frogatu honako programa hauen zuzentasun osoa.

4.1. Honako bi programa hauek $A(1..n)$ bektore ez-hutsean x osagaia zenbat aldiz agertzen den kalkulatuko dute.

i.
```
i := 1;
z := 0;
while i <= n loop
  if A(i) = x then
    z := z+1;
  end if;
  i := i+1;
end loop;
```

ii.
```
i := 0;
z := 0;
while i < n loop
  i := i+1;
  if A(i) = x then
    z := z+1;
  end if;
end loop;
```

4.2. Honako programa honek hutsa ez den $A(1..n)$ bektorean 5 zenbakiaren anizkoitzak diren balioen kopurua kalkulatuko du.

```
i := 0;
zenbat := 0;
while i < n loop
  i := i+1;
  if A(i) % 5 = 0 then
    zenbat := zenbat+1;
  end if;
end loop;
```

4.3. Honako programa honek $A(1..n)$ bektore ez-hutseko balio handiena itzuliko du.

```
i := n;
m := A(n);
while i > 1 loop
  i := i-1;
  if A(i) > m then
    m := A(i)
  end if;
end loop;
```

4.4. Honako programa honek x elementua $A(1..n)$ bektore ez-hutsean agertzen den ala ez erabakiko du.

$\{\, n \geq 1 \,\}$

```
  i := 0;
  dago := false;
  while not dago and i < n loop
    i := i+1;
    if A(i) = x then
      dago := true;
    end if;
  end loop;
```

$\{\, dago \leftrightarrow \exists j \,(\, 1 \leq j \leq n \wedge A(j) = x \,) \,\}$

4.5. Honako programa honek, n balioa ez-negatiboa izanda, Fibonacciren segidako s_n terminoaren balioa itzuliko du x aldagaian ($s_0 = 0$, $s_1 = 1$ eta $k \geq 2$ denean $s_k = s_{k-1} + s_{k-2}$).

```
x := 0;
y := 1;
k := 1;
while k <= n loop
  y := y+x;
  x := y-x;
  k := k+1;
end loop;
```

4.6. Programa honek $A(1..n)$ eta $B(1..n)$ bektore ez-hutsen $A(1..j)$ eta $B(1..j)$ azpibektore ez-hutsek batura berdina izatea eragiten duen j indize handiena k aldagaian utziko du.

```
m := 1;
k := 1;
bat := A(1);
bider := B(1);
while m < n loop
  m := m+1;
  bat := bat+A(m);
  bider := bider*B(m);
  if bat = bider then
    k := m;
  end if;
end loop;
```

5. Osatu hutsuneak (____) honako programa iteratibo honetan. Programak $A(1..n)$ bektore ez-hutsean dauden balio bikoitien kopurua balio bakoitien kopuruaren berdina den ala ez erabakiko du.

$\{\, n \geq 1 \,\}$
```
  i := 0;
  w := 0;
  z := 0;
  while
```
$\{\, \_\_ \leq i \leq \_\_ \wedge \; \_\_ = \mathcal{N}j\,(\,1 \leq j \leq \_\_ \wedge A(j)\,\%\,2 = 0\,) \wedge \; \_\_ = \mathcal{N}j\,(\,1 \leq j \leq \_\_ \wedge A(j)\,\%\,2 \neq 0\,) \,\}$
```
   i < n
  loop
```
$\{\, \mathbf{E} \equiv \_\_\_\_\_ \,\}$
```
    i := i+1;
```
$\{\, \_\_ \leq i \leq \_\_ \wedge \; \_\_ = \mathcal{N}j\,(\,1 \leq j < \_\_ \wedge A(j)\,\%\,2 = 0\,) \wedge \; \_\_ = \mathcal{N}j\,(\,1 \leq j < \_\_ \wedge A(j)\,\%\,2 \neq 0\,) \,\}$
```
    if ( A(i) % 2 = 0 ) then
```
$\{\, \_\_ \leq i \leq \_\_ \wedge \; \_\_ = \mathcal{N}j\,(\,1 \leq j \leq \_\_ \wedge A(j)\,\%\,2 = 0\,) \wedge \; \_\_ = \mathcal{N}j\,(\,1 \leq j \leq \_\_ \wedge A(j)\,\%\,2 \neq 0\,) \,\}$
```
      w := w+1;
    else
```
$\{\, \_\_ \leq i \leq \_\_ \wedge \; \_\_ = \mathcal{N}j\,(\,1 \leq j \leq i \wedge A(j)\,\%\,2 = 0\,) \wedge \; \_\_ = \mathcal{N}j\,(\,1 \leq j \leq i \wedge A(j)\,\%\,2 \neq 0\,) \,\}$
```
      z := z+1;
    end if;
  end loop;
```
$\{\, w = \mathcal{N}j\,(\,1 \leq j \leq \_\_ \wedge A(j)\,\%\,2 = 0\,) \wedge \; v = \mathcal{N}j\,(\,1 \leq j \leq \_\_ \wedge A(j)\,\%\,2 \neq 0\,) \,\}$
```
  r := (w=z);
```
$\{\, r \leftrightarrow (\,\mathcal{N}j\,(\,1 \leq j \leq \_\_ \wedge A(j)\,\%\,2 = 0\,) = \mathcal{N}j\,(\,1 \leq j \leq \_\_ \wedge A(j)\,\%\,2 \neq 0\,)\,) \,\}$

6. Eman zuzentasun partziala frogatzeko balio duen inferentzi erregela eta bukaera frogatzeko balio duten propietateak ADA programazio-lengoaiakoak diren honako kontrol-egitura bakoitzarentzat.

6.1.
$\{ \varphi \}$
```
loop
  P1;
  exit when B;
end loop;
```
$\{ \psi \}$

6.2.
$\{ \varphi \}$
```
loop
  P1;
  exit when B;
  P2;
end loop;
```
$\{ \psi \}$

4. Programa errekurtsiboen egiaztapena

Problema bat ebazteko era algoritmikoa, txikiagoak diren problema beraren kasuetarako lortzen diren soluzioak erabiliz formalizatzea ahalbidetzen duen programazio-teknika da errekurtsioa. Aldiz, problema baten soluzioa lortzeko teknika bezala agindu multzo bat errepikatzea da iterazioa. Errekurtsioa, algoritmoak eta datu-motak definitzeko teknika bezala ikusita, egiturazko indukzioaren printzipioarekin estuki lotuta dago. Indukzio matematikoa (zenbaki arrunten gaineko indukzioa) printzipio horren kasu partikularra da. Objektu multzo infinituen propietate batzuk (besteak beste, zenbaki arrunten propietate batzuk) frogatzeaz gain, egiturazko indukzioaren printzipioa erabiliz objektu multzo infinituak, multzo horiek eremutzat dituzten funtzioak eta funtzio horiek konputatzen dituzten algoritmoak defini daitezke eta, oro har, indukzio bidez definitutako objektuen gaineko eragiketak buru daitezke.

4.1. atalean, egiturazko indukzioaren oinarri matematikoak azalduko dira. Horretarako, bai funtzioen bai objektu multzoen (adibidez, *kateen*) definizio errekurtsiboaren nozioa aurkeztuko da. Gainera, objektu multzo horien gainean beste funtzio batzuk definitu ahal izango dira errekurtsiboki. 4.2. atalean, programa errekurtsiboen egitura eta funtzionamendua aztertuko da, beharrezkoa den notazioa aurkeztuz. Hurrengo atalean (4.3. atala), programa errekurtsiboen azterketa lantzeko hainbat ariketa proposatu dira. 4.4. atalean, programa errekurtsiboen zuzentasuna egiaztatzeko metodoa aurkeztuko da. 4.5. atalean, errekurtsioaren kontzeptuari buruzko bibliografia-ohar aipagarri batzuk bilduko dira. Bukatzeko, 4.6. atalean, programa errekurtsiboen egiaztapena lantzeko ariketa-zerrenda bat osatu da.

4.1. DEFINIZIO ERREKURTSIBOAK

Definizio bat errekurtsiboa izango da, definitzen duen kontzeptua (funtzioa, multzoa eta abar) definizioan bertan erabiltzen bada. Lehenengo adibide erraz bezala zenbaki arrunten faktoriala erabiliko dugu. Gainera, adibide horrek errekurtsioa eta iterazioa alderatzeko aukera emango digu. Era iteratiboan, edo eragiketa bera errepikatuz, n zenbaki arruntaren faktoriala 1 eta

n-ren artean dauden zenbaki oso guztien biderketa da. $n!$ bezala adierazi ohi da n-ren faktoriala, eta dagokion definizio iteratiboa honako hau da:

$$n! = 1 * 2 * 3 * 4 * 5 * \ldots * (n-1) * n$$

0 baino handiagoa eta 1 baino txikiagoa den zenbaki osorik ez dagoenez, 0ren faktoriala 1 da hitzarmenez (biderketaren elementu neutroa). Bestalde, faktorialaren definizio errekurtsiboa emateko, zenbaki baten faktoriala bere aurreko zenbakiaren faktorialean oinarrituz kalkula daitekeela kontuan hartzen da:

$$n! = \begin{cases} 1 & \text{baldin } n = 0 \\ n * (n-1)! & \text{baldin } n > 0 \end{cases}$$

Definizio errekurtsibo horretan, faktorialaren kontzeptua erreferentziatzen ez duen oinarrizko kasu bat edo kasu nabari bat dago: lehenengo kasua, hau da, 0ren faktoriala 1 dela adierazten duen kasua. Baina kasu induktibo edo errekurtsibo bat ere badago: bigarren kasua, n edozein zenbaki positibo izanda, n-ren faktoriala lortzeko $n-1$ zenbakiaren faktoriala (errekurtsiboki) kalkulatu eta n balioaz biderkatu behar dela adierazten duen kasua. Bi definizio horiek konparatzeko, esate baterako 3! balioa lortzeko beharrezkoak diren kalkuluak nola egiten diren aztertuko dugu. Alde batetik, definizio iteratiboa erabiliz honako hau izango genuke:

$$3! = 1 * 2 * 3$$

Beste aldetik, definizio errekurtsiboa behin eta berriz gero eta txikiagoak diren zenbakiei aplikatuko zaie, oinarrizko kasura iritsitakoan bukatuz (hau da, $0! = 1$):

$$3! = 3 * (2!) = 3 * (2 * (1!)) = 3 * (2 * (1 * (0!))) = 3 * (2 * (1 * 1))$$

Definizio errekurtsiboek (egituratutako) objektu multzo infinituak definitzeko eskaintzen duten aukera nabarmendu beharreko ezaugarria da. M multzo bat errekurtsiboki definitzeko honako elementu hauek zehaztu beharko dira:

- Gutxienez kasu nabari bat (hau da, oinarrizko kasu bat): M multzokoak diren elementuz osatutako hasierako multzo bat definitzeko.
- Gutxienez erregela induktibo/errekurtsibo bat: dagoeneko M multzoan dauden elementuetan oinarrituz, M-ri elementu berriak nola erantsi arautzeko.

- Baztertze-erregela: kasu nabarien bidez zehaztutako elementuak eta erregela induktiboen bidez errekurtsiboki lortutako elementuak baino ez dira egongo M multzoan (gehienetan, baztertze-erregela inplizitutzat hartzen da).

Adibidez, Σ alfabetoa emanda, Σ-ren gainean osa daitezkeen kate (*string*) guztiez osatutako Σ^* multzoa honela definituko litzateke errekurtsiboki:

- Oinarrizko kasua: ε kate hutsa Σ^* multzoan dago (hau da, $\varepsilon \in \Sigma^*$).
- Erregela induktiboa: w katea Σ^* multzoan baldin badago eta $x \in \Sigma$ betetzen bada, orduan $wx \in \Sigma^*$.

Indukzio bidez definitutako multzoen gaineko funtzioak (edo eragiketak) definitzeko era errazena, laburrena eta argiena errekurtsioa erabiliz izan ohi da. Adibidez, kate baten osagai kopurua kalkulatzen duen *luzera* funtzioa Σ^*-ren gainean errekurtsiboki honela defini daiteke:

- Kasu nabaria: $luzera(\varepsilon) = 0$.
- Erregela induktiboa: $luzera(wx) = luzera(w) + 1$.

Era berean, bi kateren kateaketa (_@_) honela defini daiteke:

- Kasu nabaria: $w@\varepsilon = w$, baldin eta $w \in \Sigma^*$.
- Erregela induktiboa: $w_1@(w_2x) = (w_1@w_2)x$, baldin eta $w_1 \in \Sigma^*$, $w_2 \in \Sigma^*$ eta $x \in \Sigma$.

Oro har, definizio errekurtsiboek hainbat kasu nabari eta hainbat erregela induktibo izan ditzakete. Adibidez, bi zenbaki arrunten zatitzaile komunetako handienaren honako definizio errekurtsibo honek bi kasu nabari eta bi kasu induktibo ditu:

$$zkh(a,b) = \begin{cases} a & \text{baldin } a = b \\ a & \text{baldin } b = 0 \\ zkh(b,a) & \text{baldin } a < b \\ zkh(a-b,b) & \text{baldin } a > b \end{cases}$$

4.2. PROGRAMA ERREKURTSIBOAK

Kalkuluak errepikatzearen beharra duten problemak ebazteko iterazioaren ordez erabil daitekeen teknika bat da errekurtsioa. Aurreko atalean aurkeztutako faktorialaren definizio ez-errekurtsiboa honela inplementa daiteke era iteratiboan:

```
function faktoriala(n:  Integer) return f:  Integer is
  kont:  Integer;
{ n ≥ 0 }
begin
  kont := 1;
  f := 1;
  while  { INB ≡ (f = (kont − 1)! ∧ 1 ≤ kont ≤ n + 1) }
   kont <= n
  loop
    f := f*kont;
    kont := kont+1;
  end loop;
end faktoriala;
{ f = n! }
```

Problema askoren soluzioa (beraien artean faktorialaren kalkulua) era errekurtsiboan plantea daiteke. Problemari parametro formalekiko dagokion soluzioaren deskribapena problema berari nolabait sinpleagoak diren parametro batzuekiko dagokion soluzioan oinarrituz egingo dela esan nahi du horrek. Askotan, programa errekurtsiboak iteratiboak baino errazago diseinatu eta egiaztatzen dira. Hori horrela da, bereziki, problema (nolabait) sinpleagoak diren azpiproblematan deskonposa badaiteke. Era horretako adibide oso ezagun bat atal honetan aurrerago aztertuko dugun *Hanoiko Dorre*en problema da. Halaber, zuhaitzak edo kateak bezala errekurtsiboki definitutako datuegiturekin aritu behar duten problemak ebazteko, soluzio errazena, sinpleena eta argiena errekurtsiboa izan ohi da. Programa errekurtsiboek beren buruari deitzen diote. Programa errekurtsibo bati egindako dei-segida finitua izan dadin (hau da, programa buka dadin), dei bakoitzak aurrekoak baino gertuago egon beharko du soluziotik, baita deia egin duena baino gertuago ere. Problemaren soluzioarekiko distantzia edo tartea sarrerako parametroen balioetan oinarrituz zehaz daiteke: dei errekurtsibo bat zenbat eta soluziotik gertuago, orduan eta txikiagoak izango dira parametro bezala hartuko dituen balioak. Beraz, uneren batean, deia era ez-errekurtsiboan ebatzi beharko da, hau da, dei errekurtsibo gehiago sortu gabe. Ondorioz, programa errekurtsiboek *kasuak* bezala izendatuko ditugun bi motatako baldintzazko adarrak erabiliko dituzte:

- *Kasu nabariak* edo *oinarrizko kasuak* (ez-errekurtsiboak).
- *Kasu errekurtsiboak* edo *induktiboak* (errekurtsioa dutenak).

Mota bakoitzetik gutxienez kasu bat izan behar dute programa errekurtsibo guztiek. Hala ere, oro har, mota bakoitzetik hainbat kasu izaten dituzte.

Kasu nabariek parametro edo argumentu bezala errekurtsioaren beharrik ez duten balioekin egindako deiak ebatziko dituzte. Aldiz, kasu errekurtsiboetan problemaren ebazpena dei errekurtsiboen bidez lortutako soluzio partzialetan (hau da, azpiproblemen soluzioetan) oinarritzen da eta bukaerako soluzioa osatzeko soluzio partzial horiek erabiltzen dira. Ondorioz, programa errekurtsiboek izena eta sarrerako parametroak izan beharko dituzte azpiproblemak ebatzi beharko dituzten dei errekurtsiboak deskribatu ahal izateko. Gainera, teknikoki hobeto datorkigulako, emaitzari (edo funtzioak itzultzen duen balioari) ere izena emango diogu. Zehazki, programa errekurtsibo baten *goiburua* adierazteko honako notazio hau erabiliko dugu:

```
function f(x1: T1; ...; xn: Tn) return z: R is
```

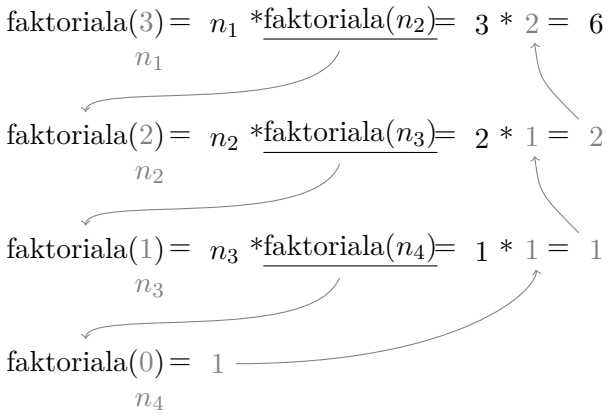
Hor, `f` funtzioaren izena da, x_i eta T_i bakoitza i-garren sarrerako parametroaren izena eta mota dira (hurrenez hurren), eta z eta R programak lortzen duen emaitzaren izena eta mota dira (hurrenez hurren). Berriro ere teknikoki horrela hobeto datorkigulako, programa errekurtsiboei egindako deiak dei horren emaitza jasoko duen aldagai bati egindako esleipen baten eskuineko aldea osatuko dutela ezarriko dugu hitzarmen bezala. Hau da, lehen erakutsi den goiburua duen `f` funtzioari egindako edozein dei errekurtsibok honako egitura hau izango du:

```
e := f(p1,...,pn)
```

Esleipen horretan, $p_1, \ldots, p_n$ dei errekurtsiboaren parametro errealak dira ($x_1, \ldots, x_n$ parametro formalen lekukoak, beraz), eta emaitza (z emaitza formalaren lekukoa den) e aldagaian utziko da. Funtzioaren *gorputza* edo *bloke nagusia*, hasieran `begin` agindua eta bukaeran `end` agindua ipiniz mugatuko da, eta `end` aginduan funtzioaren goiburuan zehaztutako izena erabiliko da:

```
begin
  // Agindu-segida
end f;
```

Gure notazioa ADA lengoaiaren sintaxiaren aldean pixka bat desberdina da, ADA lengoaian emaitzari (edo itzuliko den balioari) izenik ezin baitzaio eman. Gainera, ADA programazio-lengoaiako edozein funtziok, itzuliko den balioa zein den zehaztuko duen `return` aginduarekin bukatu beharko du bere exekuzio-fluxu bakoitza. Liburu honetan, arrazoi teknikoak direla-eta, `return` agindua alde batera utziko dugu, eta agindu hori erabili ordez, funtzioaren goiburuan definitutako emaitzaren izenari esleipen bat egingo diogu. Horrela, zuzentasunaren eta balidazioaren frogapenetan emaitza (formala edo, deien ondoren, erreala) aipatu ahal izango dugu.



4.1. irudia. *faktoriala*(3)**-ren kalkulu errekurtsiboa.**

Funtzioaren goiburuaren eta `begin` aginduaren artean idatziko dugun aurrebaldintzaz eta funtzioaren gorputzaren bukaerako `end` aginduaren ondoren idatziko dugun postbaldintzaz osatuta egongo da aurre-ondoetako espezifikazioa. Aurrebaldintzak $x_1, \ldots, x_n$ parametro formalak izango ditu aldagai libre bezala eta postbaldintzak parametro horien eta z emaitza formalaren arteko erlazioa ezarriko du (z ere libre agertuko da postbaldintzan).

Atal honetako gainerakoan, hiru programa errekurtsibo esanguratsu aztertuko ditugu.

4.2.1. Zenbaki baten faktoriala

Honako programa errekurtsibo hau lehenago emandako faktorialaren definizio errekurtsiboaren inplementazioa da:

```
function faktoriala(n:  Integer) return f:  Integer is
{ n ≥ 0 }
begin
  if n = 0 then
    f := 1;
  else
    f := faktoriala(n-1);
    f := n*f;
  end if;
end faktoriala;
{ f = n! }
```

Programa horretan, $n=0$ baldintza duen eta gorputz bezala `f := 1;` agindua duen oinarrizko kasu bakarra eta $n \neq 0$ baldintza ($n=0$ baldintzaren ukapena) eta gorputz bezala `f := faktoriala(n-1); f := n*f;` agindu-segida duen kasu induktibo bakarra definitzen dira. Kasu induktiboan egiten den dei errekurtsibo bakarra `f := faktoriala(n-1);` agindua da.

4.1. irudian $faktoriala(3)$ deiaren ebaluazioa erakusten da, dei errekurtsiboak nola egiten diren deskribatuz. $faktoriala(3)$ deia egiten denean, n aldagaiaren *bertsio* berri bat sortuko da. Bertsio berri hori n_1 izenarekin adieraziko da eta haren balioa 3 izango da. n_1 aldagaiaren balioa 0 ez denez, $faktoriala(3)$ deiak itzuliko duen balioa *else*-ren gorputzari dagokiona izango da, hau da, $n_1 * faktoriala(n_1 - 1)$. Biderketa hori ebazteko $faktoriala(n_1 - 1)$ kalkulatu behar denez, berriro $faktoriala$ funtzioari deituko zaio, oraingoan 2 balioarekin: $faktoriala(2)$. Dei horretan, n aldagaiaren beste kopia bat sortuko da, n_2 izenarekin eta 2 balioarekin. Aurreko kasuan bezala, n_2-ren balioa ez da 0 eta $faktoriala$ funtzioaren dei horrek $n_2 * faktoriala(n_2 - 1)$ itzuliko du. Era berean, biderketa hori kalkulatu ahal izateko, $faktoriala$ funtzioari hirugarren aldiz deituko zaio, sarrerako datu bezala 1 balioa emanez: $faktoriala(1)$. Dei horren ondorio bezala, n-ren bertsio berri bat sortuko da, 1 balioa izango duen n_3 izeneko bertsioa kasu honetan, eta dei errekurtsiboak $n_3 * faktoriala(n_3 - 1)$ itzuliko du emaitza gisa. Berriro ere $faktoriala$ funtzioari deituko zaio sarrerako balio bezala 0rekin, eta ondorioz, n_4 bezala adieraziko den eta 0 balioa izango duen n-ren beste bertsio bat sortuko da. Kasu horretan, deiak *then* adarraren gorputzari dagokion emaitza itzuliko du. Hau da, itzuliko den balioa 1 izango da. $faktoriala(0)$ deia ebatzi ondoren, biderketak egin beharko dira deiak egitean jarraitutako ordenaren aurkako ordenari jarraituz, $faktoriala(3)$ hasierako deiaren balioa lortu arte.

4.2.2. Bi kateren osagaiak tartekatu

Karaktere-kateak erabiltzen dituzten programen adibide bezala, *zip* eragiketaren honako definizio errekurtsiboari dagokion programa aztertuko dugu. Eragiketa horrek bi kateren karaktereak kremailera baten hortzak tartekatzen diren bezala tartekatuz lortzen den katea eraikiko du:

$$zip : \Sigma^* \times \Sigma^* \to \Sigma^*$$

$$zip(w_1, w_2) = \begin{cases} w_1 & \text{baldin } w_2 = \varepsilon \\ w_2 & \text{baldin } w_1 = \varepsilon \\ zip(y_1, y_2)x_1x_2 & \text{baldin } w_1 = y_1x_1 \text{ eta } w_2 = y_2x_2 \end{cases}$$

Hurrengo programa errekurtsiboak *zip* eragiketa inplementatzen du aurreko definizio errekurtsibo horri jarraituz. Inplementazio horretan, kateen gainean

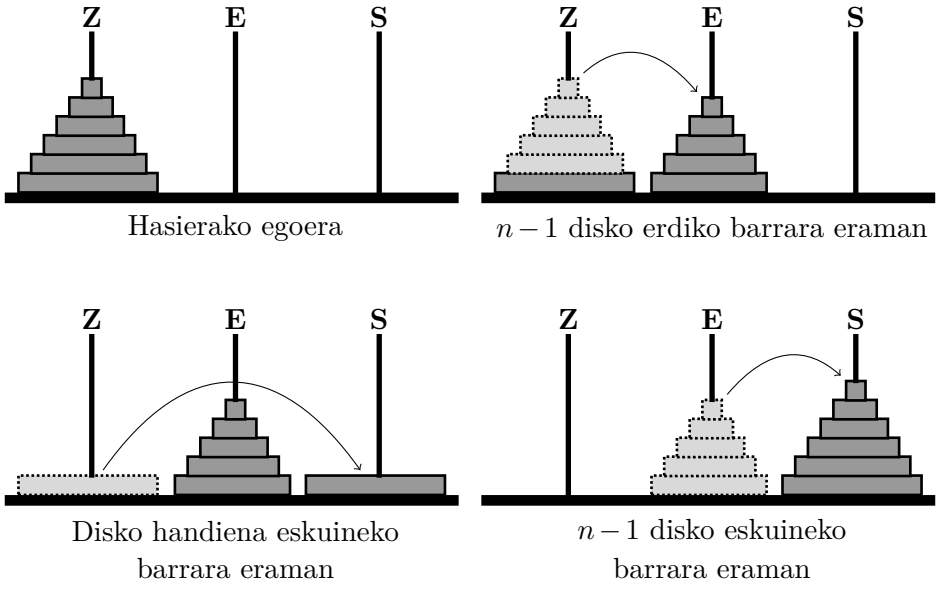
definitutako *luzera*, *azkena* eta *hasiera* funtzioak erabiltzen dira. *luzera* funtzioa 4.1. atalean definitu da dagoeneko. Kate ez-hutsa emanda, *azkena* eta *hasiera* funtzioek kate horren azken osagaia eta azken osagaia ezabatuz gelditzen den katea kalkulatzen dituzte hurrenez hurren:

```
function zip(w1, w2: Σ*) return z: Σ* is
  x1, x2: Σ;
  y1, y2: Σ*;
begin
  if luzera(w1) = 0 then
    z := w2;
  elsif luzera(w2) = 0 then
    z := w1;
  else
    x1 := azkena(w1);
    x2 := azkena(w2);
    y1 := hasiera(w1);
    y2 := hasiera(w2);
    z := zip(y1,y2);
    z := (z x1) x2;
  end if;
end zip;
```

Honako adibide honek *zip* programaren funtzionamendua erakusten du:

$$\begin{aligned} \texttt{zip}(luoe, argi) &= \texttt{zip}(luo, arg)\ ei \\ &= \texttt{zip}(lu, ar)\ ogei \\ &= \texttt{zip}(l, a)\ urogei \\ &= \texttt{zip}(\varepsilon, \varepsilon)\ laurogei \\ &= laurogei \end{aligned}$$

Adibide horretan, ez ditugu ($faktoriala(3)$-ri dagokion aurreko adibidean bezala) dei bakoitzeko $w1$ eta $w2$ parametroen bertsio berriak izendatu. $\texttt{zip}(luoe, argi)$ hasierako deiak z emaitzan utzi beharreko balioa $\texttt{zip}(luo, arg)$ deiak z emaitzan utziko duen balioari ei erantsiz kalkulatzen da. Era berean, $\texttt{zip}(luo, arg)$ deiari dagokion z-ren balioa, $\texttt{zip}(lu, ar)$ deiari dagokion z-ren balioari og erantsiz kalkulatzen da. Eta $\texttt{zip}(lu, ar)$ deiari dagokion z-ren balioa $\texttt{zip}(l, a)$ deiari dagokion z-ren balioari ur erantsiz kalkulatzen da. Azkenik, $\texttt{zip}(l, a)$ deiari dagokion z-ren balioa $\texttt{zip}(\varepsilon, \varepsilon)$ deiari dagokion z-ren balioari la erantsiz lortzen da. $\texttt{zip}(\varepsilon, \varepsilon)$ deiari dagokion z-ren balioa kate hutsa (ε) denez, atzeraka itzultzea nahikoa da aurreko deiari dagokion



4.2. irudia. *Hanoiko Dorreak* **problemaren soluzioa** n **diskorentzat.**

z-ren balioa *la* izateko, aurreko dei horren aurreko deiari dagokion z-ren balioa *laur* izateko, lehenagoko deiari dagokion z-ren balioa *laurog* izateko eta hasierako deiari dagokion z-ren balioa *laurogei* izateko.

4.2.3. Hanoiko Dorreak

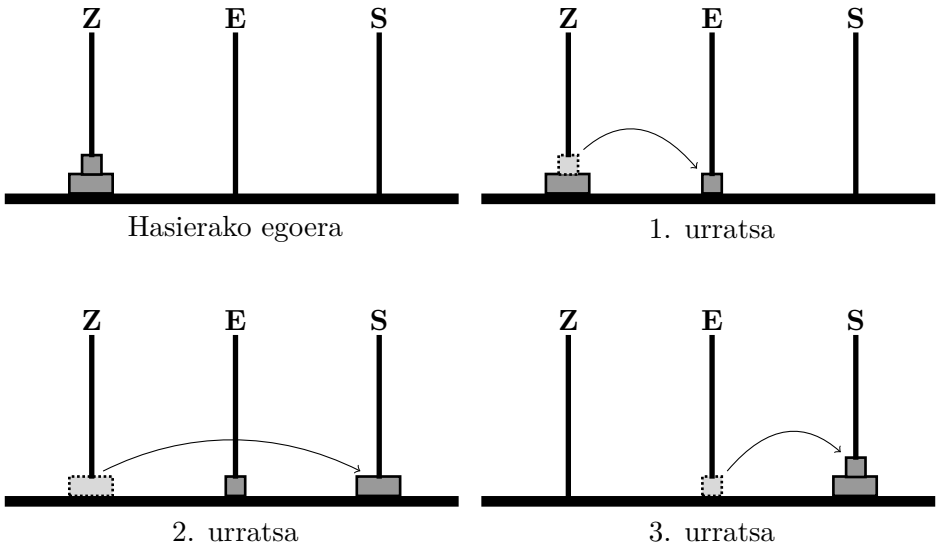
Programa errekurtsiboen diseinuaren arloan oso ezaguna den problema bat *Hanoiko Dorreak* izenekoa da. Problema honetan erradio desberdina duten n disko eta hiru barra bertikal daude. Diskoak barratan sar daitezke piramide egitura duten dorreak edo pilak osatuz (erradio txikiagokoak gorago ipiniz). Hasierako egoeran, n diskoak lehenengo barra bertikalean egongo dira (ikus 4.2. irudia). Helburua, lehenengo barra bertikaleko n diskoak azkeneko barrara eramatea da (erdiko barra bertikala laguntzaile bezala erabil daiteke), baina beti honako arau hauek betez:

1. Urrats bakoitzean disko bakar bat mugi daiteke.

2. Disko bat ezin da ipini erradio txikiagoko beste disko baten gainean.

Arazo hau oso erraz ebatz daiteke azpiproblemaren kontzeptua erabiltzen bada, programa errekurtsibo bat lortuz. Aldiz, egin beharreko diskoen

mugimenduak era iteratiboan adieraztea ez da hain erraza. Hasteko, azter dezagun kasu errazena. Disko kopurua 1 baldin bada (hau da, $n = 1$), soluzioa huskeria da, oso erraza da: disko bakarra lehenengo barra bertikaletik azken barra bertikalera eramango da. Hau, era nabarian, errekurtsiorik behar ez duen oinarrizko kasu bat da.



4.3. irudia. *Hanoiko Dorreak* **problemaren soluzioa 2 diskorentzat.**

Problemaren soluzioa 2 diskorentzat 4.3. irudian deskribatzen da. Soluzio horretan, erdiko barra laguntzaile bezala erabiltzen da diskoak mugitzeko. Hiru diskorentzat (hau da, $n = 3$ denean), problemaren soluzioa ez da huskeria beharrezkoak diren mugimendu egoki guztiez osatutako segida zein izango den pentsatzen hasten bagara. Hala ere, soluzioa askoz errazagoa den eran formula daiteke (3 disko mugitzearen azpiproblema den) 2 diskorentzako problemaren soluzioan oinarritzen bagara. Eta, oro har, n diskorentzako problemarentzat estrategia berari jarrai dakioke, $n-1$ diskorentzako azpiproblemaren soluzioa erabiliz.

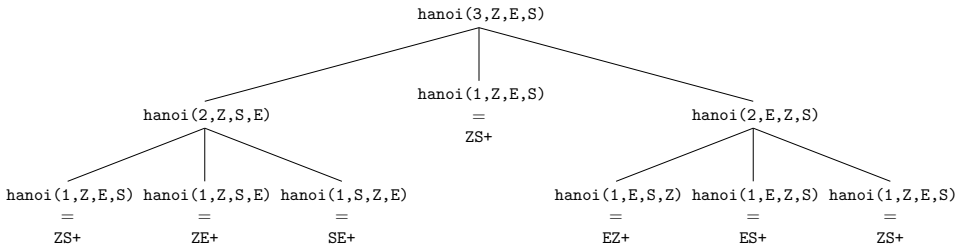
Jarraian erakusten den programa errekurtsiboak *Hanoiko Dorreak* problema ebazteko beharrezkoak diren mugimenduez osatutako katea lortzeko balio du n diskorentzat eta hurrenez hurren ezkerrean, erdian eta eskuinean kokatuta dauden hiru barrarentzat (`Z`, `E` eta `S`). Barra batetik beste barra batera egindako mugimendu bakoitza karaktere bikote baten bidez adieraziko da emaitza gisa lortuko den h katean. Karaktere bikote horietako bakoitzean,

lehenengo karaktereak abiapuntuko barra eta bigarren karaktereak helmugako barra adieraziko dituzte. Bestalde, karaktere bikote horiek + karaktereaz bananduta egongo dira. Karaktereek katean duten ordenak diskoen mugimenduen ordena era egokian adieraziko du. Kateak elkartzeko, 4.1. atalean definitutako @ eragiketa erabiltzen du programak:

```
function hanoi(n: Integer; z,e,s: Barra)
return h: Barra* is
  u,v,w: Barra*;
begin
  if n = 1 then
    h := εzs+;
  else
    u := hanoi(n-1,z,s,e);
    v := hanoi(1,z,e,s);
    w := hanoi(n-1,e,z,s);
    h := u @ v @ w;
  end if;
end hanoi;
```

Oinarrizko kasua $n = 1$ da. $hanoi(1, z, e, s)$ deiak disko bakar hori z barratik s barrara eramateko behar diren mugimenduen segida h aldagaian utzi behar du. Kasu horretan, lortu nahi den emaitza kalkulatzen dela egiaztatzea huskeria da. Kasu induktiboa $n > 1$ da eta 4.2. irudian deskribatutako estrategiari jarraituz ebazten da. Estrategia hori honako azpiproblema hauek ebaztean datza: $n - 1$ disko mugitzea eskatzen duten eta barrak ordena desberdinean erabiltzen dituzten bi azpiproblema eta azpiena dagoen diskoa mugitzea eskatzen duen azpiproblema. Azken azpiproblema hori kasu nabaria erabiliz ebazten da beti eta, horregatik, `v := hanoi(1,z,e,s);` dei errekurtsiboa egin beharrean, soluzioa zuzenean adieraz daiteke. Azkenik, h emaitza dei errekurtsibo bakoitzak osatutako mugimendu-segidak ordena egokian kateatuz lortuko da.

4.2. irudiak grafikoki adierazten duenez, azpiproblemek ($n - 1$ diskorentzat eta disko batentzat) programak berak bere parametro formalentzat bezala jokatzen dutela suposatuz, programak problema ebazteko beharrezkoak diren mugimendu egokiak egingo ditu n diskorentzat. Azpiproblemen soluzioak «egokiak» direla suposatzea ezinbestekoa da problema osoaren soluzioa diseinatzeko. Beste era batera esanda, indukzio-hipotesiaren bidez dei errekurtsibo bakoitzak eta programak problema bera ebazten dutela bermatzen da, baina dei errekurtsiboen kasuan parametro formalentzat izan beharrean parametro errealentzat. Ondorioz, hasierako problemari (hau da, n diskorentzat eta z, e eta s barrentzat) dagokion h emaitza formala u, v eta w



4.4. irudia. `hanoi` **funtzioak 3 diskorekin eta** `Z`**,** `E` **eta** `S` **barrekin sortzen duen dei-zuhaitza.**

aldagaietan gordetako emaitza partzialak kateatuz osatu beharko da. Adibidez, `hanoi(3,Z,E,S)` deiak 4.4. irudian deskribatutako dei-zuhaitza sortuko luke eta emaitza gisa honako kate hau itzuliko luke:

`ZS+ZE+SE+ZS+EZ+ES+ZS+`

Erraz ikus daiteke $n \geq 1$ diskorentzat sortzen den dei-zuhaitzean hostoak $hanoi(1,\_,\_,\_)$ erako deiak direla, eta, ondorioz, errekurtsioa ondo oinarrituta dagoela, beti kasu nabarian bukatzen baita. Beraz, mugimendu egokien segida eratuz bukatzen da beti programa.

4.3. ARIKETAK: PROGRAMA ERREKURTSIBOEN AZTERKETA

1. Aztertu honako programa hau. Zuzena al da? Zein da arazoa?

```
function faktoriala(n:  Integer) return f:  Integer is
{ n ≥ 0 }
begin
  if n = 0 then
    f := 1;
  else
    f := faktoriala(n-2);
    f := n*(n-1)*f;
  end if;
end faktoriala;
{ f = n! }
```

2. Honako programa honek 2ren n-garren berretura kalkulatzen du e aldagaian.

```
function berretura2(n:  Integer) return e:  Integer is
{ n ≥ 0 }
begin
  if n = 0 then
    e := 1;
  else
    e := 2* berretura2(n-1);
  end if;
end berretura2;
{ e = 2^n }
```

   Deskribatu 2^3 kalkulatzeko jarraituko den prozesua eta marraztu deien zuhaitza.

3. Zehaztu φ formulan x parametroak zein balio har ditzakeen honako programa errekurtsibo honek espezifikazioak adierazitako emaitza itzul dezan.

```
function f(x,n:  Integer) return e:  Integer is
{ n ≥ 1 ∧ φ }
begin
  if n = 1 then
    e := 1+x;
  else
    e := f(x,n-1);
    e := e + x**n;
  end if;
end f;
{ e = (1 - x^(n+1)) / (1 - x) }
```

4.4. PROGRAMA ERREKURTSIBOEN EGIAZTAPENA

Atal honetan programa errekurtsibo baten zuzentasun osoa, hau da, programaren zuzentasun partziala eta bukaera, formalki frogatzeko zer egin behar den deskribatuko dugu. Programa errekurtsiboen zuzentasun partzialaren frogapenean *indukzio-hipotesia* da ardatza. Parametro eta emaitza errealekiko dei-errekurtsiboen jokabidea eta parametro formalekiko programaren jokabidea guztiz berdinak direla onartzean datza indukzio-hipotesia.

Edozein `f` funtzio errekurtsibo emanda

```
function f(x1: T1; ...; xn: Tn) return z: R is
{ φ }
begin
  // Gorputza
end f;
{ ψ }
```

(φ,ψ) espezifikazioarekiko f-ren zuzentasun osoa frogatzeko hurrengo baieztapena betetzen dela frogatu behar da:

$$\{\,\varphi\,\}\ [\ \texttt{z := f(x}_1\texttt{,...,x}_n\texttt{);}\ ]\ \{\,\psi\,\}$$

Lehenik, zuzentasun partzialari dagokionez, funtzioaren gorputza egiaztatu beharko da. Horretarako, Hoareren sistema formala (3. kapituluan bezala) eta indukzio-hipotesia erabiliko ditugu. Indukzio-hipotesia f funtzio berari egindako dei errekurtsiboak dituzten esleipenetan aplikatu beharko da. Baldintzaren erregelaren formulazio orokorrena erabiliz, baldintzazko aginduaren adar bakoitzeko hirukote bat frogatu beharko da. Alde batetik, kasu nabari bakoitzak espezifikazioa betetzen duela frogatuko da. Hau da, honako baieztapen hau frogatu behar da Hoareren kalkulua erabiliz:

$$\{\,\varphi \wedge B_N\,\}\ \mathtt{P_N}\ \{\,\psi\,\}$$

Hor, B_N eta $\mathtt{P_N}$ kasu nabariaren baldintza eta gorputza dira hurrenez hurren. Bestetik, kasu induktibo bakoitzeko honako baieztapen hau frogatu behar da:

$$\{\,\varphi \wedge B_I\,\}\ \mathtt{P_I}\ \{\,\psi\,\}$$

Hor, B_I eta $\mathtt{P_I}$ kasu induktiboaren baldintza eta gorputza dira hurrenez hurren. Kasu induktiboa denez, gutxienez dei errekurtsibo bat agertuko da $\mathtt{P_I}$ gorputzean. Horregatik, aurreko kapituluan aurkeztutako inferentzi erregelez gain, indukzio-hipotesia ere erabili beharko da frogapenean. Aurretik aipatu den bezala, indukzio-hipotesiak dei errekurtsiboen jokabidea programaren espezifikazioarekiko zuzena dela edo bat datorrela onartzea ahalbidetzen digu. Beraz, programaren espezifikazioan parametro formalak dauden lekuan dei errekurtsiboko parametro eta emaitza errealak ipiniz gelditzen dena beteko du dei errekurtsiboak. Ondorioz, honako baieztapen hau frogatzen ari bagara

$$\{\,\varphi\,\}\ [\ \texttt{z := f(x}_1\texttt{,...,x}_n\texttt{);}\ ]\ \{\,\psi\,\}$$

eta honako egitura hau duen dei errekurtsiboa egiten bada

$$\texttt{w := f(t}_\texttt{1}\texttt{,...,t}_\texttt{n}\texttt{)}$$

orduan honako indukzio-hipotesi hau erabili beharko da:

$$\{\ \varphi^{t_1,\ldots,t_n}_{x_1,\ldots,x_n}\ \}\ \texttt{w := f(t}_\texttt{1}\texttt{,...,t}_\texttt{n}\texttt{);}\ \{\ \psi^{w,t_1,\ldots,t_n}_{z,x_1,\ldots,x_n}\ \}$$

Hor, $\varphi^{t_1,\ldots,t_n}_{x_1,\ldots,x_n}$ adierazpena φ formulan $x_1,\ldots,x_n$ aldagaiak $t_1,\ldots,t_n$ terminoekin ordezkatuz lortzen den formula da, eta $\psi^{w,t_1,\ldots,t_n}_{y,x_1,\ldots,x_n}$ adierazpena ψ formulan $y,x_1,\ldots,x_n$ aldagaiak $w,t_1,\ldots,t_n$ terminoekin ordezkatuz lortzen den formula da. Aldibereko ordezkapen hori 2.4. atalean azalduta dago. Hitzarmenez, (ikus 4.2. atala), dei errekurtsibo bakoitzak esleipen baten eskuineko aldea osatuko du. Beraz, indukzio-hipotesia erabiltzea 3. kapituluko esleipenaren axiomaren aldaera bat erabiltzearen baliokidetzat jo dezakegu, dei errekurtsiboekin erabiltzen den aldaera bat, hain zuzen ere.

Bigarrenik, frogatu behar da aurrebaldintza betetzen duten sarrerako datu guztientzat programa errekurtsiboa bukatu egiten dela. Bukaeraren frogapenari *indukzioaren balidazioa* deitzen zaio eta, iterazioekin gertatzen den bezala, *E borne-adierazpen* batean oinarritzen da. Kasu honetan, borne-adierazpena funtzio errekurtsiboaren $x_1,\ldots,x_n$ sarrerako parametroak erabiliz definitzen da. Alde batetik, frogatu beharko da kasu nabari bakoitzean E borne-adierazpena zenbaki arrunta dela ($E \in \mathbb{N}$). Hau da, honako hau betetzen dela:

$$(\ \varphi \wedge B_N\ ) \rightarrow E \in \mathbb{N}$$

Hor, B_N kasu nabariari dagokion baldintza da. Beste aldetik, frogatu behar da dei errekurtsibo bakoitzarekin E borne-adierazpena zenbaki arrunten multzoaren barnean txikiagotuz doala. Hau da, `w := f(t`$_1$`,...,t`$_n$`)` erako dei errekurtsibo bakoitzeko honako hau frogatu behar da:

$$(\ \varphi \wedge B_I\ ) \rightarrow (\ E^{t_1,\ldots,t_n}_{x_1,\ldots,x_n} \in \mathbb{N} \wedge E > E^{t_1,\ldots,t_n}_{x_1,\ldots,x_n}\ )$$

Hor, B_I dei errekurtsiboari dagokion kasu induktiboaren baldintza da eta $E^{t_1,\ldots,t_n}_{x_1,\ldots,x_n}$ adierazpena E borne-adierazpenean $x_1,\ldots,x_n$ aldagaiak $t_1,\ldots,t_n$ terminoekin ordezkatuz lortzen da. Ohartzekoa da, sarrerako parametrorik ez duten funtzioentzat propietate hori ezingo dela frogatu, nahitaez borne-adierazpenaren balioa konstantea izango baita. Horregatik, gutxienez sarrerako parametro bat duten funtzio errekurtsiboak baino ez ditugu kontuan hartuko. Horretaz gain, egiaztatuko ditugun funtzioek inolako albo-ondoriorik ez dutela suposatuko dugu eta, beraz, sarrerako parametroak ez dituztela aldatzen suposatuko dugu. Horrela izan ezean, ezingo da ziurtatu lortutako Hoareren hirukotearen bidez adierazitako propietatearen egiazkotasuna.

Orain, zenbaki baten faktoriala itzultzen duen funtzioaren zuzentasun osoa frogatuko dugu. Horretarako, honako baieztapen hau egiazkoa dela frogatu behar dugu:

$$\{\underbrace{n \geq 0}_{\varphi}\}\ [\ \texttt{f := faktoriala(n);}\ ]\ \{\underbrace{f = n!}_{\psi}\}$$

Frogapena:

- Kasu nabaria ($n = 0$). Honako baieztapen hau frogatu behar da:

$$\{\underbrace{n \geq 0}_{\varphi} \wedge \underbrace{n = 0}_{B_N}\}\ \underbrace{\texttt{f := 1;}}_{\texttt{P}_\texttt{N}}\ \{\underbrace{f = n!}_{\psi}\}$$

1. $(\ n \geq 0 \wedge n = 0\ ) \rightarrow (\ n = 0\ ) \rightarrow (\ 1 = n!\ )$
2. $\{\ 1 = n!\ \}$
   `f := 1;` **(EA)**
   $\{\ f = n!\ \}$
3. $\{\ n \geq 0 \wedge n = 0\ \}$
   `f := 1;` 1, 2 eta **(ODE)**
   $\{\ f = n!\ \}$

- Kasu induktiboa ($n \neq 0$). Honako baieztapen hau frogatu behar da:

$$\{\underbrace{n \geq 0}_{\varphi} \wedge \underbrace{n \neq 0}_{B_I}\}\ \underbrace{\texttt{f := faktoriala(n-1); f := n*f;}}_{\texttt{P}_\texttt{I}}\ \{\underbrace{f = n!}_{\psi}\}$$

Horretarako, honako indukzio-hipotesi hau erabiliko da:

(IH) $\{\ n-1 \geq 0\ \}$ `f := faktoriala(n-1);` $\{\ f = (n-1)!\ \}$

Frogapena:

4. $(\ n \geq 0 \wedge n \neq 0\ ) \rightarrow (\ n > 0\ ) \rightarrow (\ n-1 \geq 0\ )$
5. $\{\ n-1 \geq 0\ \}$
   `f := faktoriala(n-1);` **(IH)**
   $\{\ f = (n-1)!\ \}$
6. $\{\ n \geq 0 \wedge n \neq 0\ \}$
   `f := faktoriala(n-1);` 4, 5 eta **(ODE)**
   $\{\ f = (n-1)!\ \}$

7. $(\ f = (n-1)!\ ) \rightarrow (\ n*f = n*(n-1)!\ ) \rightarrow (\ n*f = n!\ )$
8. $\{\ n*f = n!\ \}$
   `f := n*f;` **(EA)**
   $\{\ f = n!\ \}$
9. $\{\ f = (n-1)!\ \}$
   `f := n*f;` 7, 8 eta **(ODE)**
   $\{\ f = n!\ \}$
10. $\{\ n \geq 0 \wedge n \neq 0\ \}$
    `f := faktoriala(n-1);`
    `f := n*f;` 6, 9 eta **(KPE)**
    $\{\ f = n!\ \}$

- Indukzioaren balidazioa:
  - Borne-adierazpena: $E \equiv n$
  - Kasu nabaria ($n = 0$):

$$(\ \underbrace{n \geq 0}_{\varphi} \wedge \underbrace{n = 0}_{B_N}\ ) \rightarrow (\ n = 0\ ) \rightarrow n \in \mathbb{N}$$

  - Kasu induktiboa ($n \neq 0$):

$$(\underbrace{n \geq 0}_{\varphi} \wedge \underbrace{n \neq 0}_{B_I}\ ) \rightarrow (\ n > 0\ ) \rightarrow (\ n-1 \in \mathbb{N} \wedge n > n-1\ )$$

Indukzio-hipotesiaz gain, aurreko frogapenean 3. kapituluan aurkeztutako Hoareren kalkuluko axioma eta erregelak erabili ditugu. Zuzentasun partzialaren frogapena bukatzeko, baldintzaren erregela (**BDE**) erabili behar da kasu biak (formalki) elkartu ahal izateko.

4.4.1. Errekurtsio anizkoitza eta konjuntzioaren erregela

Programa batek kasu induktibo berean dei errekurtsibo bat baino gehiago egiten badu, *errekurtsio anizkoitza* duela esango dugu. Esate baterako, *hanoi* programak errekurtsio anizkoitza erabiltzen du. Era horretako programak egiaztatzeko, beharrezkoa da Hoareren kalkuluari inferentzi erregela berri bat gehitzea. Jarraian, adibide bat erabiliz erakutsiko dugu behar horren zergatia.

Fibonacci-ren segida zenbaki arruntez osatutako segida infinitu bat da. Segida horretako n-garren terminoa errekurtsibitate anizkoitza duen honako

funtzio honen bidez kalkulatzen da:

$$fibonacci(n) = \begin{cases} n & \text{baldin } 0 \leq n \leq 1 \\ fibonacci(n-1) + fibonacci(n-2) & \text{baldin } n \geq 2 \end{cases}$$

Kapitulu honetan finkatu dugun notazioa erabiliz, Fibonacciren segidako n-garren zenbakia kalkulatzen duen honako programa hau inplementa dezakegu:

```
function fib(n:  Integer) return f:  Integer is
  x, y:  Integer;
{ n ≥ 0 }
begin
  if n <= 1 then
    f := n;
  else
    x := fib(n-1);
    y := fib(n-2);
    f := x+y;
  end if;
end fib;
{ f = fibonacci(n) }
```

fib funtzioak benetan Fibonacciren segidako n-garren zenbakia itzultzen duela egiaztatzeko, honako baieztapen hau frogatu behar da:

$$\{\, n \geq 0 \,\}\ [\ \texttt{f := fib(n);}\ ]\ \{\, f = fibonacci(n) \,\}$$

Zuzentasun partziala frogatzerakoan, besteak beste, funtzioaren kasu induktibo bakarrari buruzko honako baieztapen hau betetzen dela frogatu behar da:

$$\{\, \underbrace{n \geq 0}_{\varphi} \wedge \underbrace{n > 1}_{B_I} \,\}\ \underbrace{\texttt{x := fib(n-1); y := fib(n-2); f := x+y;}}_{\mathtt{P_I}}\ \{\, \underbrace{f = fibonacci(n)}_{\psi} \,\}$$

Horretarako, honako indukzio-hipotesi hauek erabiliko ditugu:

(IH1) $\{\, n-1 \geq 0 \,\}$ `x := fib(n-1);` $\{\, x = fibonacci(n-1) \,\}$
(IH2) $\{\, n-2 \geq 0 \,\}$ `y := fib(n-2);` $\{\, y = fibonacci(n-2) \,\}$

Baieztapen horren zuzentasun partzialari dagokion frogapena ohiko eran hasiko da:

1. $(\ n \geq 0 \wedge n > 1\ ) \rightarrow (\ n-1 \geq 0\ )$
2. $\{\ n-1 \geq 0\ \}$
   `x := fib(n-1);` **(IH1)**
   $\{\ x = fibonacci(n-1)\ \}$
3. $\{\ n \geq 0 \wedge n > 1\ \}$
   `x := fib(n-1);` 1, 2 eta **(ODE)**
   $\{\ x = fibonacci(n-1)\ \}$
4. $(\ n \geq 0 \wedge n > 1\ ) \rightarrow (\ n-2 \geq 0\ )$
5. $\{\ \underbrace{n-1 \geq 0}_{def(x:=fib(n-1))} \wedge n-2 \geq 0\ \} \equiv \{\ n-2 \geq 0\ \}$
   `x := fib(n-1);` **(KA)**
   $\{\ n-2 \geq 0\ \}$
6. $\{\ n \geq 0 \wedge n > 1\ \}$
   `x := fib(n-1);` 4, 5 eta **(ODE)**
   $\{\ n-2 \geq 0\ \}$
7. $(\ n-2 \geq 0 \wedge x = fibonacci(n-1)\ ) \rightarrow (\ n-2 \geq 0\ )$
8. $\{\ n-2 \geq 0\ \}$
   `y := fib(n-2);` **(IH2)**
   $\{\ y = fibonacci(n-2)\ \}$
9. $\{\ n-2 \geq 0 \wedge x = fibonacci(n-1)\ \}$
   `y := fib(n-2);` 7, 8 eta **(ODE)**
   $\{\ y = fibonacci(n-2)\ \}$
10. $\{\ \underbrace{n-2 \geq 0}_{def(y:=fib(n-2))} \wedge x = fibonacci(n-1)\ \}$
    `y := fib(n-2);` **(KA)**
    $\{\ x = fibonacci(n-1)\ \}$

Ohartu beharrekoa da 5. eta 10. urratsetan kontserbazioaren axioma **(KA)** erabiltzean, funtzioak albo-ondoriorik ez duela suposatzen ari garela. Ikus daitekeen bezala, $n \geq 0 \wedge n > 1$ aurrebaldintzatik abiatuz, lehenengo dei errekurtsiboa burutu ondoren x aldagaiaren balioa $fibonacci(n-1)$ izango da (frogapeneko 1-3 urratsak) eta $n-2$ ez da negatiboa izango (frogapeneko 4-6 urratsak). Bestalde, $n-2 \geq 0 \wedge x = fibonacci(n-1)$ aurrebaldintzatik abiatuz, bigarren dei errekurtsiboa burutu ondoren y aldagaiaren balioa $fibonacci(n-2)$ izango da (frogapeneko 7-9 urratsak) eta, gainera, x aldagaiaren balioa $fibonacci(n-1)$ izango da (frogapeneko 10. urratsa). Orain, alde batetik, 3. eta 6. urratsetako baieztapenak bateratu behar ditugu eta, beste aldetik, 9. eta 10. urratsetako baieztapenak bateratu behar ditugu.

Horrela lortutako bi baieztapenak erabiliz, dei errekurtsibo biak burutu ondoren x aldagaiaren balioa $fibonacci(n-1)$ izango dela eta y aldagaiaren balioa $fibonacci(n-2)$ izango dela baieztatu ahal izango dugu. Azken baieztapen hori `f := x+y;` agindua burutu ondoren f aldagaiaren balioa $fibonacci(n)$ izango dela frogatzeko beharko da. Hala ere, orain arte aurkeztu ditugun Hoareren kalkuluko axioma eta erregelen bidez ezin da formalki frogatu 3. eta 6. urratseko propietateen konjuntzioa betetzen denik ezta 9. eta 10. propietateen konjuntzioa betetzen denik ere. Arazo hori gainditzeko, Hoareren kalkuluko erregela berri bat aurkeztuko dugu, *konjuntzioaren erregela*:

(KJE) $$\frac{\{\,\varphi_1\,\}\ \mathtt{P}\ \{\,\psi_1\,\},\ \{\,\varphi_2\,\}\ \mathtt{P}\ \{\,\psi_2\,\}}{\{\,\varphi_1 \wedge \varphi_2\,\}\ \mathtt{P}\ \{\,\psi_1 \wedge \psi_2\,\}}$$

Semantikoki, aurreko erregela honela uler daiteke: programa bat aurre-ondoetako bi espezifikazio desberdinekiko partzialki zuzena dela froga badezakegu, orduan bi espezifikazio horietako aurrebaldintzen eta postbaldintzen konjuntziotik ateratzen den aurre-ondoetako espezifikazioarekiko ere partzialki zuzena izango da programa hori. Oso ohikoa da konjuntzioaren erregela erabiltzean aurrebaldintza biak edo postbaldintza biak baliokideak izatea. Adibidez, garatzen ari garen frogapenean aurrebaldintzak baliokideak dira, beraz $\varphi \wedge \varphi \equiv \varphi$ baliokidetasuna erabiliko dugu inplizituki. Aurreko frogapen hori honako urrats hauek erantsiz bukatuko dugu:

11. $\{\, n \geq 0 \wedge n > 1 \,\}$
    `x := fib(n-1);` 3, 6 eta **(KJE)**
    $\{\, n-2 \geq 0 \wedge x = fibonacci(n-1) \,\}$
12. $\{\, n-2 \geq 0 \wedge x = fibonacci(n-1) \,\}$
    `y := fib(n-2);` 9, 10 eta **(KJE)**
    $\{\, x = fibonacci(n-1) \wedge y = fibonacci(n-2) \,\}$
13. $\{\, n \geq 0 \wedge n > 1 \,\}$
    `x := fib(n-1);`
    `y := fib(n-2);` 11, 12 eta **(KPE)**
    $\{\, x = fibonacci(n-1) \wedge y = fibonacci(n-2) \,\}$
14. $(\, x = fibonacci(n-1) \wedge y = fibonacci(n-2) \,)$
    $\rightarrow\ (\, x+y = fibonacci(n-1) + fibonacci(n-2) \,)$
    $\rightarrow\ (\, x+y = fibonacci(n) \,)$
15. $\{\, x+y = fibonacci(n) \,\}$
    `f := x+y;` **(EA)**
    $\{\, f = fibonacci(n) \,\}$

16. $\{\, x = fibonacci(n-1) \wedge y = fibonacci(n-2) \,\}$
    `f := x+y;` 14, 15 eta **(ODE)**
    $\{\, f = fibonacci(n) \,\}$
17. $\{\, n \geq 0 \wedge n > 1 \,\}$
    `x := fib(n-1);`
    `y := fib(n-2);`
    `f := x+y;` 13, 16 eta **(KPE)**
    $\{\, f = fibonacci(n) \,\}$

4.5. BIBLIOGRAFIA-OHARRAK

Errekurtsio hitza latinezko *recursiõ*-tik eta *recurrõ* aditzetik dator. Aditz horrek 'itzuli' esan nahi du, eta haren adiera *«atzera itzuli, jatorrizko tokira edo egoerara itzuli»* da. Errekurtsioa, funtzioak definitzeko metodo bezala, XIX. mendean hasi zen erabiltzen eta lehenengo erabilera horiek matematikako indukzio-printzipioarekin oso lotura estua dute. Hain zuzen ere, zenbaki arrunten, zenbaki horien gainean errekurtsiboki definitutako eragiketa aritmetikoen eta zenbaki horien gaineko indukzioa definitzen duten (*Peano* edo *Dedekind-Peano*ren) axiomen lehenengo formalizazio teorikoetan erabili zen errekurtsioa ([28, 78]). XX. mendearen hasieran, funtzio eta multzoak definitzeko errekurtsio-teoriaren oinarriak definitu eta garatu zituzten ospe handiko matematikari askok: besteak beste, D. Hilbert, K. Gödel, J. Herbrand, S. C. Kleene, R. Péter eta T. Skolem-ek. [1] liburuan teoria honen historia interesgarria kontatzen da, parte hartu zuten egileen lan batzuk aipatuz. Lan horien artetik, Rózsa Péterrek 1934. urtean aurkeztutako [80] lana azpimarratuko dugu hemen. Lan hori funtzio errekurtsiboei buruz eduki algoritmiko handiko teorema sakonez beteta dago. Gainera, XX. mendeko erdialdetik aurrera, R. Péterrek funtzio errekurtsiboen teoria informatikan aplikatzeari ekin zion eta haren azken liburua [81], 1976. urtean idatzia, oso-osorik gai horri eskaini zion. Horrela, errekurtsioa programazio-teknika garrantzitsu bilakatu da, bereziki algoritmo eta programazio-lengoaien diseinuan.

Algoritmo errekurtsiboen egiaztapenari buruzko lehenengo lanak C. A. R. Hoarek argitaratu zituen 70eko hamarkadaren hasieran: [53, 41]. Lan horien ondoren, parametroak pasatzeko era desberdinen tratamendu formalari buruz eta aldagai lokalen inguruko problematikari buruz lortutako emaitzak osatu eta bateratu ziren beste lan askoren bidez. Horien artetik, [4, 47, 51, 61] nabarmendu nahiko genituzke. Elkarrekiko errekurtsioak [55] lanean agerian

ipini ziren beste arazo batzuk ere eragiten ditu[1]. Berehala, elkarrekiko errekurtsibitatea eta aldagai lokalak tratatzeko proposatutako sistema formalen zuzentasuna eta osotasuna (erlatiboa) frogatzeko balio zuten frogatzaile automatikoak aurkeztu ziren (ikus, esate baterako, [87, 60, 90]). Azken urte hauetan, algoritmo errekurtsiboen egiaztapena ulergarriago egitea eta haren inplementazioa, egiaztatzaile automatikoen barruan, kostu gutxiagokoa izatea dira helburu nagusiak. Ildo horretan [6, 29] lanak aipa daitezke. Kapitulu honetan aurkeztu den konjuntzioaren erregelaren erabilgarritasuna oso ondo erakusten da [6] lanean.

1. f funtzio batek beste g funtzio bati deitzen badio eta, bere aldetik, g funtzioak f-ri deitzen badio, elkarri deiak eginez horrela jarraituz, f eta g funtzioak *elkarrekiko errekurtsiboak* direla esaten da. Egoera hori gertatzen denean, f eta g funtzioek *errekurtsibitate gurutzatua* dutela ere esan ohi da. Bestalde, f funtzioak (edo g funtzioak) *zeharkako errekurtsibitatea* duela esango genuke. Oro har, bi funtziok baino gehiagok ere har dezakete parte era honetako errekurtsibitatean.

4.6. ARIKETAK: PROGRAMA ERREKURTSIBOEN EGIAZTAPENA

1. Formulatu honako dei honi dagokion indukzio-hipotesia:

```
w := h(x/2);
```

   h honako funtzio hau izanda:

```
function h(x: Integer) return z: Integer is
{ φ ≡ x > 0 }
{ ψ ≡ z = ⌊log2 x⌋ }  // x-ren 2 oinarriko logaritmoaren
                      // zati osoa da z
```

   Hor, φ eta ψ aurrebaldintza eta postbaldintza dira hurrenez hurren.

2. Erabaki jarraian datorren funtzioan agertzen den `lag := konb(m,n-1);` aginduari dagokion indukzio-hipotesia, proposatutako hiru aukeretatik zein den zuzena aukeratuz:

```
function konb(m,n: Integer) return z: Integer is
  lag: Integer;
{ 1 ≤ n ≤ m }
begin
  if n = 1 then
    z := m;
  else
    lag := konb(m,n-1);
    z := (lag/n)*(m-n+1);
  end if;
end konb;
{ z = m!/(n!*(m-n)!) }
```

   konb funtzioak $\binom{m}{n}$ konbinazio-zenbakia kalkulatzen du. Postbaldintzan $\binom{m}{n}$ konbinazio-zenbakia faktorialak erabiliz adierazi da.

   (A) $\{ 1 \leq n \leq m \}$ [ ]

```
lag := konb(m,n-1);
```

   $\{ lag = \dfrac{m!}{n! * (m-n)!} \}$

(B) $\{\, 1 \leq n-1 \leq m \,\}$ [ ]

```
lag := konb(m,n-1);
```

$\{\, z = \dfrac{m!}{n! * (m-(n-1))!} \,\}$

(C) $\{\, 1 \leq n-1 \leq m \,\}$ [ ]

```
lag := konb(m,n-1);
```

$\{\, lag = \dfrac{m!}{(n-1)! * (m-n+1)!} \,\}$

3. Honako funtzio honek bi zenbaki arrunten biderkadura kalkulatzen du.

```
function bider(x,y:  Integer) return z:  Integer is
{ x ≥ 0 ∧ y ≥ 0 }
begin
  if x = 0 or y = 0 then
    z := 0;
  elsif x = 1 then
    z := y;
  else
    z := bider(x/2,2*y);
    if (x % 2) /= 0 then
      z := z+y;
    end if;
  end if;
end bider;
{ z = x * y }
```

Zein da dei errekurtsiboari dagokion indukzio-hipotesia?

4. Frogatu formalki kasu induktiboaren zuzentasuna honako programa errekurtsibo hauetan.

   4.1. Zenbaki arrunt bat emanda, *dig_hand* funtzioak zenbaki horren digitu handiena itzuliko du.

```
function dig_hand(x:  Integer)
return y:  Integer is
  w:  Integer;
{ x ≥ 0 }
begin
  if x <= 9 then
    y := x;
  else
    w := dig_hand(x/10);
    if w > (x % 10) then
      y := w;
    else
      y := x % 10;
    end if;
  end if;
end dig_hand;
{ y = handiena{ x / 10^(i-1) % 10 | i ≥ 1} }
```

   4.2. *konb* funtzioak $\binom{m}{n}$ konbinazio-zenbakia kalkulatzen du.

```
function konb(m,n:  Integer) return e:  Integer is
  e1,e2:  Integer;
{ m ≥ n ≥ 0 }
begin
  if m = n or n = 0 then
    e := 1;
  else
    e1 := konb(m-1,n);
    e2 := konb(m-1,n-1);
    e := e1+e2;
  end if;
end konb;
{ e = m! / (n!*(m-n)!) }
```

4.3. frm funtzioak 3 eta 2ren n-garren berreturen arteko aldea kalkulatzen du.

```
function frm(n:  Integer) return e:  Integer is
  e1,e2:  Integer;
{ n ≥ 0 }
begin
  if n <= 1 then
    e := n;
  else
    e1 := frm(n-1);
    e2 := frm(n-2);
    e := (5*e1)-(6*e2);
  end if;
end frm;
{ e = 3^n − 2^n }
```

5. Eman aurre-ondoetako espezifikazioa eta frogatu formalki zuzentasun osoa honako programa hauentzat.

5.1. zat funtzioak arrunta den x zenbakiaren eta positiboa den y zenbaki osoaren arteko zatidura eta hondarra kalkulatzen ditu.

```
function zat(x,y:  Integer)
return (z,h):  (Integer,Integer) is
begin
  if x < y then
    (z,h) := (0,x);
  else
    (z,h) := zat(x-y,y);
    (z,h) := (z+1,h);
  end if;
end zat;
```

Definizio horretan ikusten den bezala, funtzio honek emaitza gisa bi zenbaki osoz eratutako bikote bat itzuliko du. Bikoteen gaineko esleipena aldibereko esleipenaren baliokidetzat hartuko dugu.

5.2. Osoa eta positiboa den x zenbakia eta 1 baino handiagoa den *oin* zenbaki osoa emanda, $log(oin, x)$ funtzioak *oin* oinarriko x-ren logaritmoaren zati osoa kalkulatzen du
(Oharra: $(log_{oin}\ x = z) \leftrightarrow (oin^z = x)$).

```
function log(oin,x:  Integer)
return y:  Integer is
begin
  if x = 1 or x < oin then
    y := 0;
  else
    y := log(oin,x/oin);
    y := y+1;
  end if;
end log;
```

5.3. Osoa eta positiboa den x zenbakia eta osoa eta ez-negatiboa den y zenbakia emanda, *ber* funtzioak berretura kalkulatzen du, hau da, x^y.

```
function ber(x,y:  Integer) return e:  Integer is
  m:  Integer;
begin
  if y = 0 then
    e := 1;
  elsif bikoitia(y) then
    m := ber(x,y/2);
    e := m*m;
  else
    m := ber(x,y/2);
    e := m*m*x;
  end if;
end ber;
```

5. Datu-moten espezifikazio ekuazionala

Datu-mota abstraktuak (DMA-ak) espezifikatzeko teknika ekuazionala edo aljebraikoa deskribatuko da kapitulu honetan. 5.1. atalean, DMA kontzeptua aurkeztuko da. Kontzeptu hori funtsezkoa da datu-egitura kontzeptuaren bilakaeran. Ondoren, 5.2. atalean, espezifikazio ekuazionalaren teknika azalduko da, eta 5.3. atalean teknika hori oinarrizko hiru DMA definitzeko aplikatuko da: sekuentziak, pilak eta zuhaitz bitarrak. Hiru DMA horiek asko erabiltzen dira programazioan. 5.4., 5.5. eta 5.6. ataletan, sekuentziak, pilak eta zuhaitz bitarrak lantzen hasteko ariketa sinpleak jaso dira hurrenez hurren. 5.7. atalean, espezifikazio ekuazionaletatik abiatuta propietateak frogatu ahal izateko bi frogapen-metodo azalduko dira: dedukzioa eta indukzioa. Atal berak bi azpiatal ditu aipatutako bi frogapen-metodoak aplikatzeko ariketekin. Espezifikazio ekuazional edo aljebraikoen gaineko erreferentzia bibliografikoak 5.8. atalean bildu dira. Azkenik, 5.9. atalean, gaiarekin lotutako ariketa-bilduma luzea dator.

5.1. DATU-MOTA ABSTRAKTUAK (DMA-AK)

Aurreko kapituluetan, bai aurredefinitutako oinarrizko datu-motatakoak diren aldagaiak, hots, zenbaki osoak edo boolearrak, eta bai bektoreak (edo array-ak) bezalako datu-mota egituratuak erabiltzen dituzten programei buruzko arrazoibide formalak egin ditugu. Programen gaineko arrazoibideetan datu-motei aplika dakizkiekeen eragiketen propietateak erabili ditugu. Adibidez, zenbaki osoen zein konjuntzio boolearraren elkarkortasuna, edo zatiketa osoaren oinarrizko propietateak. Horrelako objektuak eta beraien propietateak erabiltzeko dugun ohitura handia dela-eta, arrazoibide formalerako funtsezkoak diren hainbat alderditan ez gara zeharo esplizituak izan. Alderdi horietatik, honako hauek nabarmenduko ditugu:

- Eragiketetako argumentuen kopuruaren eta argumentu horien motaren ikuspuntutik sintaktikoki zuzenak diren adierazpenak bakarrik erabiltzen ditugu. Adibidez, x eta z zenbaki osoak izanda, $bikoitia\_da(x) + z$

ez da ondo eraikitako adierazpena[1], $bikoitia\_da(x+z)$, aldiz, bai.

- Erabilitako eragiketen propietate aljebraikoak ezagunak dira eta ondo definituta daude, kalkulu eta dedukzioen garapenerako oinarri formal sendoa emanez.

- Ez diogu arretarik jarri ez datuen barne-egiturari, ezta eragiketen inplementazioari ere, baizik eta haien propietate aljebraikoei. Hau da, (edozein programazio-lengoaiatan exekuzioari datxezkion) errepresentazio eta inplementazioarekin lotutako alderdietatik aldendu eta portaeraren espezifikazio edo deskribapenean jarri dugu arreta. Adibidez, *bikoitia_da* eragiketa eta zenbaki osoen propietateetan oinarrituta dagoen honako propietate hau erabiltzeko, ez da beharrezkoa zenbaki osoak eta boolearrak nola errepresentatzen eta inplementatzen diren jakitea:

$$\forall x\, \forall z\, (\, (\, bikoitia\_da(x) \wedge bikoitia\_da(z)\,) \rightarrow bikoitia\_da(x+z)\,)$$

Programazio-lengoaiek erabilgarri jartzen dituzten datu-mota aurredefinituez gain, softwarearen garapenak datu-mota berriak definitzea eskatzen du askotan. Bai erabiltzaileak definitutako moten kasuan, bai aurredefinitutakoen kasuan, datu-motaren espezifikazioa eta barne-errepresentazioa zein inplementazioa garbi bereiztea komeni da. Hori dela-eta, *abstrakzioa* egiten dela esango dugu; hau da, datu-motaren erabiltzaileak ez ditu nahastu behar inplementazioari dagozkion xehetasunak portaeraren propietateekin. Bere aldetik, datu-motaren inplementatzailea inplementazioaren xehetasunez arduratuko da. Abstrakzioarekiko komenientzia horrek eraginda sortu zen *Datu-Mota Abstraktu*en (DMA) kontzeptua. DMA baten definizioan ezinbestezkoa da mota horretakoak diren objektuen gaineko eragiketen portaera era argian eta zehatzean espezifikatzea, espezifikazio hori beteko duen inplementazioa alde batera utzita. Horrela, espezifikazioak DMA-a erabiliko duten programen gainean arrazoitzeko eta algoritmoak diseinatzeko formalismoa eskaintzen du. Hortaz, bi eremu desberdintzen dira: batetik, DMA-ak inplementatzen dituzten programak, eta, bestetik, DMA-ak erabiltzen dituztenak. Bereizte horrek atazak banatzea ahalbidetzen du, DMAen inplementazioa DMAen erabilerarekiko independentea izanik, eta alderantziz. Datu-mota garatzen duenaren eta erabiltzailea izango denaren arteko kontratu moduko bat ezartzen da. Kontratu horren arabera, garatzaileak eragiketak inplementatzen ditu eta eragiketa horien espezifikazioa esportatzen du (ikusgarri jartzen du); bestalde, datu-mota erabiltzen duenak eragiketak erabil ditzake

1. $bikoitia\_da(x)$ adierazpenak balio boolearra itzuliko baitu, eta ez zenbaki osoa.

eragiketa horien espezifikazioaren arabera, eta era horretan, inplementazioarekiko independentzia mantenduko du. Horrek guztiak aukera ematen du datu-mota osatzen duten objektuen eta objektu horiekin egin daitezkeen eragiketen portaeraren gainean formalki arrazoitzeko. Konkretuki, programen egiaztapenean eta eratorpenean aritzean funtsezkoa da aipatutakoa. Formalki ondo definituta ez leudekeen datu-motak erabiliko lituzketen programen zuzentasunaz arrazoitzea ezinezkoa litzateke.

Azken finean DMA-aren espezifikazioa DMA-aren sortzailearen eta erabiltzailearen arteko kontratuaren deskribapena da. Espezifikazio egokia definituz, eta ez bestela, lor daiteke DMA-a erabiltzen duten programatzaileen eta DMA-a inplementatzen dutenen artean lan-banaketa orekatua eta komunikazio egokia. DMA-aren barne-errepresentazioan edo inplementazioan egiten den edozein aldaketak ez du eraginik izango DMA-a erabiltzen duten programetan (beti ere egindako aldaketak espezifikazioan eraginik ez badu).

5.2. ESPEZIFIKAZIO EKUAZIONALAREN TEKNIKA

Espezifikazio ekuazional edo aljebraiko izenez ezagutzen den DMA-ak espezifikatzeko teknikak ekuazioak erabiltzen ditu espezifikatu beharreko DMA-aren gaineko eragiketak deskribatzeko[2].

Terminoen arteko ekuazioak idatzi ahal izateko, izena, ondo eraikitako terminoak eta ondo ez eraikitako adierazpenak desberdintzeko aukera ematen duen sintaxia eduki behar dute eragiketek. Hortaz, bi zatik osatzen dute espezifikazio ekuazionala: *signatura*k eta *ekuazio*ek. Signaturak DMAekin lotutako adierazpenen (terminoen) sintaxia deskribatzen du. Horretarako, liburu honetan adierazpen horietan rol nagusia edo laguntzailea duten *motak* erazagutuko dira, eta *eragiketa* bakoitzaren sintaxia deskribatuko da. Eragiketa bakoitzeko honako hau espezifikatuko dugu:

- Izena edo identifikatzailea.
- Argumentu kopurua eta argumentu bakoitzaren mota.
- Emaitzaren mota.

Eragiketak bi multzo handitan sailka daitezke. Alde batetik, eragiketa *eraikitzaileak* ditugu. Eragiketa horiek DMA-a osatzen duten objektuak eraikitzeko aukera ematen dute. Bestetik, objektuen eraikuntzara bideratuta ez dauden eragiketak izango ditugu. Azken horiek eragiketa *ez-eraikitzaile* izenez ezagutzen dira. Eragiketa eraikitzaileak bakarrik erabiliz mota osatzen

2. Jatorrizko adjektibo kalifikatzailea *aljebraikoa* da, teknika aljebren azterlan orokorretan erabilitako logiketatik sortu baitzen.

duten objektuez eratutako unibertso osoa sor daiteke. Aldi berean, eragiketa eraikitzaileak *aratzak* direla esango dugu, beraiekin eraikitako terminoen multzoaren eta mota osatzen duten objektuen artean korrespondentzia biunibokoa (edo bijektiboa) dagoenenean; hau da, termino bakoitzak objektu desberdin bat adierazten du, eta, alderantziz, objektu bakoitza termino bakar baten bitartez adieraz daiteke funtzio eraikitzaileak eta aldagaiak bakarrik erabiliz. Bestalde, eragiketa eraikitzaileak *ez-aratzak* direla esango dugu, beraiekin eraikitako terminoen multzoaren eta mota osatzen duten objektuen artean korrespondentzia unibokoa bakarrik dagoenenean; hau da, termino bakoitzak objektu bakarra adierazten du, baina objektu bat termino batek baino gehiagok adieraz dezakete. Mota baten balioen unibertsoa funtzio eraikitzaileak erabiliz definitzearen azpian egiturazko indukzioaren printzipioa dago. Printzipio hori dagoeneko 4. kapituluan azaldu da. Eragiketa *ez-eraikitzaile*en artean eragiketa *suntsitzaileak* bereiz daitezke, zeinak edozein eragiketa eraikitzaile ez-konstanteren (gutxienez parametro bat duten eragiketa eraikitzaileen) alderantzizko funtzioak baitira. Adibidez, kate baten bukaeran elementu bat txertatzen duen funtzioaren alderantzizko funtzioak honako hauek izango lirateke: azken elementua itzultzen duen funtzioa eta azken elementua kenduz geratzen den katea itzultzen duen funtzioa. Gainerako eragiketa ez-eraikitzaileak *kontsulta*ra bideratutako eragiketak dira. Adibidez, objektu bat eragiketa eraikitzaile konstantea den ala ez erabaki, luzera kalkulatu eta abar.

Ekuazioak signaturaren eragiketen eragina edo semantika deskribatzeko erabiltzen dira. Oro har, hiru ekuazio mota desberdintzen dira:

- Berdintzak, bi terminoren arteko baliokidetasuna deskribatzen dutenak. $t_1 = t_2$ itxurakoak dira, eta t_1-ek adierazten duen objektua t_2-k adierazten duen objektu bera dela adierazten dute.

- Baldintzazkoak, baldintza baten araberako baliokidetasuna deskribatzen dutenak. $t_1 = t_2$ baldin B itxurakoak dira. Kasu honetan, t_1 eta t_2 terminoen arteko berdintza gertatuko da, baldin eta B baldintza betetzen bada. Aldi berean, B baldintza $t_3 = t_4$ erako ekuazioa izan daiteke.

- Errore-ekuazioak, balioespena egitean errorea sortzen duten terminoak deskribatzen dituztenak. t terminoaren balioespenak errorea sortzen duenean honako ekuazioa idatziko dugu: $t = \texttt{errorea}$[3].

3. Era egokian idatzitako termino baten balioespenak errorea emateak esan nahi du termino horrek ez duela errepresentatzen motaren objektu bat, nahiz eta terminoa sintaktikoki zuzena den adierazpena izan eta teorikoki motaren objektu bat errepresentatu beharko lukeen.

Ekuazioen interpretazioa egitean honako hau kontuan izan behar da:

- Ekuazioak unibertsalki kuantifikatuta daudela suposatzen da. Hau da, $t_1 = t_2$ ekuazioa inplizituki honako ekuazio honen baliokidea da: $\forall x_1 \ldots \forall x_n (t_1 = t_2)$, hor $ald(t_1) \cup ald(t_2) = \{x_1, \ldots, x_n\}$ izanik.
- Erroreak hedatu egiten dira. Beraz, t terminoak errorea eragiten badu, t terminoa barne duen beste edozein terminok ere errorea eragingo du. Adibidez,

  $$\frac{x}{0} = \texttt{errorea}$$

  betetzen bada, orduan

  $$(\ \frac{x}{0}\ ) + y = \texttt{errorea}$$

  beteko da.

Ondoren, azaldutako kontzeptuak erakusteko zenbaki arruntak erabiliko ditugu: *nat* DMA-a, hain zuzen. Zenbaki arruntek osatutako objektu multzoa espezifikatzea izango dugu helburu, *zero* eta *hurrengoa* eragiketa eraikitzaileak erabiliz eta egiturazko indukzio bidez eraikiz. Zenbaki arrunt baten aurreko zenbakia kalkulatzen duen eragiketa osagarria ere izango dugu. Espezifikazioaren klausulak, hots, signatura eta ekuazioak, honako hauek dira[4]:

Mota: *nat*
Lag: (ez dago)
Eragiketak:

$zero: \rightarrow nat$	(eraikitzailea)
$hurrengoa: nat \rightarrow nat$	(eraikitzailea)
$aurrekoa: nat \rightarrow nat$	

Ekuazioak:

(n.1) $aurrekoa(zero) = \texttt{errorea}$
(n.2) $aurrekoa(hurrengoa(n)) = n$

Bi eragiketa eraikitzaileak erabiliz (*zero* eta *hurrengoa*), espezifikatutako DMA-aren objektu guztiak honela definitzen dira:

$$zero,\ hurrengoa(zero),\ hurrengoa(hurrengoa(zero)),\ \ldots$$

Hor, objektu guztiak desberdinak izango dira. Azpimarratu behar da eragiketa eraikitzaileen bidez (*zero* eta *hurrengoa*) osatutako terminoen multzoaren eta zenbaki arrunten multzoaren artean korrespondentzia biunibokoa

4. Hemendik aurrera «Lag» klausula ez da agertuko, hutsa baldin bada.

ezartzen dela. Hortaz, *zero* eta *hurrengoa* eragiketak *nat* datu-motaren eraikitzaile aratzak direla esan dezakegu. Azpimarratzekoa da definizio hori egiturazko indukzioan oinarritzen dela, *nat* motako edozein balio *zero* (oinarrizko eraikitzailea) edo *hurrengoa*(n) (eraikitzaile induktiboa) erakoa izanda, eta n aldi berean *nat* motako balioa izanda.

nat motako objektuak adierazteko *zero* eta *hurrengoa* eragiketak erabiltzeak helburu argitzailea du, azken batean errazagoa baitzaigu termino horiek berridaztea, *zero* eraikitzailearen ordez 0 balioa erabiltzea, eta *hurrengoa* eragiketaren ordez haren baliokidea den $+1$ eragiketa erabiltzea. Era horretan, lehen ikusitako *zero*, *hurrengoa*(*zero*), *hurrengoa*(*hurrengoa*(*zero*)), ... terminoak honela berridatz daitezke: 0, $0+1$, $(0+1)+1$, ... edo, oraindik era sinpleagoan honela: 0, 1, 2, ...

nat multzoaren espezifikazio sinplifikatua aurkeztu dugu espezifikazio ekuazionalen definiziorako zein erabilerarako sarrera gisa. Ondoren, espezifikazio ekuazionaletarako jarraibideak xehetasun gehiagoz azalduko ditugu.

Esan dugunez, eragiketa eraikitzaileek zehazten dute espezifikatu beharreko mota osatzen duten objektuen multzoa. Azken finean, eragiketa eraikitzaileek objektuen unibertsoa definitzen dute eta, ondorioz, beraien portaera deskribatzeko ekuazioak definitu beharrik ez dago.

Bestalde, eragiketa ez-eraikitzaileak ekuazioen bitartez definitzen dira. Ekuazio horiek espezifikatu nahi den mota osatzen duten balioen gainean eragiketa ez-eraikitzaileak aplikatzearen emaitzak definitzen dituzte. *nat* datu-motaren espezifikazioan azpimarratu dugu mota horretako edozein balio *zero* eta *hurrengoa* funtzio eraikitzaileen bitartez adieraz daitekeela egiturazko indukzioa erabiliz. Hortaz, *nat* motaren gaineko edozein eragiketa ez-eraikitzaileren portaera definitzeko ekuazioek *zero* eta *hurrengoa*(n) argumentuak edukiko dituzte, eta, era horretan, *nat* motako balio posible guztien gaineko portaera definituko da. Definitzeko era hau egiturazko indukzioa erabiliz funtzio eraikitzaileen bidez objektuak definitzeko aukera duen edozein DMAtara orokor daiteke.

Gerta daiteke eragiketa ez-eraikitzaile baten aplikazioa ez egotea definituta datu-mota osatzen duten objektu multzokoa den objektu konkretu baterako. Kasu horretan errore-ekuazioa definitu beharra dago.

Aurretik azaldutako *nat* motari dagokion adibidera itzuliz, DMA horren espezifikazioan *aurrekoa* eragiketa ez-eraikitzailea (edo haren baliokidea den -1) ere definitu dugu. Eragiketa horren semantika funtzio eraikitzaileen gaineko ekuazioen bidez definitu da; hau da, *aurrekoa*(*zero*) eta *aurrekoa*(*hurrengoa*(n)) espresioen balioa definitu da bi ekuazioren bidez. *aurrekoa* eragiketa ez dago definituta *zero* eraikitzailerako, emaitza ez baita

nat motakoa. Errore-ekuazioa da, beraz. Bestalde, $hurrengoa(n)$ terminoaren gainean *aurrekoa* aplikatzeak ez du arazorik, eta berdintza-ekuazio baten bitartez deskribatzen da. Hortaz, *aurrekoa* eragiketa ez-eraikitzailea ondoko bi ekuazioen bitartez definituta gelditzen da:

(n.1) $aurrekoa(zero) = \texttt{errorea}$
(n.2) $aurrekoa(hurrengoa(n)) = n$

Liburu honetan eraikitzaile ez-aratzak dituzten datu-mota abstraktuekin arituko ez garen arren, azaldu berri dugun adibidea erabiliko dugu ezaugarri hori duten datu-moten espezifikazioek aurkezten duten problematikaren zertzelada bat emateko. Demagun *nat* motari dagokion espezifikazioa hartuz *aurrekoa* eragiketa hirugarren eragiketa eraikitzaile gisa duen datu-mota berri bat definitzea erabakitzen dugula, zenbaki negatiboak ere eduki nahi ditugulako; hau da, zenbaki arrunten ordez zenbaki osoak definitu nahi ditugu. Kasu horretan espezifikazioa honako hau litzateke:

Mota: *oso*
Eragiketak:
$zero$: $\rightarrow oso$ (eraikitzailea)
$hurrengoa$: $oso \rightarrow oso$ (eraikitzailea)
$aurrekoa$: $oso \rightarrow oso$ (eraikitzailea)
Ekuazioak:
(o.1) $aurrekoa(hurrengoa(n)) = n$
(o.2) $hurrengoa(aurrekoa(n)) = n$

Ukaezina da eragiketa eraikitzaileak ez direla aratzak, eta bi ekuazioak ez dira *aurrekoa* eta *hurrengoa* definitzeko, baizik eta ez-aratzak izatearen zergatia zehazteko. Beraz, bi ekuazio horiek eraikitzaileen bidez definituta dauden eta beraien artean baliokideak diren termino-bikote posible guztiak zehazten dituzte; hau da, zenbaki oso bera errepresentatzen duten terminoak zehazten dira. $aurrekoa(zero)$ terminoarentzat ekuazio berezirik ez dago $aurrekoa(zero)$ terminoa *oso* datu-motako objektua baita oraingo honetan. Eraikitzaileak aratzak ez direnean, eraikitzaileen gainean egiturazko indukzioz eragiketa ez-eraikitzaileak definitzen dituzten axiomez gain, eraikitzaileez bakarrik idatzitako terminoen arteko baliokidetasunak deskribatzeko beharrezkoak diren axiomak ere ipini behar dira espezifikazioetan.

kontagailua izeneko ondoko DMA-a *nat* motaren aldaera bat da bi eragiketa berri dituelarik: *kontagailua* DMA-aren lehenengo hiru eragiketak bakarrik (bi eraikitzaileak eta konstantea ez den eraikitzaile bakarraren suntsitzailea) kontuan izanik *nat* motaren baliokidea den DMA-a lortzen da, baina izen desberdinekin. Lehen urrats gisa honako signatura hau zehaztuko dugu:

Mota: *kontagailua*
Lag: *nat*
Eragiketak:

hasierakoa: $\rightarrow$ *kontagailua*	(eraikitzailea)
gehitu: *kontagailua* $\rightarrow$ *kontagailua*	(eraikitzailea)
gutxitu: *kontagailua* $\rightarrow$ *kontagailua*	
hasieratu: *kontagailua* $\rightarrow$ *kontagailua*	
balioa: *kontagailua* $\rightarrow$ *nat*	

Bigarren urrats gisa eragiketa ez-eraikitzaileak (*gutxitu*, *hasieratu* eta *balioa*) definituko ditugu, eragiketa eraikitzaileen gainean egiturazko indukzioa aplikatuz. Horrela, honako sei ekuazio hauek sortuko dira:

Ekuazioak:

(k.1) $gutxitu(hasierakoa) = \texttt{errorea}$
(k.2) $gutxitu(gehitu(x)) = x$
(k.3) $hasieratu(hasierakoa) = hasierakoa$
(k.4) $hasieratu(gehitu(x)) = hasierakoa$
(k.5) $balioa(hasierakoa) = zero$
(k.6) $balioa(gehitu(x)) = hurrengoa(balioa(x))$

Kontuan izan *gutxitu* eragiketa ez dagoela definituta *hasierakoa* eraikitzailearentzat, *hasieratu* eragiketak beti *hasierakoa* itzuliko duela, eta *balioa* eragiketak *kontagailu* motako objektuak *nat* motako balio bihurtuko dituela definizio induktibo baten bitartez.

kontagailua DMA-aren espezifikazio ekuazionalean mota horren gaineko eragiketak deskribatu dira, bai ikuspuntu sintaktikotik (idazkera formala) bai ikuspuntu semantikotik (portaera).

bool motari dagokionez, T eta F balioez osatuta dagoela kontsideratuko dugu, bi balio horiek argumenturik gabeko eragiketa eraikitzaileak direlarik (konstanteak, hain zuzen). Horretaz gain, eragiketa ez-eraikitzaileak ere definitzakegu, hots, prefixu bidezko notazioa duen $\neg$ eragiketa (ukapena), eta infixu bidezko notazioa duten $\vee$ (disjuntzioa) eta $\wedge$ (konjuntzioa) eragiketak. Eragiketa horiek definitzeko eraikitzaileetan oinarri gaitezke ($\neg$ eta $\vee$ eragiketen kasuan bezala), edo aurretik definitutako eragiketak erabil ditzakegu ($\wedge$ eragiketaren kasuan, esaterako, definizioak $\neg$ eta $\vee$ erabiltzen ditu):

Mota: *bool*
Eragiketak:
T: $\rightarrow bool$ (eraikitzailea)
F: $\rightarrow bool$ (eraikitzailea)
$\neg\ \_$: $bool \rightarrow bool$
$\_ \vee \_$: $bool \times bool \rightarrow bool$
$\_ \wedge \_$: $bool \times bool \rightarrow bool$
Ekuazioak:
(b.1) $\neg\mathsf{T} = \mathsf{F}$
(b.2) $\neg\mathsf{F} = \mathsf{T}$
(b.3) $\mathsf{T} \vee \mathsf{x} = \mathsf{T}$
(b.4) $\mathsf{F} \vee \mathsf{x} = \mathsf{x}$
(b.5) $x \wedge y = \neg(\ \neg x \vee \neg y\ )$

Kontuan izan *disjuntzioa* ($\vee$) eragiketaren definizioak lehen argumentuaren gaineko egiturazko indukzioa erabiltzen duela, hau da, lehen argumentuan funtzio eraikitzaileak agertzen dira, bigarren argumentuan, aldiz, aldagaiak.

Edozein DMAri eragiketa berriak gehi dakizkioke, bere oinarrizko funtzionalitatea aberasteko asmoz. Eta eragiketa horien espezifikazioek dagoeneko deskribatu ditugun jarraibideak beteko dituzte: signaturan sintaxia definituko da (hemen DMA berri laguntzaileak gehi daitezke), eta motaren eraikitzaileen gaineko ekuazioen bitartez eragiketa berrien portaera definituko da. Adibidez, *nat* DMA-ari gehitutako *zeroa_da* eragiketa honela espezifikatuko litzateke:

Mota: *nat*
Lag: *bool*
Eragiketa:
$zeroa\_da$: $nat \rightarrow bool$
Ekuazioak:
(n.3) $zeroa\_da(zero) = \mathsf{T}$
(n.4) $zeroa\_da(hurrengoa(n)) = \mathsf{F}$

Era berean, *txikiago edo berdin* ($\leq$), *txikiago* ($<$), *handiago* ($>$) eta *handiago edo berdin* ($\geq$) eragiketa erlazional infixuei dagokien espezifikazioa honako hau da:

Mota: *nat*
Lag: *bool*
Eragiketak:
$\_ \leq \_$: $nat \times nat \to bool$
$\_ < \_$: $nat \times nat \to bool$
$\_ > \_$: $nat \times nat \to bool$
$\_ \geq \_$: $nat \times nat \to bool$
Ekuazioak:
(n.5) $zero \leq y = \mathsf{T}$
(n.6) $hurrengoa(x) \leq zero = \mathsf{F}$
(n.7) $hurrengoa(x) \leq hurrengoa(y) = x \leq y$
(n.8) $zero < zero = \mathsf{F}$
(n.9) $zero < hurrengoa(y) = \mathsf{T}$
(n.10) $hurrengoa(x) < zero = \mathsf{F}$
(n.11) $hurrengoa(x) < hurrengoa(y) = x < y$
(n.12) $x > y = (\, y \leq x \, \wedge \, \neg \, (\, x \leq y \,) \,)$
(n.13) $x \geq y = y \leq x$

Goiko espezifikazioan eragiketa eraikitzaileen gaineko indukzioa zorrotz jarraitzen ez duten ekuazioak erabili ditugu. Horrela, (n.12)-k eta (n.13)-k eragiketa bana definitzen dute aldez aurretik definitutako eragiketak erabiliz. (n.5) ekuazioak bi ekuazio laburbiltzen ditu bakar batean, alde batetik, y zero deneko kasuari dagokiona, eta, beste aldetik, y beste zenbaki baten hurrengoa deneko kasuari dagokiona, kasu bietan balioa T baita. Kontuan izan (n.8) eta (n.9) ekuazioak ekuazio bakar batekin ere ordezka daitezkeela: $zero < y = (y \neq zero)$, hor $\neq$ ikurrak desberdintasun sintaktikoa adierazten duelarik, eta eraikitzaile aratzak dituzten mota aljebraikoen kasuan desberdintasunarekin bat dator. Era berean, eraikitzaile aratzak dituzten moten objektuen berdintasuna berdintasun sintaktikoarekin bat dator. Arrazoi horregatik ez dugu definitu ez $=$ ez $\neq$ aurreko espezifikazioan.

Batuketa $(+)$ eta *kenketa* $(-)$ eragiketa infixuekin eta *handiena* eragiketarekin *nat* datu-mota aberasteko honako espezifikazio hau idatziko genuke:

Mota: *nat*
Eragiketak:
$\_ + \_$: $nat \times nat \to nat$
$\_ - \_$: $nat \times nat \to nat$
handiena: $nat \times nat \to nat$

Ekuazioak:

(n.14) $zero + y = y$

(n.15) $hurrengoa(x) + y = hurrengoa(x + y)$

(n.16) $$zero - y = \begin{cases} \texttt{errorea} & \text{baldin } \neg\, zeroa\_da(y) \\ zero & \text{bestela} \end{cases}$$

(n.17) $$hurrengoa(x) - y = \begin{cases} \texttt{errorea} & \text{baldin } y > hurrengoa(x) \\ hurrengoa(x) & \text{baldin } zeroa\_da(y) \\ x - aurrekoa(y) & \text{bestela} \end{cases}$$

(n.18) $$handiena(x, y) = \begin{cases} x & \text{baldin } y \leq x \\ y & \text{bestela} \end{cases}$$

Kenketaren kasuan (n.17) ekuazioak errorea sortuko du, baldin eta lehenengo argumentua zero ez bada eta bigarren argumentua baino txikiagoa bada. Hori gertatuko litzateke, adibidez, $2-3$ kalkulatzean:

$$\begin{aligned} & hurrengoa(hurrengoa(zero)) - hurrengoa(hurrengoa(hurrengoa(zero))) \\ & \quad =_{(n.17)} \quad \texttt{errorea} \end{aligned}$$

(n.17) ekuazioa beste era batera ere formula daiteke. Formulazio alternatibo horri (n.17)' deituko diogu: alde batetik, $zeroa\_da(y)$ betetzen baldin bada, $hurrengoa(x) - y$ adierazpenaren balioa $hurrengoa(x)$ izango da; eta, beste aldetik, $hurrengoa(x) - y$ adierazpenaren balioa $x - aurrekoa(y)$ izango da gainerako kasu guztietan. *nat* motako bi elementuen arteko kenketa kalkulatzean, lehenengoa zero ez balitz eta bigarren elementua baino txikiagoa balitz, aipatutako (n.17)' formulazio alternatiboa errekurtsiboki aplikatzen jarraituko litzateke lehenengo argumentua *zero* izan arte. Une horretan errorea sortuko litzateke (n.16) ekuazioaren aplikazioaren ondorioz. Jarraian, (n.17)' erabiliz $2-3$ kenketa kalkulatzeko prozesua aurkeztuko da:

$$\begin{aligned} & hurrengoa(hurrengoa(zero)) - hurrengoa(hurrengoa(hurrengoa(zero))) \\ & \quad =_{(n.17)'} \quad hurrengoa(zero) - \\ & \qquad\qquad aurrekoa(hurrengoa(hurrengoa(hurrengoa(zero)))) \\ & \quad =_{(n.2)} \quad hurrengoa(zero) - hurrengoa(hurrengoa(zero)) \\ & \quad =_{(n.17)'} \quad zero - aurrekoa(hurrengoa(hurrengoa(zero))) \\ & \quad =_{(n.2)} \quad zero - hurrengoa(zero) \\ & \quad =_{(n.16)} \quad \texttt{errorea} \end{aligned}$$

nat edo *bool* bezalako datu-mota ez-egituratuen espezifikazio ekuazionaletatik egituratuak diren *nat_en_multzoak* eta *bool_en_multzoak* edo

nat_en_kateak eta *bool_en_kateak* bezalako datu-mota egituratuen espezifikazio ekuazionaletara aldatzean, multzo desberdinen espezifikazioak funtsean baliokideak direla ikus daiteke, mota desberdineko kateen espezifikazioak ere baliokideak izango diren bezala. Beste era batean esanda, datumota egituratu (multzo, kate eta abar) baten gaineko edozein eragiketa datumota osatzen duten osagaien motarekiko (zenbaki arruntak, boolearrak eta abar) independenteki defini daiteke. Hortaz, multzo, kate edo edozein datumota egituraturen espezifikazio desberdinak (espezifikazio bat osagai mota posible bakoitzeko) espezifikazio bakar batekin ordezka ditzakegu, hau da, *T_ren_multzoak*, *T_ren_kateak* eta abar, T edozein datu-mota espezifikorekin ordezka daitekeen parametroa izanda, adibidez, *nat* edo *bool*. DMA egituratuen espezifikazioetan erabiltzen den notazioaren arabera, DMA-aren osagaien mota errepresentatzen duen parametroa parentesien artean jartzen da. Beraz, *array*, *pila*, *ilara*, *sekuentzia* eta *zuhaitza* datu-mota egituratuak $array(T)$, $pila(T)$, $ilara(T)$, $sekuentzia(T)$ eta $zuhaitza(T)$ espezifikatuko dira, T osagaien mota errepresentatzen duen parametro generikoa izanda. Beraz, $sekuentzia(bool)$, $pila(nat)$, $zuhaitza(nat)$ eta abar, T parametroa mota konkretu batekin ordezkatzearen ondorioz lortzen diren datumota espezifikoak dira.

5.3. OINARRIZKO DMA-EN ESPEZIFIKAZIO EKUAZIONALA

Atal honetan datu-egituren arloan ezagunenetakoak diren hiru datu-moten espezifikazio ekuazionala ikusiko dugu: sekuentziak, pilak eta zuhaitz bitarrak. Hiru DMAek funtsezko ezaugarri bat partekatzen dute: egiturazko indukzioan oinarritutako definizioa onartzen dute. Ezaugarri hori dela-eta, beraien espezifikazio ekuazionala eragiketa eraikitzaile aratzetan oinarritzen da, eta espezifikazioa bera naturalki sortzen da.

5.3.1. Sekuentziak

Sekuentziak egitura linealak dira. Ez dute ez luzera finkorik ez aurredefinitutako luzerarik ere. Sekuentzia bereko osagai guztiak mota berekoak dira, mota edozein izan daitekeelarik. Hala izanik, $sekuentzia(T)$ notazioa erabiliko dugu edozein T motatako *sekuentzia* DMA-a adierazteko.

$sekuentzia(T)$ DMA-a bi eragiketa eraikitzaileren bidez definitzen da: sekuentzia hutsa ($\langle\,\rangle$-z adierazten dena) eta $\bullet$ eragiketa. Eragiketa horrek elementu bat (lehenengo posizioan jarrita) eta beste sekuentzia bat (lehenengo elementuaren ondoren jarrita) elkartuz sekuentzia berria sortzen du. Adibidez, osoak diren lehenengo bost zenbaki positiboen sekuentzia honela

adieraz daiteke eragiketa eraikitzaileak erabiliz:

$$1 \bullet (2 \bullet (3 \bullet (4 \bullet (5 \bullet \langle\,\rangle))))$$

Edo, era laburtuan, honela adierazita:

$$\langle 1,\, 2,\, 3,\, 4,\, 5 \rangle$$

Honako beste eragiketa hauek oso ezagunak dira *sekuentzia*(T) datumotarako:

- *lehena* (lehen elementua itzultzen du),
- *hondarra* (sekuentziaren lehen elementua kenduta gelditzen den sekuentzia itzultzen du),
- *hutsa_da* (sekuentzia hutsa den ala ez esaten du),
- $\in$ (elementua sekuentzian dagoen ala ez esaten du),
- @ (bi sekuentzia kateatzen ditu).

Aipatutako eragiketak dituen *sekuentzia*(T) DMA-ak ondoko espezifikazio ekuazionala du:

Mota: *sekuentzia*(T)
Lag: *bool*
Eragiketak:

$\langle\,\rangle$: $\rightarrow$ *sekuentzia*(T)	(eraikitzailea)
$\_ \bullet \_$: $T \times$ *sekuentzia*$(T) \rightarrow$ *sekuentzia*(T)	(eraikitzailea)
lehena: *sekuentzia*$(T) \rightarrow T$	
hondarra: *sekuentzia*$(T) \rightarrow$ *sekuentzia*(T)	
hutsa_da: *sekuentzia*$(T) \rightarrow$ *bool*	
$\_ \in \_$: $T \times$ *sekuentzia*$(T) \rightarrow$ *bool*	
$\_@\_$: *sekuentzia*$(T) \times$ *sekuentzia*$(T) \rightarrow$ *sekuentzia*(T)	

Ekuazioak:

(s.1)	$lehena(\langle\,\rangle) = \texttt{errorea}$
(s.2)	$lehena(x \bullet s) = x$
(s.3)	$hondarra(\langle\,\rangle) = \texttt{errorea}$
(s.4)	$hondarra(x \bullet s) = s$
(s.5)	$hutsa\_da(\langle\,\rangle) = \mathsf{T}$
(s.6)	$hutsa\_da(x \bullet s) = \mathsf{F}$
(s.7)	$y \in \langle\,\rangle = \mathsf{F}$
(s.8)	$y \in (x \bullet s) = (\ y = x \ \vee\ y \in s\ )$
(s.9)	$\langle\,\rangle @ s2 = s2$
(s.10)	$(x \bullet s1) @ s2 = x \bullet (s1 @ s2)$

Eraikitzaile ez-konstantea den eragiketa bakarraren ($\bullet$ eragiketaren) alderantzizkoak dira *lehena* eta *hondarra* funtzioak, edo beste era batera esanda, funtzio suntsitzaileak dira. Esan bezala, espezifikazio ekuazionalak mota bateko objektuen propietateen gainean arrazoitzeko bidea ematen digu. Horrela, $1 \bullet (2 \bullet (3 \bullet \langle\,\rangle))$ sekuentzia emanda, hau da $\langle 1,\, 2,\, 3\rangle$, honako adierazpen hauek ebalua ditzakegu definitutako ekuazioak aplikatuz:

$$
\begin{aligned}
&\mathit{lehena}(\langle 1,\, 2,\, 3\rangle) = 1\\
&\mathit{hondarra}(\langle 1,\, 2,\, 3\rangle) = \langle 2,\, 3\rangle\\
&\mathit{hutsa\_da}(\langle 1,\, 2,\, 3\rangle) = \mathsf{F}\\
&2 \in \langle 1,\, 2,\, 3\rangle = \mathsf{T}\\
&4 \in \langle 1,\, 2,\, 3\rangle = \mathsf{F}\\
&\langle 1,\, 2,\, 3\rangle @ \langle 4,\, 5\rangle = \langle 1,\, 2,\, 3,\, 4,\, 5\rangle
\end{aligned}
$$

Ondoren, azkeneko adibidearen ebaluazio-urratsak azaltzeari ekingo diogu espezifikazioaren ekuazioak erabiliz. Beste adibide guztiak ariketa gisa gelditzen dira irakurlearentzat.

$$
\begin{aligned}
&(1 \bullet (2 \bullet (3 \bullet \langle\,\rangle)))@(4 \bullet (5 \bullet \langle\,\rangle))\\
&\quad =_{(s.10)} \quad 1 \bullet ((2 \bullet (3 \bullet \langle\,\rangle))@(4 \bullet (5 \bullet \langle\,\rangle)))\\
&\quad =_{(s.10)} \quad 1 \bullet (2 \bullet ((3 \bullet \langle\,\rangle)@(4 \bullet (5 \bullet \langle\,\rangle))))\\
&\quad =_{(s.10)} \quad 1 \bullet (2 \bullet (3 \bullet (\langle\,\rangle@(4 \bullet (5 \bullet \langle\,\rangle)))))\\
&\quad =_{(s.9)} \quad 1 \bullet (2 \bullet (3 \bullet (4 \bullet (5 \bullet \langle\,\rangle))))
\end{aligned}
$$

Orain *sekuentzia*(T) DMA-aren oinarrizko definizioa aberats dezakegu eragiketa berriak gehituz. Eragiketa berri horien definizioak gehitzeak ez dakar inolako aldaketarik jatorrizko espezifikazioan. Adibidez, *luzera*, *txertatu*, *azkena_kendu* eta *ezabatu* eragiketak gehi daitezke. *luzera* eragiketak sekuentzia baten osagai kopurua itzultzen du. *txertatu* eragiketak mota bateko elementu bat, mota bereko sekuentzia bat eta sekuentziako posizio bat adierazten duen zenbaki arrunta emanda, sekuentziako posizio horretan elementua txertatuz lortzen den sekuentzia itzultzen du, jatorrizko sekuentzian posizio horretan zegoen elementua eta haren eskuineko guztiak txertatutako elementu berriaren eskuinaldean geratuko direlarik. *azkena_kendu* eragiketak sekuentzia bateko azken elementua (eskuineko ertzean dagoena) kenduta geratzen den sekuentzia itzultzen du. Azkenik, T motako elementu bat eta T mota bereko osagaiez osatutako sekuentzia bat emanda, sekuentzian elementuak dituen agerpen guztiak ezabatuta lortzen den sekuentzia itzultzen du *ezabatu* eragiketak. Eragiketa horiek honela espezifikatuko lirateke[5]:

5. Hemendik aurrera, *nat* motako objektuentzako ohiko notazioa erabiliko dugu, hau da, 0, 1, +, $*$ eta abar.

Mota: $sekuentzia(T)$
Lag: nat
Eragiketak:
$luzera$: $sekuentzia(T) \to nat$
$txertatu$: $T \times sekuentzia(T) \times nat \to sekuentzia(T)$
$azkena\_kendu$: $sekuentzia(T) \to sekuentzia(T)$
$ezabatu$: $T \times sekuentzia(T) \to sekuentzia(T)$
Ekuazioak:

(s.11) $luzera(\langle\,\rangle) = 0$

(s.12) $luzera(x \bullet s) = 1 + luzera(s)$

(s.13) $$txertatu(x, \langle\,\rangle, pos) = \begin{cases} \texttt{errorea} & \text{baldin } pos \neq 1 \\ x \bullet \langle\,\rangle & \text{bestela} \end{cases}$$

(s.14) $$txertatu(x, y \bullet s, pos) = \begin{cases} \texttt{errorea} & \text{baldin } pos = 0 \\ x \bullet (y \bullet s) & \text{baldin } pos = 1 \\ y \bullet (txertatu(x, s, pos - 1)) & \text{bestela} \end{cases}$$

(s.15) $azkena\_kendu(\langle\,\rangle) = \texttt{errorea}$

(s.16) $$azkena\_kendu(x \bullet s) = \begin{cases} \langle\,\rangle & \text{baldin } hutsa\_da(s) \\ x \bullet azkena\_kendu(s) & \text{bestela} \end{cases}$$

(s.17) $ezabatu(x, \langle\,\rangle) = \langle\,\rangle$

(s.18) $$ezabatu(x, y \bullet s) = \begin{cases} ezabatu(x, s) & \text{baldin } x = y \\ y \bullet ezabatu(x, s) & \text{bestela} \end{cases}$$

Kontuan izan, *txertatu* eragiketaren kasuan, sekuentzia hutsa ez bada eta emandako posizioa sekuentziaren luzera gehi 1 baino handiagoa baldin bada, (s.14) ekuazioaren aplikazio errekurtsiboaren ondorioz sekuentzia berria sortzen joango den arren, errorea sortuko dela (s.13) ekuazioa aplikatzean, bukaeran sekuentzia hutsa eta 1 baino handiagoa izango den hirugarren argumentua izango baititugu. 5.1. atalean aipatu den bezala, erroreak hedatu egiten dira, eta, beraz, termino osoaren ebaluazioa errorea izango da. Honako hau egoera horren adibidea da:

$$\begin{aligned} & txertatu(3, (1 \bullet (2 \bullet \langle\,\rangle)), 8) \\ =_{(s.14)} \quad & 1 \bullet txertatu(3, (2 \bullet \langle\,\rangle), 7) \\ =_{(s.14)} \quad & 1 \bullet (2 \bullet txertatu(3, \langle\,\rangle, 6)) \\ =_{(s.13)} \quad & 1 \bullet (2 \bullet \texttt{errorea}) \\ = \quad & \texttt{errorea} \end{aligned}$$

Beste aukera bat (s.14) ekuazioa berridaztea da, *pos* 0 baldin bada edo *pos* $y \bullet s$ sekuentziaren luzera gehi 1 baino handiagoa baldin bada errorea

sortuz, eta beste bi kasuak zeuden bezala utziz. Aukera hori (s.14)' izendatuz, aurreko adibidearen garapena honela litzateke:

$$txertatu(3,(1 \bullet (2 \bullet \langle\,\rangle))),8) =_{(s.14)'} \texttt{errorea}$$

Aurretik azaldu dugun bezala, T parametroak aukera ematen du sekuentzietarako definitutako eragiketa guztiak mota guztietarako baliagarri izan daitezen. Adibidez, *luzera* eragiketa baliagarria da zenbaki arrunt (*nat*), zenbaki oso (*oso*), boolear (*bool*) eta abarretarako. Baina, batzuetan beharrezkoa izaten da mota bakar baterako zentzua duten eragiketak definitzea. Horrelako kasua da, esaterako, *handiena* eragiketa, zenbaki arruntez osatutako sekuentzia baten baliorik handiena itzultzen duena. Eragiketa horrek ez du zentzurik ordena aurredefiniturik ez duten motetarako. Horregatik, mota generikoa aberastu beharrean mota espezifikoa aberastuko dugu, aipatutako kasuan *sekuentzia*(*nat*) datu-motarako *handiena* eragiketa gehituz:

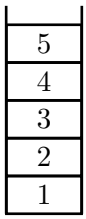
Mota: $sekuentzia(nat)$
Eragiketak:
$handiena$: $sekuentzia(nat) \to nat$
Ekuazioak:
(s.19) $handiena(\langle\,\rangle) = \texttt{errorea}$
(s.20) $handiena(x \bullet s) =$

$$\begin{cases} x & \text{baldin } hutsa\_da(s) \\ handiena(x \bullet hondarra(s)) & \text{baldin } \neg\, hutsa\_da(s) \\ & \quad \wedge\ x > lehena(s) \\ handiena(s) & \text{bestela} \end{cases}$$

5.3.2. Pilak

pila(T) DMA-aren izendapena mundu errealeko objektuez (paperak, platerak, ...) osatutako pilekin duen antzekotasunetik dator. Pilak, sekuentzien antzera, luzera aldakorreko egitura linealak dira. Dena den, portaerari dagokionez, pilek murriztapen gehiago dituzte: kontsultak eta aldaketak egituraren mutur bakar batean egin daitezke. Mutur horri *gailur* deritzo (mundu errealeko pilen antzekotasuna kontuan hartuz, azken horiek bertikalak baitira). Elementuak gailurretik hartzen dira eta gailurrean ipintzen dira, eta pilaren gailurreko elementua besterik ez dago kontsultatzerik. Hortaz, pilan sartzen den azken osagaia izango da irtengo den lehena. Adibidez, 5.1. irudiko pila (bertikalki marraztuta) pila huts batean osoak diren lehenengo bost zenbaki positiboak banan-banan txikienetik handienera sartuz sortu da.

pila(T) DMA-aren eragiketa eraikitzaileak *pilahutsa* eta *pilaratu* dira; eta eragiketa ez-eraikitzaileen arteko ohikoenak *gailurrekoa*, *despilatu* eta *hutsa_da*. Azken horien espezifikazio ekuazionala honako hau da:



5.1. irudia. Pila baten adibidea.

Mota: $pila(T)$
Usa: *bool*
Eragiketak:

$pilahutsa$:	$\rightarrow pila(T)$	(eraikitzailea)
$pilaratu$:	$T \times pila(T) \rightarrow pila(T)$	(eraikitzailea)
$gailurrekoa$:	$pila(T) \rightarrow T$	
$despilatu$:	$pila(T) \rightarrow pila(T)$	
$hutsa\_da$:	$pila(T) \rightarrow bool$	

Ekuazioak:

(p.1) $gailurrekoa(pilahutsa) = \texttt{errorea}$
(p.2) $gailurrekoa(pilaratu(x,p)) = x$
(p.3) $despilatu(pilahutsa) = \texttt{errorea}$
(p.4) $despilatu(pilaratu(x,p)) = p$
(p.5) $hutsa\_da(pilahutsa) = \mathsf{T}$
(p.6) $hutsa\_da(pilaratu(x,p)) = \mathsf{F}$

pilahutsa eta *pilaratu* eragiketa eraikitzaileak erabiliz, 5.1. irudiko pila honela adieraziko litzateke:

$$pilaratu(5,pilaratu(4,pilaratu(3,pilaratu(2,pilaratu(1,pilahutsa)))))$$

gailurrekoa eta *despilatu* eragiketak *pilaratu* eragiketaren alderantzizkoak dira, hau da, eragiketa suntsitzaileak dira.

Ondoren, $pila(T)$ datu-mota aberastuko dugu honako eragiketa hauekin: *altuera*, pila baten osagai kopurua itzultzen du; *hartu*, pila bat eta n zenbaki arrunta emanda gailurretik hasita lortzen diren lehenengo n elementuez osatutako pila itzultzen du, jatorrizko ordena mantenduz; *ezabatu*, pila bat eta n zenbaki arrunta emanda, gailurretik hasita lortzen diren lehenengo n elementuak ezabatuta lortzen den pila itzultzen du; eta, azkenik, *mugitu*, bi pila emanda bigarren pilarekiko gailentzen diren lehenengo pilako elementuak

bigarren pilaren gainean jarrita lortzen den pila itzultzen du (lehenengo pilatik mugitutako elementuak alderantzizko ordenan gelditutako dira irteerako pilan).

Mota: $pila(T)$
Lag: nat
Eragiketak:
$altuera$: $pila(T) \to nat$
$hartu$: $nat \times pila(T) \to pila(T)$
$ezabatu$: $nat \times pila(T) \to pila(T)$
$mugitu$: $pila(T) \times pila(T) \to pila(T)$
Ekuazioak:

(p.7) $altuera(pilahutsa) = 0$

(p.8) $altuera(pilaratu(x,p)) = 1 + altuera(p)$

(p.9) $$hartu(n, pilahutsa) = \begin{cases} \texttt{errorea} & \text{baldin } n \neq 0 \\ pilahutsa & \text{bestela} \end{cases}$$

(p.10) $$hartu(n, pilaratu(x,p)) = \begin{cases} pilahutsa & \text{baldin } n = 0 \\ pilaratu(x, hartu(n-1,p)) & \text{bestela} \end{cases}$$

(p.11) $$ezabatu(n, pilahutsa) = \begin{cases} \texttt{errorea} & \text{baldin } n \neq 0 \\ pilahutsa & \text{bestela} \end{cases}$$

(p.12) $$ezabatu(n, pilaratu(x,p)) = \begin{cases} pilaratu(x,p) & \text{baldin } n = 0 \\ ezabatu(n-1,p) & \text{bestela} \end{cases}$$

(p.13) $mugitu(pilahutsa, q) = q$

(p.14) $$mugitu(pilaratu(x,p), q) = \begin{cases} q & \text{baldin } altuera(q) \geq altuera(pilaratu(x,p)) \\ mugitu(p, pilaratu(x,q)) & \text{bestela} \end{cases}$$

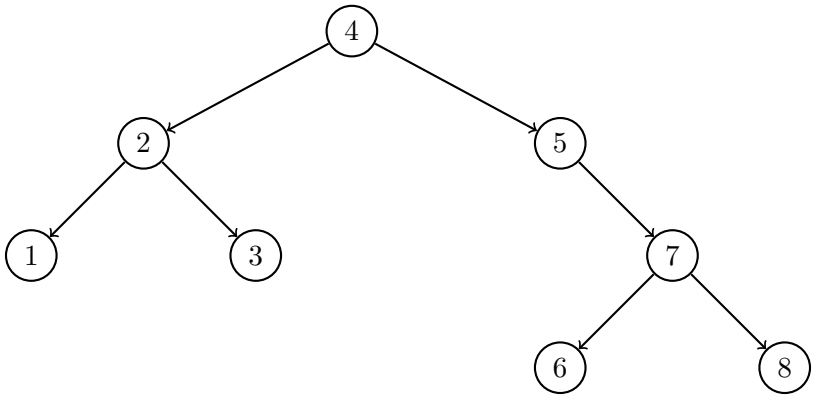
5.3.3. Zuhaitz bitarrak

Sekuentzia eta pilekin gertatzen ez den bezala, *zuhaitzak adabegi* izendatzen diren elementuez osatutako datu-egitura hierarkikoak dira. Adabegien arteko erlazioa ez da lineala (hau da, adabegi bakoitzak ez du gehienez ondorengo adabegi bat), adabegi baten ondorengoak adabegi bat baino gehiago izan daitezke eta, edozein kasutan, bada inoren ondorengoa ez den adabegia zuhaitz ez-huts guztietan, *erro* deritzona, hain zuzen.

*zuhaitz bitar*ren adabegiek gehienez ere bi ondorengo izaten dituzte, posizioaren arabera *ezkerreko* eta *eskuineko* deritzenak. Zuhaitz bitar kontzeptuaren definizioa emateko era egokiena definizio induktiboa (edo errekurtsiboa) dugu.

- Zuhaitz hutsa (adabegirik gabekoa) zuhaitz bitarra da.

- e adabegia eta ezk zein esk zuhaitz bitarrak emanda, e erro gisa eta erroaren bi ondorengo gisa ezk eta esk azpizuhaitzez osatuta dagoen zuhaitza zuhaitz bitarra da.



5.2. irudia. Zuhaitz bitar baten adibidea.

5.2. irudian 4 zenbaki osoa erroaren balioa da, ezkerreko azpizuhaitzaren erroa 2 da, eta eskuinekoarena, 5. Era berean azpizuhaitz bakoitza beste bi azpizuhaitzez osatuta dago, eta, horrela jarraituz zuhaitz osoa errekurtsiboki deskriba daiteke.

Zuhaitz motako datu-egituretan erabilitako terminologia bizitzako elementu batzuekiko antzekotasunean oinarritzen da (zuhaitza landare gisa, genealogikoa eta abar). Adibidez, *hostoa* erabiltzen da azpizuhaitzik ez duen adabegia izendatzeko, eta *adarra* da zuhaitzaren errotik zuhaitzaren hosto bateraino dauden adabegi guztien segida. Adibidez, 5.2. irudiko zuhaitzak lau hosto ditu, zehazki, 1, 3, 6 eta 8 balioak dituztenak. 5 balioa duen adabegiak azpizuhaitz huts bat du, ezkerrekoa, hain zuzen, baina, eskuineko azpizuhaitza hutsa ez denez, ez da hostoa. Adarrei dagokienez, honako hauek

dira:

$$\langle 4,\ 2,\ 1\rangle$$
$$\langle 4,\ 2,\ 3\rangle$$
$$\langle 4,\ 5,\ 7,\ 6\rangle$$
$$\langle 4,\ 5,\ 7,\ 8\rangle$$

Zuhaitzaren *sakonera* adar luzeenaren luzera da. Hortaz, aurreko zuhaitzaren sakonera 4 da, adar luzeenen luzera, hain zuzen. Horretaz gain, zuhaitz baten adabegiak *mailaka* egituratuak balira bezala ikus daitezke, eta *aurrekoa-ondorengoa* erlazioa definitzen da erroaren eta erro baten azpizuhaitzaren erroaren artean. Euskaraz, erlazio hori *guraso-ume* erlazioa izendatzen da, konkretuki, *guraso* adabegia eta *ezkerreko umeaz* zein *eskuineko umeaz* hitz egiten da.

Zuhaitz bitarrak oso baliagarriak dira adierazpen matematikoak (aritmetikoak, boolearrak eta abar), lengoaia naturalaren esaldiak, objektuen sailkapenak, erabaki-diagramak eta abar errepresentatzeko. Eta, gainera, informazioa biltegiratzeko zein eskuratzeko ere erabiltzen dira, zuhaitz-egituretan bilaketak, txertaketak eta ezabaketak egitura linealetan baino era eraginkorragoan egiten baitira. Zuhaitzaren elementuak era egokian antolatuz gero, elementuen bilaketa oso eraginkorra bihur daiteke. Bereziki, *bilaketa dikotomikoa* era naturalean inplementatzen da dikotomikoak eta orekatuak diren zuhaitzetan:

- Zuhaitz bitar bat *dikotomikoa* da, baldin eta elementu bakoitza bere ezkerreko azpizuhaitzaren elementu guztiak baino handiagoa edo berdina bada, eta bere eskuineko azpizuhaitzaren elementu guztiak baino txikiagoa bada.

- Zuhaitz bat orekatuta dago, baldin eta bere adabegi guztien ezkerreko azpizuhaitzaren sakoneraren eta eskuineko azpizuhaitzaren sakoneraren arteko aldea 1 baino txikiagoa edo berdina bada.

Zuhaitz bitarren espezifikazio ekuazionala, jakina, beren definizio errekurtsiboan oinarritzen da. *zuhbit*(T) (T datu-mota generikoa duten elementuz osatutako zuhaitz bitarra) DMA-aren eragiketa eraikitzaileak *hutsa* eta *errotu* dira. Signaturan kontsultarako oinarrizko eragiketak diren *erroa*, *ezker*, *eskuin* eta *hutsa_da* ere gehituko ditugu:

Mota: $zuhbit(T)$
Lag: $bool$
Eragiketak:
$hutsa$: $\rightarrow zuhbit(T)$ (eraikitzailea)
$errotu$: $T \times zuhbit(T) \times zuhbit(T) \rightarrow zuhbit(T)$ (eraikitzailea)
$erroa$: $zuhbit(T) \rightarrow T$
$ezker$: $zuhbit(T) \rightarrow zuhbit(T)$
$eskuin$: $zuhbit(T) \rightarrow zuhbit(T)$
$hutsa\_da$: $zuhbit(T) \rightarrow bool$
Ekuazioak:
(a.1) $erroa(hutsa) = \texttt{errorea}$
(a.2) $erroa(errotu(x, ezk, esk)) = x$
(a.3) $ezker(hutsa) = \texttt{errorea}$
(a.4) $ezker(errotu(x, ezk, esk)) = ezk$
(a.5) $eskuin(hutsa) = \texttt{errorea}$
(a.6) $eskuin(errotu(x, ezk, esk)) = esk$
(a.7) $hutsa\_da(hutsa) = \mathsf{T}$
(a.8) $hutsa\_da(errotu(x, ezk, esk)) = \mathsf{F}$

5.2. irudian 5 erroa duen azpizuhaitza hartzen badugu, *hutsa* eta *errotu* eragiketa eraikitzaileak erabilita honako errepresentazioa lortzen da:

$$errotu(5, hutsa, errotu(7, errotu(6, hutsa, hutsa), errotu(8, hutsa, hutsa)))$$

Funtzio suntsitzaileak (*errotu* eragiketaren alderantzizkoak) *erroa*, *ezker* eta *eskuin* dira.

Zuhaitz bitarren gaineko bestelako kalkulu ohikoak dira zuhaitzaren sakonera (*sakonera*), adabegien kopurua (*adabegikop*), elementu bat zuhaitzean dagoen ala ez (*dago*), zenbaki arrunt batek zehaztutako sakonera baino beherago dauden adabegi guztiak ezabatzea (*inausi*), edo zuhaitzean ezkerren dagoen adarra lortzea (*ezker_adar*). Jatorrizko espezifikazio ekuazionala eragiketa berri horiekin aberasteko asmoz, honako ekuazio hauek gehituko genituzke:

Mota: $zuhbit(T)$
Lag: $bool$, nat, $sekuentzia(T)$
Eragiketak:
$sakonera$: $zuhbit(T) \rightarrow nat$
$adabegikop$: $zuhbit(T) \rightarrow nat$
$dago$: $T \times zuhbit(T) \rightarrow bool$
$inausi$: $nat \times zuhbit(T) \rightarrow zuhbit(T)$
$ezker\_adar$: $zuhbit(T) \rightarrow sekuentzia(T)$

Ekuazioak:

(a.9) $sakonera(hutsa) = 0$

(a.10) $sakonera(errotu(x, ezk, esk)) = 1 + handiena(sakonera(ezk), sakonera(esk))$

(a.11) $adabegikop(hutsa) = 0$

(a.12) $adabegikop(errotu(x, ezk, esk)) = 1 + adabegikop(ezk) + adabegikop(esk)$

(a.13) $dago(x, hutsa) = \mathsf{F}$

(a.14) $$dago(x, errotu(y, ezk, esk)) = \begin{cases} \mathsf{T} & \text{baldin } x = y \\ dago(x, ezk) \vee dago(x, esk) & \text{bestela} \end{cases}$$

(a.15) $inausi(n, hutsa) = hutsa$

(a.16) $$inausi(n, errotu(x, ezk, esk)) = \begin{cases} hutsa & \text{baldin } n = 0 \\ errotu(x, inausi(n-1, ezk), & \text{bestela} \\ \qquad inausi(n-1, esk)) & \end{cases}$$

(a.17) $ezker\_adar(hutsa) = \langle\,\rangle$

(a.18) $$ezker\_adar(errotu(x, ezk, esk)) = \begin{cases} x \bullet ezker\_adar(ezk) & \text{baldin } \neg\, hutsa\_da(ezk) \\ x \bullet ezker\_adar(esk) & \text{bestela} \end{cases}$$

5.4. ARIKETAK: SEKUENTZIAK

1. Osatu (____) hutsuneak sekuentzia baten alderantzizkoa kalkulatzen duen funtzioaren honako espezifikazio ekuazional honetan:

   Mota: $sekuentzia(T)$
   Lag: (ez dago)
   Eragiketa:
   $alderantzizkoa$: $sekuentzia(T) \to sekuentzia(T)$
   Ekuazioak:
   (1) $alderantzizkoa(\_\_\_\_\_) = \langle\,\rangle$
   (2) $alderantzizkoa(\_\_\_\_\_) = alderantzizkoa(s) @ \langle x \rangle$

2. Aztertu *aurrizkia* funtzioa definitzeko ematen diren espezifikazio ekuazionalak eta arrazoitu espezifikazio bakoitza zuzena den ala ez:

   aurrizkia funtzioak, s sekuentzia eta n zenbaki arrunta emanda, s-ko hasierako n elementuz osatutako azpisekuentzia itzuliko du. s sekuentziak n elementu ez badauzka, orduan s sekuentzia itzuliko du.

Adibidez:

- $aurrizkia(\langle 4,4,2,1,1,8\rangle,3) = \langle 4,4,2\rangle$
- $aurrizkia(\langle 4,4,2,1,1,8\rangle,9) = \langle 4,4,2,1,1,8\rangle$

Mota: $sekuentzia(T)$
Lag: nat
Eragiketa:
$aurrizkia$: $sekuentzia(T) \times nat \to sekuentzia(T)$

2.1. Ekuazioak:
(1) $aurrizkia(s,0) = \langle\,\rangle$
(2) $aurrizkia(x \bullet s, n+1) = x \bullet aurrizkia(s,n)$

2.2. Ekuazioak:
(1) $aurrizkia(s,0) = \langle\,\rangle$
(2) $aurrizkia(\langle\,\rangle, n+1) = \langle\,\rangle$
(3) $aurrizkia(x \bullet s, n+1) = x \bullet aurrizkia(s,n)$

2.3. Ekuazioak:
(1) $aurrizkia(\langle\,\rangle, 0) = \langle\,\rangle$
(2) $aurrizkia(\langle\,\rangle, n) = \langle\,\rangle$
(3) $aurrizkia(x \bullet s, n) = x \bullet aurrizkia(s, n-1)$

5.5. ARIKETAK: PILAK

1. Osatu (_) hutsuneak, pila bat emanda, lehenengo osagaitzat pilaren gailurra duen eta jarraian pilako beste osagaiak goitik beherako ordena berean dituen sekuentzia itzultzen duen funtzioaren honako espezifikazio ekuazional honetan:

Mota: $pila(T)$
Lag: $sekuentzia(T)$
Eragiketa:
$etzan$: $pila(T) \to sekuentzia(T)$
Ekuazioak:
(1) $etzan(\_\_\_\_\_) = \langle\,\rangle$
(2) $etzan(\_\_\_\_\_) = (\_\_) \bullet etzan(\_\_\_\_\_)$

2. Aztertu *agerpen_pos* funtzioa definitzeko ematen diren espezifikazio ekuazionalak eta arrazoitu espezifikazio bakoitza zuzena den ala ez:

> p pila bat, x elementu bat eta n zenbaki positiboa emanda, *agerpen_pos* funtzioak p-ren n-garren posizioan (gailurretik behera kontatuta) x-a agertzen den ala ez erabaki behar du.

$(p = $  $\wedge\, x = 6 \wedge n = 3) \rightarrow agerpen\_pos(p,x,n) = \mathsf{T}$

Mota: $pila(T)$
Lag: nat, $bool$
Eragiketa:
$agerpen\_pos : pila(T) \times T \times nat \rightarrow bool$

2.1. Ekuazioak:
(1) $agerpen\_pos(pilahutsa,x,n) = \texttt{errorea}$
(2) $agerpen\_pos(pilaratu(y,p),x,0) = \texttt{errorea}$
(3) $agerpen\_pos(pilaratu(y,p),x,n+1) =$
$$\begin{cases} y = x & \text{baldin } n = 0 \\ agerpen\_pos(p,x,n) & \text{bestela} \end{cases}$$

2.2. Ekuazioak:
(1) $agerpen\_pos(pilahutsa,x,n) = \texttt{errorea}$
(2) $agerpen\_pos(pilaratu(y,p),x,0) = \texttt{errorea}$
(3) $agerpen\_pos(pilaratu(y,p),x,n) =$
$$\begin{cases} \mathsf{T} & \text{baldin } y = x \wedge n = 1 \\ \mathsf{F} & \text{baldin } y \neq x \wedge n = 1 \\ agerpen\_pos(p,x,n-1) & \text{bestela} \end{cases}$$

2.3. Ekuazioak:
(1) $agerpen\_pos(pilahutsa,x,n) = \mathsf{F}$
(2) $agerpen\_pos(pilaratu(y,p),x,0) = \mathsf{F}$
(3) $agerpen\_pos(pilaratu(y,p),x,n+1) =$
$$\begin{cases} y = x & \text{baldin } n = 0 \\ agerpen\_pos(p,x,n) & \text{bestela} \end{cases}$$

5.6. ARIKETAK: ZUHAITZ BITARRAK

1. Osatu (____) hutsuneak a zuhaitz bitarraren hosto kopurua kalkulatzen duen funtzioaren espezifikazio ekuazionalean.

Mota: $zuhbit(T)$
Lag: nat
Eragiketa:
$hostokop$: $zuhbit(T) \to nat$
Ekuazioak:
(1) $hostokop(\_\_\_\_\_) = 0$
(2) $hostokop(\_\_\_\_\_) =$

$$\begin{cases} 1 & \text{baldin } \_\_\_\_\_ \\ hostokop(ezk) + hostokop(esk) & \text{bestela} \end{cases}$$

2. Arrazoitu honako espezifikazio hau zergatik ez den zuzena a zuhaitz bitarraren adabegi kopurua kalkulatzeko.

Mota: $zuhbit(T)$
Lag: nat
Eragiketa:
$adabegikop$: $zuhbit(T) \to nat$
Ekuazioak:
(1) $adabegikop(hutsa) = 0$
(2) $adabegikop(errotu(x, ezk, esk)) =$
$adabegikop(ezk) + adabegikop(esk)$

5.7. PROPIETATEEN FROGAPENA

DMAen espezifikazio ekuazionala adierazpenak ebaluatzeko eta propietateak era arrazoituan frogatzeko oinarri formala da. Bai dedukziozko bai indukziozko arrazoibidea erabil daiteke espezifikazio ekuazionalekin. Propietate batzuen frogapenak dedukzio bidezkoak izan daitezke, baina, gehienak indukzio bidezkoak dira.

5.7.1. Dedukzio bidezko frogapenak

Dedukzio bidezko frogapen ekuazionalak adierazpenen berridazketan oinarritzen dira. Gehienetan, bi adierazpen baliokideak direla frogatzen da adierazpen horiek berridatziz edo sinplifikatuz. Berridazketaren urrats bakoitzean,

espezifikazioaren ekuazioren bat edo aurretik frogatutako propietateren bat erabili behar da.

Adibide gisa, $pila(T)$ DMA-ari buruzko hurrengo propietatea frogatuko dugu[6]:

$$altuera(despilatu(pilaratu(x,pilaratu(y,pilahutsa)))) = 1$$

Arrazoibide deduktiboari jarraituz eta $pila(T)$ DMA-aren ekuazioak erabiliz, frogapena honako hau izango litzateke:

$$\begin{array}{ll} & altuera(despilatu(pilaratu(x,pilaratu(y,pilahutsa)))) \\ =_{(p.4)} & altuera(pilaratu(y,pilahutsa)) \\ =_{(p.8)} & altuera(pilahutsa)+1 \\ =_{(p.7)} & 0+1 \\ =_{(n.14)} & 1 \end{array}$$

Lehenengo urratsean (p.4) ekuazioa ($despilatu(pilaratu(x,p)) = p$) erabili dugu, bigarren urratsean (p.8) ekuazioa ($altuera(pilaratu(x,p)) = 1 + altuera(p)$) eta hirugarren urratsean (p.7) ekuazioa ($altuera(pilahutsa) = 0$). Azkenik, frogapena (n.14) axiomarekin ($zero + y = y$) bukatzen da.

Beste propietate hau ere arrazoibide deduktiboa erabiliz froga daiteke[7]:

$$(\neg\, hutsa\_da(p)) \rightarrow (p = pilaratu(gailurrekoa(p), despilatu(p)))$$

$pila(T)$ DMA-aren definizioa kontuan hartuz,

$$(\neg\, hutsa\_da(p)) \rightarrow (\exists x\, \exists q\, (p = pilaratu(x,q)))$$

propietatea betetzen dela ziurta dezakegu. Propietate hori honako frogapen honetako lehenengo eta azkeneko urratsetan erabili da:

$$\begin{array}{ll} & pilaratu(gailurrekoa(p), despilatu(p)) \\ = & pilaratu(gailurrekoa(pilaratu(x,q)), despilatu(pilaratu(x,q))) \\ =_{(p.2)} & pilaratu(x, despilatu(pilaratu(x,q))) \\ =_{(p.4)} & pilaratu(x,q) \\ = & p \end{array}$$

6. Kontuan izan ekuazioetan agertzen diren aldagai guztiak unibertsalki kuantifikatutzat hartzen direla.

7. Edozein p: $pila(T)$-rako.

5.7.2. Ariketak: Dedukzioa

Frogatu formalki honako propietate hauek[8]:

1. s hutsa ez den edozein sekuentzia izanda:

$$\{\, s = lehena(s) \bullet hondarra(s) \,\}$$

2. a hutsa ez den edozein zuhaitz bitar izanda:

$$\{\, a = errotu(erroa(a), ezker(a), eskuin(a)) \,\}$$

5.7.3. Indukzio bidezko frogapenak

$sekuentzia(T)$, $pila(T)$ eta $zuhbit(T)$ DMAen espezifikazioak definizio induktiboz osatzen dira, eragiketa eraikitzaileek kasu nabariaren eta kasu induktiboaren rola hartzen dutelarik. Horregatik, aipatutako DMAen gaineko propietate interesgarrienak frogatu ahal izateko indukzioaren erabilera behar izaten da.

Jarraian, propietate induktiboen adibide desberdinak aurkeztuko dira, hau da, frogatuak izateko egiturazko indukzioaren printzipioa erabiltzea beharrezkoa deneko adibideak emango dira. Lehenengo, zenbaki arrunten gaineko batuketaren elkartze-propietatea frogatuko dugu, 5.2. atalean emandako zenbaki arrunten espezifikazioa erabiliz. Gero, batuketaren trukatze-propietatea frogatuko dugu, baina, ikusiko denez frogapen horrek bi propietate induktibo laguntzaile behar ditu eta azken horien frogapenak ere emango ditugu. Laburbilduz, zenbaki arrunten gaineko lau propietateren frogapen induktiboak aurkeztuko ditugu. Ondoren, sekuentzien gaineko hiru propietate frogatuko ditugu. Lehenengoan batuketaren elkarkortasuna erabili beharko da. Eta, azkenik, zuhaitz bitarren gaineko propietate bat frogatuko dugu.

Zenbaki arrunten batuketarentzat elkartze-propietatea betetzen dela frogatu ahal izateko honako hau frogatuko dugu: n_1, n_2, n_3 edozein zenbaki arrunt izanda,

$$(n_1 + n_2) + n_3 = n_1 + (n_2 + n_3)$$

n_1-en gaineko indukziozko frogapena egingo dugu. Propietatea (+.E)

8. Propietatean parte hartzen duten elementuek mota egokia dutela suposatzen da.

izendatuko dugu[9]. Kasu nabarian $n_1 = zero$ izango da:

$$\begin{array}{ll} (n_1 + n_2) + n_3 & \\ \quad = & (zero + n_2) + n_3 \\ \quad =_{(n.14)} & n_2 + n_3 \\ \quad =_{(n.14)} & zero + (n_2 + n_3) \\ \quad = & n_1 + (n_2 + n_3) \end{array}$$

Kasu induktiboan $n_1 = hurrengoa(x)$ izango da eta x-ren gaineko honako indukzio-hipotesi hau (IH) erabiliko dugu:

$$(x + n_2) + n_3 = x + (n_2 + n_3)$$

Kasu induktiborako frogapena honako hau da:

$$\begin{array}{ll} (n_1 + n_2) + n_3 & \\ \quad = & (hurrengoa(x) + n_2) + n_3 \\ \quad =_{(n.15)} & hurrengoa(x + n_2) + n_3 \\ \quad =_{(n.15)} & hurrengoa((x + n_2) + n_3) \\ \quad =_{(IH)} & hurrengoa(x + (n_2 + n_3)) \\ \quad =_{(n.15)} & hurrengoa(x) + (n_2 + n_3) \\ \quad = & n_1 + (n_2 + n_3) \end{array}$$

Propietate bat era induktiboan frogatu ahal izateko ohikoa izaten da aldez aurretik beste propietate bat frogatu behar izatea. Hori gertatzen da, esaterako, batuketaren trukatze-propietatea frogatzean. Zehazki, n_1 eta n_2 edozein zenbaki arrunt izanda, $n_1 + n_2 = n_2 + n_1$ betetzen dela frogatzeko honako bi propietate hauen frogapena beharrezkoa da aldez aurretik: $n + zero = n$ edozein n zenbaki arruntetarako eta $hurrengoa(n_1) + n_2 = n_1 + hurrengoa(n_2)$ edozein n_1 eta n_2 zenbaki arruntetarako. Azken bi propietate horiek $(+.0)$ eta $(+.S)$ izendatuko ditugu hurrenez hurren. $n + zero = n$ frogatzeko kasu nabarian $n = zero$ dela kontsideratu behar dugu:

$$\begin{array}{ll} n + zero & \\ \quad = & zero + zero \\ \quad =_{(n.14)} & zero \\ \quad = & n \end{array}$$

Kasu induktiboan $n = hurrengoa(x)$ dela kontsideratuko dugu eta x-ren gaineko honako indukzio-hipotesi hau (IH) erabiliko dugu:

$$x + zero = x$$

9. Hemendik aurrera batuketaren propietateak izendatzeko (+.k) erabiliko dugu, k karakterea propietate bakoitzerako karaktere bereizle bat izango delarik.

Kasu induktiboaren frogapena honako hau litzateke:

$$\begin{array}{ll} n + zero & \\ \quad = & hurrengoa(x) + zero \\ \quad =_{(n.15)} & hurrengoa(x + zero) \\ \quad =_{(IH)} & hurrengoa(x) \\ \quad = & n \end{array}$$

$hurrengoa(n_1) + n_2 = n_1 + hurrengoa(n_2)$ betetzen dela frogatzeko, kasu nabarian $n_1 = zero$ dela kontsideratuko dugu:

$$\begin{array}{ll} hurrengoa(n_1) + n_2 & \\ \quad = & hurrengoa(zero) + n_2 \\ \quad =_{(n.15)} & hurrengoa(zero + n_2) \\ \quad =_{(n.14)} & hurrengoa(n_2) \\ \quad =_{(n.14)} & zero + hurrengoa(n_2) \\ \quad = & n_1 + hurrengoa(n_2) \end{array}$$

Kasu induktiboan $n_1 = hurrengoa(x)$ dela kontsideratuko dugu eta x-ren gaineko honako indukzio-hipotesi hau (IH) erabiliko dugu:

$$hurrengoa(x) + n_2 = x + hurrengoa(n_2)$$

Kasu induktiborako frogapena honako hau da:

$$\begin{array}{ll} hurrengoa(n_1) + n_2 & \\ \quad = & hurrengoa(hurrengoa(x)) + n_2 \\ \quad =_{(n.15)} & hurrengoa(hurrengoa(x) + n_2) \\ \quad =_{(IH)} & hurrengoa(x + hurrengoa(n_2)) \\ \quad =_{(n.15)} & hurrengoa(x) + hurrengoa(n_2) \\ \quad = & n_1 + hurrengoa(n_2) \end{array}$$

Ondoren, edozein n_1 eta n_2 zenbaki arruntetarako batuketaren trukakortasuna frogatzeari ekingo diogu n_1-en gaineko indukzioa aplikatuz:

$$n_1 + n_2 = n_2 + n_1$$

(+.T) izendatuko dugu propietatea. Kasu nabaria $n_1 = zero$ denean daukagu:

$$\begin{array}{ll} n_1 + n_2 & \\ \quad = & zero + n_2 \\ \quad =_{(n.14)} & n_2 \\ \quad =_{(+.0)} & n_2 + zero \\ \quad = & n_2 + n_1 \end{array}$$

Kasu induktiboan $n_1 = hurrengoa(x)$ dela kontsideratuko dugu eta x-ren gaineko honako indukzio-hipotesi hau (IH) erabiliko dugu:

$$x + n_2 = n_2 + x$$

Kasu induktiborako frogapena honako hau da:

$$\begin{array}{ll} n_1 + n_2 & \\ \quad = & hurrengoa(x) + n_2 \\ \quad =_{(n.15)} & hurrengoa(x + n_2) \\ \quad =_{(IH)} & hurrengoa(n_2 + x) \\ \quad =_{(n.15)} & hurrengoa(n_2) + x \\ \quad =_{(+.T)} & n_2 + hurrengoa(x) \\ \quad = & n_2 + n_1 \end{array}$$

Ondoren, indukzio bidezko frogapena erabiliko dugu edozein s_1 eta s_2 sekuentziatarako honako hau betetzen dela frogatzeko:

$$luzera(s_1 @ s_2) = luzera(s_1) + luzera(s_2)$$

Kasu induktiboaren frogapenean aurretik frogatutako (+.E) propietatea erabiliko da. @ eragiketa lehenengo sekuentziaren gainean indukzioa aplikatuz definitzen denez, frogapenean s_1-en gaineko indukzioa aplikatuko dugu. Kasu nabarian $s_1 = \langle\,\rangle$ izango da:

$$\begin{array}{ll} luzera(s_1 @ s_2) & \\ \quad = & luzera(\langle\,\rangle @ s_2) \\ \quad =_{(s.9)} & luzera(s_2) \\ \quad =_{(n.14)} & 0 + luzera(s_2) \\ \quad =_{(s.11)} & luzera(\langle\,\rangle) + luzera(s_2) \\ \quad = & luzera(s_1) + luzera(s_2) \end{array}$$

Kasu induktiboan $s_1 = x \bullet r$ beteko da, x: T eta r: $sekuentzia(T)$ izanda. Indukzio-hipotesia (IH) honako hau da:

$$luzera(r @ s_2) = luzera(r) + luzera(s_2)$$

Eta kasu induktiboaren frogapena honako hau da:

$$\begin{array}{ll} luzera(s_1 @ s_2) & \\ \quad = & luzera((x \bullet r) @ s_2) \\ \quad =_{(s.10)} & luzera(x \bullet (r @ s_2)) \\ \quad =_{(s.12)} & 1 + luzera(r @ s_2) \\ \quad =_{(IH)} & 1 + (luzera(r) + luzera(s_2)) \\ \quad =_{(+.E)} & (1 + luzera(r)) + luzera(s_2) \\ \quad =_{(s.12)} & luzera(x \bullet r) + luzera(s_2) \\ \quad = & luzera(s_1) + luzera(s_2) \end{array}$$

Hor, (+.E) zenbaki arrunten gainean aurretik indukzioz frogatutako elkartze-propietatea da.

5.3.1. atalean sekuentzietarako definitu diren eragiketetan ikus daitekeen bezala, eragiketa batzuek errorea sortzen dute sekuentzia hutsen kasuan. Hori gertatzen da, esaterako, *azkena_kendu* eta *handiena* eragiketetan. Eragiketa horiek erabiltzera eramaten gaituzten propietateak frogatzean, kasu nabarian elementu bakarreko sekuentzia kontsideratu behar da, eta kasu induktiboan gutxienez bi elementu dituen sekuentzia aurreikusi behar da. Ondoren, s hutsa ez den edozein sekuentzia izanik,

$$luzera(s) = 1 + luzera(azkena\_kendu(s))$$

propietatea betetzen duela frogatuko dugu indukzioz: Kasu nabarian $s = x \bullet \langle\,\rangle$ izango da:

$$\begin{array}{ll} luzera(s) & \\ = & luzera(x \bullet \langle\,\rangle) \\ =_{(s.12)} & 1 + luzera(\langle\,\rangle) \\ =_{(s.16)} & 1 + luzera(azkena\_kendu(x \bullet \langle\,\rangle)) \\ = & 1 + luzera(azkena\_kendu(s)) \end{array}$$

Kasu induktiboan, $s = x \bullet r$ izango da, x: T eta r: $sekuentzia(T)$ izanda, eta, gainera, r sekuentzia ez-hutsa izanda. Indukzio-hipotesia (IH) honako hau da:

$$luzera(r) = 1 + luzera(azkena\_kendu(r))$$

Eta kasu induktiboaren frogapena honako hau:

$$\begin{array}{ll} luzera(s) & \\ = & luzera(x \bullet r) \\ =_{(s.12)} & 1 + luzera(r) \\ =_{(IH)} & 1 + (1 + luzera(azkena\_kendu(r))) \\ =_{(s.12)} & 1 + luzera(x \bullet azkena\_kendu(r)) \\ =_{(s.16)} & 1 + luzera(azkena\_kendu(x \bullet r)) \\ = & 1 + luzera(azkena\_kendu(s)) \end{array}$$

Kontuan izan funtsezkoa dela kasu induktiboan r sekuentzia hutsa ez izatea, horrek bermatzen baitu *azkena_kendu* eragiketak ez duela errorerik sortuko.

Kasu nabari bat baino gehiago edo kasu induktibo bat baino gehiago duten eragiketetan propietateak frogatzean kasuistika desberdinak eduki behar dira kontuan. Ondoren, edozein s sekuentziatarako

$$ezabatu(x, ezabatu(x, s)) = ezabatu(x, s)$$

betetzen dela indukzioz frogatuko dugu. Kasu nabarian, $s = \langle\,\rangle$ izango da:

$$\begin{array}{ll} & ezabatu(x, ezabatu(x, s)) \\ = & ezabatu(x, ezabatu(x, \langle\,\rangle)) \\ =_{(s.17)} & ezabatu(x, \langle\,\rangle) \\ = & ezabatu(x, s) \end{array}$$

Kasu induktiboan $s = y \bullet r$ izango da, y: T eta r: $sekuentzia(T)$ izanda. Indukzio-hipotesia (IH) honako hau da:

$$ezabatu(x, ezabatu(x, r)) = ezabatu(x, r)$$

Kasu induktiboaren frogapena honako hau da:

$$\begin{array}{ll} & ezabatu(x, ezabatu(x, s)) \\ = & ezabatu(x, ezabatu(x, y \bullet r)) \\ =_{(s.18)} & \begin{cases} ezabatu(x, ezabatu(x, r)) & \text{baldin } x = y \\ ezabatu(x, y \bullet ezabatu(x, r)) & \text{bestela} \end{cases} \\ =_{(s.18)} & \begin{cases} ezabatu(x, ezabatu(x, r)) & \text{baldin } x = y \\ y \bullet ezabatu(x, ezabatu(x, r)) & \text{bestela} \end{cases} \\ =_{(IH)} & \begin{cases} ezabatu(x, r) & \text{baldin } x = y \\ y \bullet ezabatu(x, r) & \text{bestela} \end{cases} \\ =_{(s.18)} & ezabatu(x, y \bullet r) \\ = & ezabatu(x, s) \end{array}$$

(s.18) ekuazioaren bigarren erabilerak *bestela* kasuan besterik ez du eraginik; $x = y$ kasuak, aldiz, (s.18) ekuazioa aplikatzen ez zaionez, aldaketarik ez du jasaten.

Sekuentziak eta pilak ez bezala, zuhaitz bitarrek datu-egitura ez-linealak errepresentatzen dituzte. Horregatik, ohikoa da zuhaitzen gaineko eragiketek errekurtsibitate anizkoitza izatea, konkretuki, errekurtsibitate bikoitza (5.3.3. atalean definitutako *sakonera*, *adabegikop*, *dago* eta *inausi* eragiketetan bezala) edo azpizuhaitz bakoitzeko kasu errekurtsibo bana (5.3.3. atalean definitutako *ezker_adar* eragiketan bezala). Era horretako eragiketak dituzten indukzio bidezko frogapenetan, oro har, bi indukzio-hipotesi behar izaten dira. Ondoren, indukzioz frogatuko dugu edozein n zenbaki arruntetarako eta edozein a zuhaitz bitarretarako honako propietate hau betetzen dela:

$$sakonera(inausi(n, a)) \leq n = \mathsf{T}$$

sakonera eta *inausi* 5.3.3. atalean zuhaitz bitarren gainean espezifikatutako

bi eragiketa dira. Kasu nabarian $a = hutsa$ izango da:

$$\begin{array}{ll} & sakonera(inausi(n,a)) \leq n \\ = & sakonera(inausi(n,hutsa)) \leq n \\ =_{(a.15)} & sakonera(hutsa) \leq n \\ =_{(a.9)} & 0 \leq n \\ =_{(n.5)} & \mathsf{T} \end{array}$$

Kasu induktiboan, $a = errotu(x, ezk, esk)$ izango da, x: T eta ezk, esk: $zuhbit(T)$ izanda. (IH1) eta (IH2) indukzio-hipotesiak honako hauek dira hurrenez hurren:

$$\begin{array}{l} sakonera(inausi(m, ezk)) \leq m = \mathsf{T} \\ sakonera(inausi(m, esk)) \leq m = \mathsf{T} \end{array}$$

Kasu induktiboaren frogapena honako hau da:

$$\begin{array}{ll} & sakonera(inausi(n,a)) \leq n \\ = & sakonera(inausi(n, errotu(x, ezk, esk))) \leq n \\ & \left\{ \begin{array}{ll} sakonera(hutsa) \leq n & \text{baldin } n = 0 \\ sakonera(errotu(x, inausi(n-1, ezk), & \text{bestela} \\ \qquad inausi(n-1, esk))) \leq n & \end{array} \right. \\ =_{(a.16)} & \left\{ \begin{array}{ll} 0 \leq n & \text{baldin } n = 0 \\ sakonera(errotu(x, inausi(n-1, ezk), & \text{bestela} \\ \qquad inausi(n-1, esk))) \leq n & \end{array} \right. \\ =_{(a.9)} & \left\{ \begin{array}{ll} \mathsf{T} & \text{baldin } n = 0 \\ sakonera(errotu(x, inausi(n-1, ezk), & \text{bestela} \\ \qquad inausi(n-1, esk))) \leq n & \end{array} \right. \\ =_{(n.5)} & \left\{ \begin{array}{ll} \mathsf{T} & \text{baldin } n = 0 \\ 1 + handiena(& \text{bestela} \\ \qquad sakonera(inausi(n-1, ezk)), & \\ \qquad sakonera(inausi(n-1, esk))) \leq n & \end{array} \right. \\ =_{(a.10)} & \left\{ \begin{array}{ll} \mathsf{T} & \text{baldin } n = 0 \\ 1 + sakonera(inausi(n-1, ezk)) \leq n & \\ \quad \text{baldin } n > 0 \wedge & \\ \qquad (sakonera(inausi(n-1, esk)) \leq & \\ \qquad\quad sakonera(inausi(n-1, ezk))) & \\ 1 + sakonera(inausi(n-1, esk)) \leq n & \text{bestela} \end{array} \right. \\ =_{(n.18)} & \left\{ \begin{array}{ll} \mathsf{T} & \text{baldin } n = 0 \\ sakonera(inausi(n-1, ezk)) \leq n-1 & \\ \quad \text{baldin } n > 0 \wedge & \\ \qquad (sakonera(inausi(n-1, esk)) \leq & \\ \qquad\quad sakonera(inausi(n-1, ezk))) & \\ sakonera(inausi(n-1, esk)) \leq n-1 & \text{bestela} \end{array} \right. \\ =_{(n.7)} & \end{array}$$

$$
\begin{array}{ll}
 & \left\{\begin{array}{ll}
\mathsf{T} & \text{baldin } n=0 \\
\mathsf{T} & \\
\quad \text{baldin } n>0 \wedge & \\
\quad\quad (\ sakonera(inausi(n-1,esk)) \leq & \\
\quad\quad\ \ sakonera(inausi(n-1,ezk))\ ) & \\
sakonera(inausi(n-1,esk)) \leq n-1 & \text{bestela}
\end{array}\right. \\
=_{(IH1)} & \\
 & \left\{\begin{array}{ll}
\mathsf{T} & \text{baldin } n=0 \\
\mathsf{T} \quad \text{baldin } n>0 \wedge & \\
\quad\quad (\ sakonera(inausi(n-1,esk)) \leq & \\
\quad\quad\ \ sakonera(inausi(n-1,ezk))\ ) & \\
\mathsf{T} & \text{bestela}
\end{array}\right. \\
=_{(IH2)} & \\
= \quad \mathsf{T} &
\end{array}
$$

Frogapen horretan, *nat* DMAren espezifikazio ekuazionalean emandako (n.7) $hurrengoa(x) \leq hurrengoa(y) = x \leq y$ ekuazioa aplikatzean $1 + (sakonera(inausi(n-1,ezk))$ eta $1 + (sakonera(inausi(n-1,esk))$ terminoek, hurrenez hurren, $hurrengoa(sakonera(inausi(n-1,ezk))$ terminoa eta $hurrengoa(sakonera(inausi(n-1,esk))$ terminoa errepresentatzen dituztela kontsideratzen da. Era berean $n-1$ terminoak $aurrekoa(n)$ errepresentatzen du. Eta indukzio-hipotesiak $n-1$ terminoaren gainean egin dira.

5.7.4. Ariketak: Indukzioa

Frogatu formalki honako propietate hauek[10]:

1. p pila q pilaren gainean iraulita lortzen den pilaren altuera bi pila horien altueren batura da.

2. p eta q edozein pila izanda, p pila q-ren gainean iraultzearen ondorioz lortzen den pilaren altuera eta p pila q-ren gainean gainjartzearen ondorioz lortzen den pilaren altuera berdinak dira.

3. Edozein a zuhaitz bitarren adabegi kopurua $2^{sakonera(a)} - 1$ da gehienez (hor, $sakonera(a)$ espresioak a zuhaitzaren sakonera adierazten du).

5.8. BIBLIOGRAFIA-OHARRAK

Espezifikazio aljebraikoa programen edo programa-zatien portaera deskribatzeko formalismo gisa sortu zen, egitura aljebraikoak edo aljebrak aztertzeaz arduratzen ziren *Aljebra Unibertsala* eta *Aljebra Modernoan* [20] erabiltzen

10. Propietate bakoitzean parte hartzen duten elementuek mota egokia dutela suposatu behar da.

ziren logika eta metodo formaletan oinarrituz. 1971n IBMren T. J. Watson Research Center-eko Mathematical Sciences Department-ean ADJ (J. Goguen, J. Thatcher, E. Wagner eta J. Wright) taldea sortu zen datu-mota abstraktuen espezifikazio aljebraikorako oinarriak garatzeko asmoz. ADJren hasierako argitalpenen artean [44] azpimarratu behar da, gehien aipatu dena izan baita, eta erreferentziei dagokienez osatuena delako. ADJ taldeak aitortu izan du [63] eta [49] argitalpenek beren lanean izan zuten eragina. Bi lan horiek arlo honetako hazitzat hartzen dira.

1985ean argitaratutako [37] liburua liburu aitzindaria da espezifikazio aljebraikoaren oinarrien definizioan. Egileak, H. Ehrig eta B. Mahr, alor honetan oso emankorra den eta gaur egun jardunean dagoen Technical University of Berlin-eko taldekoak dira.

1979an, J. Goguen-ek OBJ [43] sortu zuen, espezifikazio aljebraikoen exekuzioan oinarritutako lehenengo programazio-lengoaia. OBJ lengoaiak lengoaia-familia bat sortzeko bidea ireki du, lengoaia horien artean OBJ3 [45], Maude [26], CafeOBJ [30] eta CASL [74] nabarmentzen direlarik; azken horien bilakaera aurrera doa eta oso interesgarriak diren aplikazio praktikoak sortzen ari dira.

DMA-ak eta beraien espezifikazioak interes handia sortu zuten konputazio-zientzietan. Esaterako, 2008an B. Liskov-ek *A. M. Turing Award*-a jaso zuen DMAen arloan eta arlo horrekin erlazionatutako bestelako gai batzuetan egindako ekarpen praktiko eta teorikoengatik[11]. Ikerketa sakon eta jarraitu baten ondorioz ekarpen anitz egiten ari dira hainbat azpiarlotan —adibidez, berridazketa-logikak, programazio-lengoaia ekuazionalak eta abar— baita gertuko arloetan ere —esate baterako, grafoen transformazioan—. [19] eta [38] lanetan XX. mendeko azken hiru hamarkadetan lortutako aurrerapenen laburpen onak (osoak, zehatzak eta erreferentzia askorekin) ematen dira, bai maila teorikoan, bai maila praktikoan, bereziki garatutako tresnak eta lengoaiak azalduz. Berriki izandako aurrerapenei dagokienez, liburu honekin erlazionatuen dagoena azpimarratuko dugu: espezifikazio aljebraikorako lengoaia batzuk frogatzaile automatiko interaktiboekin lotzen ari dira, adibidez, CASL lengoaia HOL/Isabelle frogatzaile automatiko interaktiboarekin lotu da espezifikatutako sistemen propietateen frogapena errazteko asmoz.

11. ACM (Association for Computing Machinery, AEB) elkarteak emanda, *A. M. Turing Award* sariak konputazio-zientzietan gertatzen diren ekarpen garrantzitsuenak aintzatesten ditu, eta arlo horretan lor daitekeen sari garrantzitsuena kontsideratzen da, hau da, *Konputazioaren Nobel sari*tzat har daiteke. 1966tik urtero ematen den saria da. Lehen sariduna Alan Jay Perlis izan zen. Eta Barbara Liskov izan zen saria jaso zuen bigarren emakumea. Saria jaso zuen lehen emakumea Frances Elizabeth Allen izan zen 2006an. 2012an ere beste emakume batek jaso zuen, Shafi Goldwasser-ek hain zuzen (Silvio Micalirekin batera).

5.9. ARIKETAK: DATU-MOTEN ESPEZIFIKAZIO EKUAZIONALA

1. Eman, *sekuentzia* DMA aberasteko helburuarekin, ondoren deskribatzen diren funtzioen espezifikazio ekuazionala (aurretik definitutako eragiketak erabil daitezke):

   1.1. Elementu bat sekuentzia batean agertzen den aldi kopurua itzuli.

   1.2. Zenbaki arruntez osatutako sekuentzia baten osagai bakoitzari bat balioa batu.

   1.3. Sekuentzia batean osagai errepikatuak dauden ala ez erabaki.

   1.4. Sekuentzia baten elementu errepikatuak ezabatu ezkerrerago dagoen osagaia utziz.

   1.5. Sekuentzia baten elementu errepikatuak ezabatu eskuinerago dagoen osagaia utziz.

   1.6. Sekuentzia batean ondoz ondoko bi osagai berdin, gutxienez behin, agertzen diren erantzun.

   1.7. Sekuentzia bat beste baten aurrizkia den ala ez erantzun.

   1.8. Sekuentzia bateko elementu guztiak banan-banan bikoiztu. Adibidez, $bikoiztu(\langle a,\ d,\ k,\ f\rangle)$ funtzioaren emaitza honako hau izango litzateke: $\langle a,\ a,\ d,\ d,\ k,\ k,\ f,\ f\rangle$.

   1.9. Sekuentzia bat emanda, osagai berdinez osatutako azpisekuentzia bakoitza azpisekuentziaren osagai batekin ordezkatu. Adibidez, $\langle 1,\ \underline{5,\ 5,\ 5,\ 5},\ 7,\ 9,\ 5,\ \underline{6,\ 6},\ 5\rangle$ sekuentzia emanda, emaitza honako hau izango da: $\langle 1,\ \underline{5},\ 7,\ 9,\ 5,\ \underline{6},\ 5\rangle$.

   1.10. Bi sekuentzia ordenaturen nahasketa ordenatua eman.

   1.11. Positiboak diren zenbaki osoz eratutako s sekuentzia emanda, s sekuentziaren balio bikoitiez eratutako pila eman. 0 zenbakia bikoititzat hartu behar da.

   1.12. *aurrizkia* funtzioak, s sekuentzia eta n zenbaki arrunta emanda, s-ko hasierako n elementuz osatutako azpisekuentzia itzultzen du. s sekuentziak n elementu ez badauzka, orduan s sekuentzia itzuliko da. Adibidez:

   - $aurrizkia(\langle 4,\ 4,\ 2,\ 1,\ 1,\ 8\rangle, 3) = \langle 4,\ 4,\ 2\rangle$
   - $aurrizkia(\langle 4,\ 4,\ 2,\ 1,\ 1,\ 8\rangle, 9) = \langle 4,\ 4,\ 2,\ 1,\ 1,\ 8\rangle$

2. Idatzi, *pila* DMA aberasteko helburuarekin, ondoren deskribatzen diren funtzioen espezifikazio ekuazionala (aurretik definitutako eragiketak erabil daitezke):

2.1. Bi pila emanda, bata bestearen gainean *irauli* ondoren lortzen den pila itzuli.

2.2. Bi pila emanda, bata bestearen gainean *gainjarriz* lortzen den pila itzuli.

2.3. Pila bat emanda, pilaren elementuez osatutako sekuentzia itzuli, pilaren gailurreko elementua sekuentziaren azkenengo elementu bezala ipiniz, eta pilaren beste elementuak ere ordena horretan ipiniz.

2.4. Pila bat eta e elementua emanda, e balioaren lehenengo agerpena baino gorago dauden elementu guztiak ezabatu pilan. e-ren lehenengo agerpena pilaren gailurretik hasitako lehenengo agerpena izango da. Eta e ez bada agertzen, pila hutsa itzuliko da.

2.5. Pila batean berdinak diren ondoz ondoko balioen errepikapenak ezabatu.

2.6. p pila bat, x elementu bat eta n zenbaki positiboa emanda, p-ren n-garren posizioan (gailurretik behera kontatuta) x-a agertzen den ala ez erabaki.

2.7. Pila batean e elementuaren lehenengo agerpena (gailurretik hasita) baino gorago dauden elementu guztiak beste pila batean irauli. e ez bada agertzen, pila osoa irauliko da.

3. Idatzi, *zuhbit* DMA-a aberasteko helburuarekin, ondoren deskribatzen diren funtzioen espezifikazio ekuazionala (aurretik definitutako eragiketak erabil daitezke):

3.1. Zuhaitz bitar baten adabegi kopurua kalkulatu.

3.2. Zuhaitza osoa den ala ez erabaki (zuhaitz bitar bat osoa izango da, haren adabegi guztiek 0 edo 2 ondorengo baldin badituzte).

3.3. Zuhaitz bitar bateko barne-adabegien (hostoak ez direnak) kopurua kalkulatu.

3.4. Zuhaitz bitar batek maila konkretu batean duen adabegi kopurua kalkulatu (erroaren maila 1 da, eta beste edozein adabegiren maila bere gurasoarena baino bat gehiago da).

3.5. Zuhaitz bitar batek maila konkretu batean duen adabegi-sekuentzia (ezkerretik eskuinera) itzuli.

3.6. Zuhaitz bitar bat beste baten aurrizkia den ala ez erabaki.

3.7. Zuhaitz bitar batean elementu bat agertzen den aldi kopurua kalkulatu.

3.8. Zuhaitz bitar baten muga kalkulatu (muga = hostoen sekuentzia).

3.9. Zuhaitz bitar batean bi ondorengo dituzten adabegien sekuentzia itzuli. Adibidez, 5.2. irudiko zuhaitzarentzat, irteerako sekuentzia honako hau izango da: $\langle 2, 4, 7 \rangle$.

3.10. Zuhaitz bitar batean bi ondorengo dituzten adabegien batura itzuli. Adibidez, 5.2. irudiko zuhaitzarentzat, emaitza honako hau izango da: $2+4+7=13$.

3.11. Bi zuhaitz bitar emanda, bata bestearen ispilu-irudia den ala ez erabaki.

3.12. Zuhaitz bitar bat *inordenan* zeharkatuz lortzen den sekuentzia itzuli.

3.13. Zuhaitz bitar bat eta e elementua emanda, erro bezala e elementua duten azpizuhaitz guztiak ezabatu zuhaitz hutsarekin ordezkatuz (adar batean e behin baino gehiagotan agertuz gero, sakonera txikieneko adabegian egin behar da ordezkapena). Hortaz, e balioa duen adabegirik ez da egongo irteerako zuhaitzean.

3.14. Bi zuhaitz bitar emanda, erro bezala bigarren zuhaitzaren erroaren balioa duten lehenengo zuhaitzeko azpizuhaitzak bigarren zuhaitzarekin ordezkatu (lehenengo zuhaitzaren adar batean bigarren zuhaitzaren erroa behin baino gehiagotan agertuz gero, sakonera gutxieneko adabegian egingo da ordezkapena).

3.15. Zuhaitz bitar bat emanda, erroaren balioa zuhaitz osora zabaldu, hau da, irteerako zuhaitzaren adabegi guztiek jatorrizko zuhaitzaren erroaren balioa izango dute.

3.16. Zuhaitz bitar bat eta x eta y balioak emanda, x-ren agerpenak zuhaitzean y-rekin ordezkatu.

4. Idatzi ondoren deskribatzen diren eragiketak burutzen dituzten funtzioen espezifikazio ekuazionala:

4.1. p pilak eta s sekuentziak elementu berak eta ordena berean dituzten erabaki: p-ko gailurrekoa s-ko lehenengoari egokitzen zaio.

4.2. Sekuentzia bat (T motako elementuz osatua) zuhaitz bitar baten (T motako elementuz osatua) *adar osoa* den ala ez erabaki. Hau da, funtzio horrek erabakitzen du ea emandako sekuentziak emandako zuhaitzaren errotik hostoren batera joateko igaro behar diren adabegi guztiak dauzkan, eta igarotako ordena berean.

5. Idatzi honako enuntziatu hauek adierazten dituzten asertzioak aurreko ariketetan definitutako funtzioak erabiliz.

5.1. s eta r sekuentzia ordenatuen nahasketa eginda, t sekuentziaren alderantzizkoa lortzen da.

5.2. Zenbaki osozko p pila zenbaki palindromoa eratzen duten digituz osatuta dago.

5.3. a zuhaitzak barne-adabegi gehiago ditu hostoak baino.

5.4. a zuhaitzaren mugak b zuhaitzaren mugak baino adabegi gehiago ditu.

5.5. a zuhaitz bitarra inordenan zeharkatzean lortzen den sekuentzia p pila behetik gora zeharkatuta lortzen diren elementuen sekuentziaren berdina da.

6. Frogatu formalki honako propietate hauek[12]:

6.1. s edozein sekuentzia izanda, $s@\langle\,\rangle = s$.

6.2. s_1, s_2 eta s_3 edozein sekuentzia izanda:

$$(s_1@s_2)@s_3 = s_1@(s_2@s_3)$$

6.3. s_1 eta s_2 edozein sekuentzia izanda:

$$alderantzizkoa(s_1@s_2) = alderantzizkoa(s_2)@alderantzizkoa(s_1)$$

6.1. eta 6.2. ariketetako propietateak erabili behar dira.

6.4. s edozein sekuentzia izanik:

$$luzera(s) = luzera(alderantzizkoa(s))$$

Zenbaki arrunten batuketaren trukatze-propietatea eta sekuentzien gaineko honako propietate hau erabili:

$$luzera(u@v) = luzera(u) + luzera(v)$$

6.5. s edozein sekuentzia izanik:

$$alderantzizkoa(alderantzizkoa(s)) = s$$

6.3. propietatea erabili behar da.

12. Propietate bakoitzean parte hartzen duten elementuek mota egokia dutela ulertzen da.

6.6. Edozein x elementu eta edozein s_1 eta s_2 sekuentzientzat:

$$aldikop(x, s_1@s_2) = aldikop(x, s_1) + aldikop(x, s_2)$$

aldikop eragiketak elementu bat eta sekuentzia bat emanda, elementua sekuentzian zenbat aldiz agertzen den itzultzen du.

6.7. s edozein sekuentzia izanik:

$$aldikop(x, s) = aldikop(x, alderantzizkoa(s))$$

6.6. propietatea erabili behar da.

6.8. s edozein sekuentzia izanik:

$$aldikop(x, ezabatu(x, s)) = 0$$

6.9. s_1 eta s_2 edozein sekuentzia izanik:

$$x \in (s_1@s_2) = (x \in s_1) \vee (x \in s_2)$$

Disjuntzioaren ($\vee$) elkartze-propietatea erabili.

6.10. s edozein sekuentzia izanik:

$$x \in s = x \in alderantzizkoa(s)$$

Disjuntzioaren ($\vee$) trukatze-propietatea erabili, eta 6.9. ariketako propietatea.

6.11. s_1 eta s_2 edozein sekuentzia izanik:

$$ezabatu(x, s_1@s_2) = ezabatu(x, s_1)@ezabatu(x, s_2)$$

6.12. s edozein sekuentzia izanik:

$$ezabatu(x, alderantzizkoa(s)) = alderantzizkoa(ezabatu(x, s))$$

6.11. ariketako propietatea erabili.

6. Errekurtsibotik iteratiborako transformazioa

Kapitulu honetan, transformazio-metodo bat aurkeztuko da. Metodo horrek programa errekurtsibo batetik hasita baliokidea den programa iteratibo bat aterako du era arrazoitu eta justifikatuan. Programak baliokideak izango dira, funtzio beraren inplementazio desberdinak izango direlako. Are gehiago, programen baliokidetasuna asertzioen bidez frogatuta eta dokumentatuta geratuko da, batez ere iterazioak beteko duen inbariantearen bidez. 6.1. atalean, errekurtsibotik iteratiborako transformazioaren ideia eta garrantzia azalduko dira. *Destolesketa/tolesketa* metodoa (ikus [23]) 6.2. atalean aurkeztuko da, adibide errazen bidez zein errekurtsibotik iteratiborako transformazioa errazten duten ezaugarriak dituzten programak abstraitzen dituzten programa-eskemen bidez. 6.3. atalean, aurreko atalean aurkeztutako oinarrizko eskemara egokitzen diren ariketa batzuk proposatuko dira. 6.4. atalean, metodoaren erabilgarritasuna zabalduko da oinarrizko eskema orokortuz. Lau aldaera proposatuko dira, eta aldaera horietako bakoitzeko adibide bat sakonki aztertuko da dagokion azpiatalean. 6.5. atalean, metodo horien jatorri eta bilakaerari buruzko bibliografia-oharrak eta informazioa bildu da. Azkenik, 6.6. atalean, kapituluan ikusitako teknikei buruzko ariketak datoz.

6.1. ERREKURTSIBOTIK ITERATIBORA TRANSFORMATZEKO METODOAK

Problema mota askotarako, soluzio algoritmikoak definitzeko era zuzenena, sinpleena eta argiena errekurtsioa da. 4. kapituluan ikusitako programa errekurtsiboak eta 5. kapituluan aurkeztutako espezifikazio ekuazionalak dira algoritmoak eta espezifikazioak (exekutagarriak izan daitezkeenak) definitzeko teknika errekurtsiboaren erabileraren adibideak. Baina batzuetan, soluzio iteratiboak errekurtsiboak baino eraginkorragoak izaten dira. Soluzio errekurtsiboetan, datuak (parametroak eta aldagai lokalak) gorde eta berreskuratu behar izateak exekuzio-kostua areagotu egiten du, eta batzuetan kalkuluen errepikapenak ere gertatzen dira. Horrek guztiak eraginkortasun eza eragiten du, bai denbora aldetik bai espazio aldetik. Adibidez, Fibonacci-ren

zenbakiak itzultzen dituen funtzioaren zuzeneko inplementazioa (ikusi 6.4.2. atala) oso txarra da eraginkortasunari dagokionez, kalkuluen errepikapenagatik. Datuak gorde eta berreskuratu beharrak sortzen duen arazoa oso erraz uler daiteke n zenbaki arruntaren faktorialaren kalkulu errekurtsiboa aztertuz; izan ere, hasteko n, $(n-1)$, ..., 3, 2, 1, 0 balioak gordez joan behar da, eta, gero, alderantzizko ordenan berreskuratu beharko dira. Kalkulu iteratiboan, ordea, arazo hori ez da gertatzen. Oro har, ebazpen errekurtsiboetan n dei eragiten dituen dei baten ebaluazioak n altuerako pila bat erabili behar du. Bertsio iteratiboak, aldiz, espazio konstantearekin nahikoa izan dezake gauzatu beharreko iterazio kopurua edozein izanda ere. Askotan, soluzio errekurtsiboa kontuan hartzen ez bada, eraginkorra den programa iteratibo bat diseinatzea oso zaila da. Aitzitik, programa errekurtsiboa iteratibo bihurtzeko diseinu-metodoari jarraituz, soluzio zuzen, sinple eta argi batetik hasita (programa errekurtsiboa bera) ateratzen den programa iteratiboa era arrazoituan justifikatuta geratuko da. Gehienetan, hasierako programa errekurtsiboa espezifikaziotik oso gertu dago (edo espezifikazioa bera da), eta ikuspuntu horretatik transformazioa programa-eratorpentzat har daiteke. Beraz, programa errekurtsibo batetik hasita, baliokidea den programa iteratibo bat lortuko da. Transformazio horren helburu nagusia eraginkorragoa den programa bat edukitzea izango da. Diseinu-metodo hau sendoa eta erabilgarria da, transformazioa era metodikoan egiten delako baliokidetasuna bermatuz. Honela eraikitako programa iteratiboak dagozkien bertsio errekurtsiboak baino eraginkorragoak izango dira, eta beraien zuzentasuna ondo dokumentatuta eta justifikatuta geldituko da.

Errekurtsibotik iteratiborako transformazio-metodoen artean, oso ohikoak dira kalkulu errekurtsiboaren imitazio iteratiboan oinarritzen direnak. Oso ezaguna da konpilatzaileak erabiltzen duen pilaren kudeaketa imitatzen duen metodoa (besteak beste, ikusi [8]). Konpilatzaileak deien parametroak eta aldagai lokalak gordetzen ditu pilan. Metodo horri jarraituz ateratako iterazioak konpilatzaileak egiten dituen pilaratzeko eta despilatzeko eragiketa guztiak egiten ditu. Horrela lortutako iterazioaren exekuzioa motela ez izateko, kodea transformatu edo optimizatu beharko da. Beste metodo batzuetan (ikusi [86]), dei errekurtsiboen zuhaitza zeharkatzeko era emulatzen da. Exekuzioa imitatzen duten metodo hauei kontrajarrita, deskribapen errekurtsiboaren errekurrentzia-erlazio batean oinarritzen diren transformazio-teknikak ditugu. Teknika horiek moldakorragoak dira eta betetzen den errekurrentzia-erlazio bakoitza simulatuz programa iteratibo bat ateratzea ahalbidetzen dute. Kapitulu honetan aurkeztuko dugun metodoa deskribapen errekurtsiboen transformazio formalean oinarritzen da. Metodo hori *destolesketa/tolesketa metodoa* izenez ezagutzen da eta R. M. Burstall eta

J. Darlington-ek proposatu zuten 1977. urtean [23]. Metodo horren bitartez transformatzen den jatorrizko deskribapen errekurtsiboa programa errekurtsibo bat, definizio induktibo bat edo espezifikazio ekuazional bat izan daiteke, eta deskribapen errekurtsibo horrek betetzen duen errekurrentzia-erlazioa inbariantetzat hartuko da programa iteratibo bat formalki eratortzeko. Metodoan erabiltzen den programen transformazioa programak egiaztatzeko eta eratortzeko balio duten teknika formalekin erlaziona daiteke.

Azkenik, ikuspegi pedagogikotik begiratuta, errekurtsibotik iteratiborako transformazio-metodoek errekurtsioaren eta iterazioaren arteko erlazioa aztertzeko aukera ematen dute, eta hori oso garrantzitsua da programazioa era egokian ikasten hasteko. Beraz, kapitulu honen helburua bikoitza da. Batetik, errekurtsibotik iteratiborako transformazio-metodoak aurkeztea programa iteratiboen diseinurako. Bestetik, errekurtsioa bera eta errekurtsioak iterazioarekin duen erlazioa sakonago aztertzea.

6.2. DESTOLESKETA/TOLESKETA METODOA — OINARRIZKO ESKEMA

Atal honetan, destolesketa/tolesketa metodoa zer den azaltzeko asmoz, adibide batzuei eta programa-eskema bati aplikatuko zaie aipatutako metodoa. Metodo honetan programa errekurtsiboaren semantika adierazten duen errekurrentzia-erlazioa inbariantetzat erabiliko da programa iteratibo bat lortzeko. Programa errekurtsiboak, eta bereziki haren kasu errekurtsiboek, errekurrentzia-erlazioa nola destolestu behar den adieraziko dute argumentu desberdinekin bere buruari dei egiten dionean. Tolesketa destolesketarekin lortzen diren adierazpenen bateratzea da, inbariantea berriro betearazteko. Horrela, iterazioak inbariantea kontserbatuko du. Gainera, programa errekurtsiboaren kasu nabariek iterazioak noiz bukatu behar duen adierazten dute, eta, inbariantearekin batera, iterazioa bukatutakoan itzuli behar den emaitza definitzeko balio dute. Iterazioa exekutatu aurretik inbariantea bete dadin, hasieraketan inbarianteko aldagaiei balio egokiak esleituko zaizkie. Laburbilduz, iterazioaren diseinua programa errekurtsiboan eta inbariante batean oinarritzen da. Diseinua hiru fasetan burutzen da, bakoitza iterazio baten zuzentasun partziala bermatzen duen propietate bati dagokiona (3.9. atala): (1) hasieraketa, (2) bukaera eta (3) iterazioaren gorputza[1].

Argitasunari lehentasuna emateko eta metodoa hobeto ulertzen laguntzeko, hasteko definizio errekurtsibo sinpleetarako aurkeztuko da metodoa.

1. Iterazioen eratorpen formalaren metodoak ere hiru urrats horiei jarraitzen die (ikusi 7. kapitulua).

Programa-eskema bati eta eskema horretara egokitzen den adibide bati aplikatuko zaie metodoa aldi berean. Bai eskemak bai adibideak kasu nabari bakar bat eta kasu errekurtsibo bakar bat izango dituzte, eta kasu errekurtsiboan dei errekurtsibo bakar bat egingo da. 6.4. atalean, metodoa kasu orokorrera zabalduko da.

Izan bedi honako eskemara egokitzen den funtzio baten deskribapen errekurtsiboa:

$$f : T_1 \times \ldots \times T_n \rightarrow T' \qquad (6.1)$$
$$f(\overline{x}) = \begin{cases} b(\overline{x}) & \text{baldin } a(\overline{x}) \\ f(t(\overline{x})) \otimes g(\overline{x}) & \text{bestela} \end{cases}$$

Hor, $\overline{x} = x_1, \ldots, x_n$ parametro formalak dira. Parametro horiek, alde batetik, $x_1 \in T_1, \ldots, x_n \in T_n$ betetzen dute eta, beste aldetik, $\varphi(\overline{x})$ aurrebaldintza betetzen dute. Bestalde, T' emaitzaren datu-mota da. Hemendik aurrera, $\overline{T}$ adierazpenak $T_1, \ldots, T_n$ laburtuko du. Atal honetan, $\otimes$ eragiketa elkarkorra dela eta haren elementu neutroa ε dela suposatuko dugu. Aurreko eskema betetzen duen funtzio baten zuzeneko inplementazio errekurtsiboa honako hau izango litzateke:

```
function f_er(x:  T) return e:  T' is
{ φ(x) }
begin
  if a(x) then
    e := b(x);
  else
    e := f_er(t(x))⊗g(x);
  end if;
end f;
{ e = f(x) }
```

Metodoa aurkezteko, aurreko eskemaren adibidea den *oinarria_ald* funtzioa erabiliko dugu. *oinarria_ald* funtzioak n zenbaki arruntaren *oin* oinarriko adierazpena itzuliko du, baldin eta $2 \leq oin \leq 9 \wedge n \geq 0$:

$$oinarria\_ald\colon oso \times oso \rightarrow sekuentzia(oso) \qquad (6.2)$$
$$oinarria\_ald(n, oin) = \begin{cases} \langle n \rangle & \text{baldin } n < oin \\ oinarria\_ald(n/oin, oin) @ \langle n \,\%\, oin \rangle & \text{bestela} \end{cases}$$

$oinarria\_ald(n, oin)$ adierazpenaren balioa *oin* oinarrian n zenbaki arrunta adierazten duen digitu-sekuentzia da. *oinarria_ald* funtzioa honako era honetan inplementa daiteke errekurtsiboki:

```
function oinarria_ald_er(n,oin:  Integer)
return e:  sekuentzia(Integer) is
```
$\{\ 2 \leq oin \leq 9 \wedge n \geq 0\ \}$
```
begin
  if n < oin then
    e := <n>;
  else
    e := oinarria_ald_er(n/oin,oin) @ <n % oin>;
  end if;
end oinarria_ald;
```
$\{\ e = oinarria\_ald(n, oin)\ \}$

Definizio errekurtsibo bat iteratibo bihurtzeko, errekurrentzia-erlazio bat lortu behar dugu. Errekurrentzia-erlazio horrek erakutsi behar du zein izango den kasu nabari batera iritsi arte definizio errekurtsiboaren aplikazio sistematikoa. Errekurrentzia-erlazioa lortzeko, destolesketa/tolesketa teknika erabiliko dugu adibide zehatz baten gainean. Esate baterako, $oinarria\_ald(123,4)$ adierazpena hartuko dugu eta espresio horren kalkulua egingo dugu definizio errekurtsiboari jarraituz. Kalkulu horretan, destolesketa-urrats bakoitzean funtzioaren definizioa aplikatu behar da, eta tolesketa-urrats bakoitzean destolesketaren bidez lortutako adierazpena sinplifikatu behar da:

$$
\begin{aligned}
& oinarria\_ald(123,4) \\
& \quad = oinarria\_ald(30,4)@\langle 3\rangle && \text{(destolestu)} \\
& \quad = (oinarria\_ald(7,4)@\langle 2\rangle)@\langle 3\rangle && \text{(destolestu)} \\
& \quad = oinarria\_ald(7,4)@\langle 2,\ 3\rangle && \text{(tolestu)} \\
& \quad = (oinarria\_ald(1,4)@\langle 3\rangle)@\langle 2,3\rangle && \text{(destolestu)} \\
& \quad = oinarria\_ald(1,4)@\langle 3,\ 2,\ 3\rangle && \text{(tolestu)} \\
& \quad = \langle 1\rangle@\langle 3,\ 2,\ 3\rangle \\
& \quad = \langle 1,\ 3,\ 2,\ 3\rangle
\end{aligned}
$$

Destolesketa-urratsak kasuan kasuko adierazpenetan funtzioaren kasu errekurtsiboa aplikatzean dautza. Horrela, $oinarria\_ald(123,4)$ adierazpena destolestuz $oinarria\_ald(30,4)@\langle 3\rangle$ adierazpena lortuko dugu, eta, era berean, $oinarria\_ald(30,4)$ azpiadierazpena destolestuz $(oinarria\_ald(7,4)@\langle 2\rangle)@\langle 3\rangle$ gelditu ko da. Tolesketa-urratsak, destolesketaren bidez lortutako adierazpenak sinplifikatzeko dira, eta ezinbestekoa da adierazpen horietan agertzen den eragiketa elkarkorra izatea. Adibide horretan agertzen den eragiketa, hau da, sekuentzien kateaketa (@) elkarkorra da. Beraz, destolesketa-urratsean lortutako $(oinarria\_ald(7,4)@\langle 2\rangle)@\langle 3\rangle$ espresioa tolestu ondoren $oinarria\_ald(7,4)@(\langle 2\rangle@\langle 3\rangle)$ lortuko da, eta hori $oinarria\_ald(7,4)@\langle 2,\ 3\rangle$

espresioaren baliokidea da. Bigarren tolesketa-urratsa ere antzeko eran egingo da. Errekurrentzia-erlazioa transformazio hori aztertuz lor daiteke, destolesketa/tolesketa urrats bakoitzaren ondoren gelditutako adierazpena kontuan hartuz. Kasu honetan, $oinarria\_ald(123,4)$ hasierako adierazpenetik honako espresio hauek atera dira:

$$\begin{aligned} oinarria\_ald(123,4) &= oinarria\_ald(30,4)@\langle 3\rangle \\ &= oinarria\_ald(7,4)@\langle 2,\ 3\rangle \\ &= oinarria\_ald(1,4)@\langle 3,\ 2,\ 3\rangle \end{aligned}$$

Aurreko adierazpenak orokortuz, n eta oin hasierako edozein bi balio izanda ere, $oinarria\_ald(n,oin)$ espresioaren balioa $oinarria\_ald(m_1,m_2)@s$ bezala adieraz daiteke m_1, m_2 eta s aldagaien balio egokietarako. Beraz, errekurrentzia-erlazioa honako hau izango litzateke:

$$oinarria\_ald(n,oin) = oinarria\_ald(m_1,m_2)@s$$

Eraikiko den iterazioak konputazio hori simulatuko du $oinarria\_ald(n,oin)$ espresioaren balioa kalkulatzeko. Simulazio horretan, m_1, m_2 eta s aldagaiak erabiliko dira une desberdinetan sortuko diren balioak gordetzeko. Aurreko adibidean, m_1 aldagaiak 123, 30, 7 eta 1 balioak hartuko ditu prozesuan zehar. Erraz ikus daitekeenez, m_2 aldagaiak oin balioa hartuko du iterazioaren urrats guztietan. Hortaz, honako errekurrentzia-erlazioa hau aurrekoa baino egokiagoa da:

$$oinarria\_ald(n,oin) = oinarria\_ald(m,oin)@s$$

Kontuan izan lehenengo errekurrentzia-erlazioa erabiliko balitz, eraikiko litzatekeen programaren konputazioan zehar m_2 aldagaiak oin balio konstantea hartuko lukeela. Azkenik, iterazio bakoitzean elementu berri bat izango duen sekuentzia gordeko da s aldagaian: $\langle 3\rangle, \langle 2,\ 3\rangle$ eta $\langle 3,\ 2,\ 3\rangle$ balioak hartuko ditu, hain zuzen ere.

Metodo bera 6.1. eskemara egokitzen den edozein f funtziori aplikatuz gero, honako errekurrentzia-erlazio hau lortuko da:

$$f(\overline{x}) = f(\overline{y}) \otimes z$$

Hor, $\otimes$ eragiketa $T' \times T' \to T'$ motakoa den eta elementu neutro bezala ε balioa duen eragiketa elkarkorra izango da. Gainera, $\overline{y}$ eta z aldagai berriak

izango dira[2]. Garatu dugun *oinarria_ald* adibidean bezala, errekurrentzia-erlazioa iterazioaren inbariantetzat hartuko da, eta aldagai berriak konputazio errekurtsiboa simulatzeko erabiliko dira, iterazio bakoitzean destolesketa/tolesketa urrats bat emanez eta errekurrentzia-erlazioa kontserbatuz. 3.9. atalean ikusitako inbariantearen kontzeptuaren arabera, iterazio zuzen batek inbariantearekiko bete behar dituen hiru propietateak bermatzea da helburua:

- Hasieraketaren bidez $\overline{y}$ eta z aldagai berriei esleitzen zaizkien balioek inbariantea betearazi behar dute.
- Iterazioa bukatu ondoren, $\overline{y}$ eta z aldagaiak erabil daitezke funtzioaren emaitza kalkulatzeko, postbaldintza betez.
- Iterazioaren gorputzean, balio berriak esleitu behar zaizkie $\overline{y}$ eta z aldagaiei inbariantea kontserbatuz. Aldagai horien transformazio bakoitza destolesketa/tolesketa urrats bat eginez erabakiko da. Horrez gain, aldaketa horrekin $\overline{y}$ aldagaiez bakarrik osatuta egongo den borne-adierazpenaren balioa txikiagotu egingo dela frogatu beharko da, iterazioaren bukaera bermatuz.

Funtzio iteratiboari *f_it* deituko zaio eta haren goiburua bat etorriko da simulatu behar duen *f_er* funtzio errekurtsiboaren goiburuarekin:

```
function f_it(x:  T) return e:  T' is
```

Ikus daitekeen bezala, sarrera-parametro ($\overline{x}$) eta emaitza (e) berak ditu, eta mota berekoak: $\overline{T}$ eta T' hurrenez hurren. Gainera, errekurrentzia-erlazioan erabiltzen diren $\overline{y}$ eta z aldagai berriak erazagutu behar dira. Aurretik ikusi dugunez, $\overline{x}$ eta e-ren mota berekoak izango dira hurrenez hurren:

```
y:  T;
z:  T';
```

oinarria_ald funtzioaren kasuan, funtzio iteratiboaren goiburua eta aldagai erazagupena honela gelditu dira:

```
function oinarria_ald_it(n,oin:  Integer)
return e:  sekuentzia(Integer) is
  m:  Integer;
  s:  sekuentzia(Integer);
```

2. $\otimes$ eragiketa elkarkorra ez bada, orduan $\otimes$ eragiketaren mota $T' \times S \to T'$ izango da, S edozein mota izanda, eta errekurrentzia-erlazioan $\otimes$ eragiketaz bestelakoak diren eragiketak erabiliko dira (bat edo gehiago).

Ondoren, kalkulu errekurtsiboa simulatzen duen iterazioaren atal bakoitza deskribatuko dugu, bai *oinarria_ald* funtziorako (ikusi 6.2.) bai 6.1. eskemari jarraitzen dion edozein funtziotarako:

- Hasieraketa. Errekurrentzia-erlazioko aldagai berrien hasierako balioek inbariantea bete behar dute era nabarian, destolesketa/tolesketa urratsik egin gabe. Beraz, *oinarria_ald* funtzioaren kasuan, honako aldibereko esleipen hau egitea nahikoa da:

  $$\texttt{(m,s) := (n,<>);}$$

  Inbariantean m eta s aldagaiak beraien hasierako balioekin ordezkatuz, hurrengo egiazko adierazpena lortzen da:

  $$oinarria\_ald(n, oin) \;=\; oinarria\_ald(n, oin)@\langle\,\rangle$$

  6.1. eskemako egitura duten funtzioen kasuan, honako aldibereko esleipen hau egin behar da:

  $$\texttt{(}\overline{\texttt{y}}\texttt{,z) := (}\overline{\texttt{x}}\texttt{,}\varepsilon\texttt{);}$$

  Hor, ε balioa $\otimes$ eragiketaren elementu neutroa da. Hasierako balioek inbariantea betearazten dute era nabarian:

  $$f(\overline{x}) \;=\; f(\overline{x}) \otimes \varepsilon$$

  ε balioa $\otimes$ eragiketaren elementu neutroa baita.

- Bukaera. Iterazioaren bukaera-baldintza definitzeko, kontuan hartu behar da iterazioak destolesketa/tolesketa urratsak simulatuko dituela. Beraz, iterazioak bukatu behar du $\overline{y}$ aldagaien balioek kasu nabariaren baldintza betetzen dutenean. 6.1. eskeman kasu nabariaren baldintza $a(\overline{x})$ denez, iterazioaren baldintza honako hau izango da:

  $$\underline{\texttt{while}}\ \texttt{not a(}\overline{\texttt{y}}\texttt{)}\ \underline{\texttt{loop}}$$

  Hor, $\overline{x}$ aldagaiak $\overline{y}$ aldagaiekin ordezkatu dira. Bestalde, *oinarria_ald* funtzioaren kasuan, kasu nabariaren baldintza $n < oin$ da, hortaz iterazioaren baldintza honako hau izango da:

  $$\underline{\texttt{while}}\ \texttt{not (m < oin)}\ \underline{\texttt{loop}}$$

  Baldintza hori lortzeko, n aldagaia m-rekin ordezkatu da. Gainera, 6.1. eskemako funtzioek $f(\overline{y})$ deirako $b(\overline{y})$ balioa itzultzen dute kasu nabariaren baldintza betetzen bada. Inbariantean $f(\overline{y})$ deia $b(\overline{y})$ balioarekin ordezkatuz, honako adierazpen hau geldituko da:

  $$f(\overline{x}) \;=\; b(\overline{y}) \otimes z$$

Beraz, honako esleipen honek postbaldintza betearaziko du:

```
e := b(ȳ) ⊗ z;
```

oinarria_ald funtzioaren kasuan, $oinarria\_ald(m, oin)$ deiak $\langle m \rangle$ balioa itzuliko du, kasu nabariaren baldintza betetzen bada. Hortaz, inbariantea honela gelditu ko da:

$$oinarria\_ald(n, oin) = \langle m \rangle @ s$$

eta e-ri egin beharreko esleipena honako hau izango da:

```
e := <m> @ s;
```

Edo baliokidea den beste hau:

```
e := m • s;
```

- Iterazioaren gorputza (destolesketa/tolesketa). Iterazioaren ziklo bakoitzean, $\overline{y}$ eta z aldagaiei balio berriak esleitu behar zaizkie inbariantea kontserbatuz. Honako irizpide hau erabiliko da: iterazio-ziklo bakoitzean destolesketa/tolesketa urrats bat *simulatu* behar da. Hau da, destolesketa/tolesketa urrats bakoitzak $\overline{y}$ eta z aldagaietan egiten dituen aldaketak dira iterazioaren gorputzak egin behar dituen aldaketak. Hortaz, kasu errekurtsiboa erabiliz, inbariantea destolestu eta tolestu egin behar da $\overline{y}$ eta z aldagaien balio berriak kalkulatzeko. 6.1. eskemako funtzioen kasuan eta $\otimes$ eragiketa elkarkorra dela suposatuz, destolesketa/tolesketa urrats bat honelakoa izango da:

$$
\begin{array}{ll}
f(\overline{x}) & \\
\quad = f(\overline{y}) \otimes z & \\
\quad = (f(t(\overline{y})) \otimes g(\overline{y})) \otimes z & \text{(destolestu)} \\
\quad = f(\underbrace{t(\overline{y})}) \otimes \underbrace{(g(\overline{y}) \otimes z)} & \text{(tolestu)} \\
\quad = f(\ \overline{y}'\ ) \otimes \qquad z' &
\end{array}
$$

Hor, $\overline{y}'$ eta z' adierazpenak $\overline{y}$ eta z aldagaien balio berriak dira. Hau da, iterazioaren gorputzak honako aldibereko esleipena inplementatu behar du:

$$(\overline{y},z) := (t(\overline{y}),g(\overline{y}) \otimes z);$$

oinarria_ald funtzioa hartuz eta kasu errekurtsiboa erabiliz, inbariantearen destolesketa/tolesketa urrats bat honelakoa da:

$$\begin{aligned}
&oinarria_ald(n,oin)\\
&\quad = oinarria_ald(m,oin)@s\\
&\quad = (oinarria_ald(m/oin,oin)@\langle m \,\%\, oin\rangle)@s && \text{(destolestu)}\\
&\quad = oinarria_ald(\underbrace{m/oin}_{m'},oin)@\underbrace{(\langle m \,\%\, oin\rangle@s)}_{s'} && \text{(tolestu)}\\
&\quad = oinarria_ald(\quad m' \quad ,oin)@\qquad s'
\end{aligned}$$

Beraz, iterazioaren gorputza honako aldibereko esleipen hau izango da:

```
(m,s) := (m/oin,<m % oin> @ s);
```

Erraz ikus daitekeenez, m borne-adierazpentzat har daiteke, iterazioaren ziklo guztietan gero eta txikiagoak diren balioak hartuko ditu eta. Hau da, $\neg(m < oin)$ iterazioaren baldintza betetzen denez, $m \geq oin \geq 2$ ondoriozta daiteke *oinarria_ald* funtzioaren $(2 \leq oin \leq 9 \wedge n \geq 0)$ aurrebaldintzatik. Beraz, $m > m/oin$ ere beteko da, m/oin balioa iterazio bakoitzean m-ri esleitzen zaion balioa izanda.

Iterazioaren gorputza $\overline{y}$ eta z aldagaiei egindako aldibereko esleipen bat da. Programazio-lengoaia guztietan ezin denez egin aldibereko esleipena, aldibereko esleipenak sekuentziatu egin beharko dira, beharrezkoa denean aldagai laguntzaileak erabiliz. *oinarria_ald* funtzioaren kasuan, s-ren balio berria m-ren hasierako balioa erabiliz kalkulatu behar da. Hau da, m-ren balioa eguneratu aurretik. Hortaz, lehenik s-ren balioa eguneratuko dugu eta ondoren m-ren balioa:

```
s := <m % oin> @ s;
m := m/oin;
```

Laburbilduz, 6.1. eskemarekin bat datozen funtzioetan destolesketa/tolesketa teknika erabiliz honako era honetako funtzioak lortuko dira:

```
function f_it(x̄: T̄) return e: T' is
  ȳ: T̄;
  z: T';
{ φ(x̄) }
begin
  (ȳ,z) := (x̄,ε);
  while { f(x̄) = f(ȳ) ⊗ z }
    not a(ȳ)
  loop { E ≡ e(ȳ) }
    (ȳ,z) := (t(ȳ),g(ȳ) ⊗ z);
  end loop;
  e := b(ȳ) ⊗ z;
end f_it;
{ e = f(x̄) }
```

Adibidez, *oinarria_ald* funtzioari destolesketa/tolesketa teknika aplikatuz, honako funtzio hau lortuko da:

```
function oinarria_ald_it(n,oin: Integer)
return e: sekuentzia(Integer) is
  m: Integer;
  s: sekuentzia(Integer);
{ 2 ≤ oin ≤ 9 ∧ n ≥ 0 }
begin
  m := n;
  s := <>;
  while { oinarria_ald(n,oin) = oinarria_ald(m,oin)@s }
    not (m < oin)
  loop { E ≡ m }
    s := <m % oin> @ s;
    m := m/oin;
  end loop;
  e := <m> @ s;
end oinarria_ald_it;
{ e = oinarria_ald(n,oin) }
```

6.3. ARIKETAK: DESTOLESKETA/TOLESKETA METODOA — OINARRIZKO ESKEMA

1. Bihurtu iteratibo f: $oso \times oso \rightarrow oso$ funtzio errekurtsiboa destolestu/tolestu metodoa erabiliz. Lehenengo argumentuak ez-negatiboa izan behar du, hortaz, $x \geq 0$ aurrebaldintza ezarri behar da:

$$f(x,y) = \begin{cases} y & \text{baldin } x = 0 \\ x * y + f(x-1, y+1) & \text{baldin } x > 0 \end{cases}$$

2. Bihurtu iteratibo $disjuntzioa$: $sekuentzia(bool) \rightarrow bool$ funtzio errekurtsiboa destolestu/tolestu metodoa erabiliz. $True$ aurrebaldintza dugu sarrera gisa, balio boolearrez osatutako edozein sekuentzia onartuko baita:

$$(1)\ disjuntzioa(\langle\,\rangle) = \mathsf{F}$$
$$(2)\ disjuntzioa(x \bullet s) = x \ \vee\ disjuntzioa(s)$$

3. Bihurtu iteratibo $berdinak$: $sekuentzia(T) \times sekuentzia(T) \rightarrow bool$ funtzio errekurtsiboa destolestu/tolestu metodoa erabiliz. Sarrerako bi sekuentziek luzera bera eduki behar dute:

$$(1)\ berdinak(\langle\,\rangle, r) = \mathsf{T}$$
$$(2)\ berdinak(x \bullet s, r) = (\ x = lehena(r)\ ) \wedge berdinak(s, hondarra(r))$$

6.4. DESTOLESKETA/TOLESKETA METODOA — ESKEMA OROKORRA

Deskribapen errekurtsiboek kasu nabari bat eta kasu errekurtsibo bat baino gehiago izan dezakete. Are gehiago, kasu errekurtsibo batek dei errekurtsibo bat baino gehiago egin dezake. Aukera horien guztien ondorioz, errekurrentzia-erlazioa (errekurrentzia-erlaziorik baldin badago) kasu errekurtsiboetan erabiltzen ez den beste eragiketa batean oinarri daiteke. Oro har, $\overline{x}$ parametro formalak dituen f funtzio baten *errekurrentzia-erlazioa* honelakoa izango da:

$$f(\overline{x}) \ =\ F(f(\overline{y}), \overline{z})$$

Hor, F funtzioak $f(\overline{y})$ eta $\overline{z}$ erabiliz $f(\overline{x})$ deiaren emaitza kalkulatuko du. Kasu nabariek iterazioaren bukaera adierazten dute: $\overline{y}$ tupla osatzen duten aldagaien balioek kasu nabari baten baldintza betetzen dutenean iterazioak

bukatu egin behar du $f(\overline{y})$ adierazpena ezin delako destolestu. Beraz, iterazioaren bukaera-baldintza kasu nabari guztien baldintzen disjuntzioa da, $\overline{x}$ aldagaiak $\overline{y}$ aldagaiekin ordezkatuz. Eta e aldagaian itzuliko den emaitza kalkulatzeko, betetzen den kasu nabariari dagokion balioa inbariantean $f(\overline{y})$ adierazpena dagoen lekuan ipini beharko da. Hau da, $f(\overline{x})$-ren definizioak hurrengo n kasu nabari baldin baditu:

```
if a1(x̄) then
  f(x̄) := b1(x̄);
elsif a2(x̄) then
  f(x̄) := b2(x̄);
⋮
elsif an(x̄) then
  f(x̄) := bn(x̄);
end if;
```

iterazioaren bukaera-baldintza $a_1(\overline{y}) \vee a_2(\overline{y}) \vee \ldots \vee a_n(\overline{y})$ izango da, eta bukaeran e aldagaiari esleituko zaion balioa honela kalkulatuko da:

```
if a1(ȳ) then
  e := F(b1(ȳ),z̄);
elsif a2(ȳ) then
  e := F(b2(ȳ),z̄);
⋮
elsif an(ȳ) then
  e := F(bn(ȳ),z̄);
end if;
```

Bestetik, kasu errekurtsibo bat baino gehiago dagoenean, destolesketa/tolesketa urrats bat egingo da kasu errekurtsibo bakoitzeko. Eta baldintzazko agindu bat erabiliz, iterazioaren gorputzean kasu bakoitzeko egin behar den aldibereko esleipena inplementatuko da. Kasu bakoitzean, inbariantean $f(\overline{y})$ destolestuko da eta, gero, destolesketaren bidez lortutako espresioa tolestu beharko da

$$F(f(\overline{y}'),\overline{z}')$$

erako adierazpen bat lortuz. Adierazpen horretan, $\overline{y}'$ eta $\overline{z}'$ kasu errekurtsibo horretan $\overline{y}$ eta $\overline{z}$ aldagaiei esleitu behar zaizkien balio berriak dira. Azkenik, f funtzioak errekurtsio anizkoitza erabiltzen badu, inbariantean f-ren dei errekurtsibo bat baino gehiago egon daiteke. Horren ondorioz, tolesketa egin ahal izateko, destolesten diren inbarianteko dei errekurtsiboak zein izango diren erabaki beharko da.

Atal honetako gainerakoan, metodo honen erabileraren lau adibide deskribatuko ditugu. Adibide horietan, atal honetan aipatu diren ezaugarriak eta xehetasunak dituzten funtzioak agertuko dira, eta, ondorioz, 6.2. ataleko adibidea baino konplexuagoak izango dira. Hasteko, espezifikazio ekuazionaleko adibide bat aztertuko da. Adibide horretako funtzioak bi kasu nabari eta bi kasu errekurtsibo ditu. Gainera, erabiltzen den eragiketa ez-elkarkorra da. Bigarren adibidean errekurtsio bikoitza erabiltzen da. Hirugarren adibideko funtzioak errore-kasu bat du eta eskema orokorreko $\overline{z}$ aldagaiei dagokien aldagairik ez du erabiltzen. Azkenik, laugarren adibidean eskema orokorreko $\overline{z}$ aldagaiei dagozkien aldagaien balioa ez da iterazio guztietan eguneratzen.

6.4.1. Adibidea: Sekuentziak nahastu

Atal honetan transformazio-metodoa *nahastu* funtzioari aplikatuko zaio. Sarrerako datu gisa bi sekuentzia emanda, *nahastu* funtzioak bi sekuentzia horien nahasketa itzuliko du elementuen ordena kontuan hartuz. Sarrerako sekuentziak ordenatuta baldin badaude, orduan *nahastu* funtzioaren emaitza ere sekuentzia ordenatu bat izango da, baina *nahastu* funtzioak sekuentzia ez-ordenatuak ere nahas ditzake. Beraz, sarrerako sekuentziak ordenatuta egotea eskatuko duen aurrebaldintza ez dugu kontuan hartuko metodoa aplikatzerakoan. Kontuan izan metodoak programa baliokideak itzultzen dituenez, eraikiko den programaren emaitza sekuentzia ordenatu bat izango dela, sarrerako sekuentziak ordenatuak baldin badira. Har dezagun *nahastu* funtzioaren honako espezifikazio ekuazional hau:

$$
\begin{array}{l}
\mathit{nahastu}\colon \mathit{sekuentzia}(T) \times \mathit{sekuentzia}(T) \to \mathit{sekuentzia}(T) \\
(1)\quad \mathit{nahastu}(\langle\,\rangle, t) = t \\
(2)\quad \mathit{nahastu}(x \bullet h, t) = \\
\qquad \begin{cases}
x \bullet h & \text{baldin } \mathit{hutsa\_da}(t) \\
x \bullet \mathit{nahastu}(h, t) & \text{baldin } \neg\, \mathit{hutsa\_da}(t) \wedge \\
 & \quad x \leq \mathit{lehena}(t) \\
\mathit{lehena}(t) \bullet & \text{bestela} \\
\quad \mathit{nahastu}(x \bullet h, \mathit{hondarra}(t)) &
\end{cases}
\end{array}
$$

Ikus daitekeenez, *nahastu* funtzioaren definizioak bi kasu nabari eta bi kasu errekurtsibo ditu, eta $\bullet$ eragiketa ez-elkarkorra erabiltzen du.

Espezifikazio ekuazionalari besterik gabe jarraituz, funtzio hori honela inplementa daiteke:

```
function nahastu_er(s,t:  sekuentzia(T))
return n:  sekuentzia(T) is
begin
  if hutsa_da(s) then
    n := t;
  elsif hutsa_da(t) then
    n := s;
  elsif lehena(s) <= lehena(t) then
    n := lehena(s) • nahastu_er(hondarra(s),t);
  else
    n := lehena(t) • nahastu_er(s,hondarra(t));
  end if;
end nahastu_er;
```

$\{\, n = nahastu(s,t) \,\}$

Aurreko ataleko adibidean bezala, destolesketa/tolesketa teknika adierazpen konkretu batean aplikatu behar da errekurrentzia-erlazioa lortzeko. Komenigarria da aukeratutako adierazpenaren destolesketa-urratsetan bi kasu errekurtsiboak erabiltzea, lortuko den errekurrentzia-erlazioa esanguratsua eta orokorra izan dadin. Adibidez, $nahastu(\langle 1,\ 4,\ 6\rangle, \langle 2,\ 3\rangle)$ adierazpena egokia da, bertako elementuen konparaketak egitean kasu errekurtsibo bietako baldintzak beteko baitira. Lehenik, $nahastu(\langle 1,\ 4,\ 6\rangle, \langle 2,\ 3\rangle)$ destolesteko lehenengo kasu errekurtsiboa erabiliko da, $1 \bullet nahastu(\langle 4,\ \rangle, \langle 2,\ 3\rangle)$ lortuz. Bigarrenik, $nahastu(\langle 4,\ 6\rangle, \langle 2, 3\rangle)$ destolesteko bigarren kasu errekurtsiboa erabiliko da, $2 \bullet nahastu(\langle 4,\ 6\rangle, \langle 3\rangle)$ lortuz. Adierazpen osoa $1 \bullet (2 \bullet nahastu(\langle 4,\ 6\rangle, \langle 3\rangle))$ da eta ezin da tolestu $\bullet$ eragiketa erabiliz. Arazoa, $\bullet$ eragiketa ez-elkarkorra izatea da, eta, horregatik, ezin da beste adierazpen bat osatu 1, 2 eta $nahastu(\langle 4,\ 6\rangle, \langle 3\rangle)$ azpiadierazpenak beste era batera elkartuz. Arazoa ebazteko, elkarkorra den beste eragiketa bat erabili behar dugu. Adibide honetan, aurreko adierazpena toles daiteke sekuentzien kateaketa (@) erabiliz, eragiketa hori elkarkorra baita: $\langle 1,2\rangle @ nahastu(\langle 4,\ 6\rangle, \langle 3\rangle)$. Beraz, $\bullet$ eragiketa @-rekin ordezkatuz destolesketa/tolesketa prozesua honelakoa izango da:

$$
\begin{aligned}
&nahastu(\langle 1,\ 4,\ 6\rangle, \langle 2,\ 3\rangle) \\
&\quad = 1 \bullet nahastu(\langle 4,\ 6\rangle, \langle 2,\ 3\rangle) && \text{(destolestu)} \\
&\quad = \langle 1\rangle @ nahastu(\langle 4,\ 6\rangle, \langle 2,\ 3\rangle) && \text{(tolestu)} \\
&\quad = \langle 1\rangle @ (2 \bullet nahastu(\langle 4,\ 6\rangle, \langle 3\rangle)) && \text{(destolestu)} \\
&\quad = \langle 1,\ 2\rangle @ nahastu(\langle 4,\ 6\rangle, \langle 3\rangle) && \text{(tolestu)} \\
&\quad = \langle 1,\ 2\rangle @ (3 \bullet nahastu(\langle 4,\ 6\rangle, \langle\ \rangle)) && \text{(destolestu)} \\
&\quad = \langle 1,\ 2,\ 3\rangle @ nahastu(\langle 4,\ 6\rangle, \langle\ \rangle) && \text{(tolestu)}
\end{aligned}
$$

$= \langle 1, 2, 3 \rangle @ \langle 4, 6 \rangle$
$= \langle 1, 2, 3, 4, 6 \rangle$

Destolesketa/tolesketa prozesuaren emaitza aztertuz, honako errekurrentzia-erlazio hau lor daiteke:

$$nahastu(s,t) = u@nahastu(v,w)$$

Errekurrentzia-erlazio horretan s, t, u, v eta w zenbaki osozko sekuentziak dira, eta, gainera, u, v eta w aldagai berriak dira. Errekurrentzia-erlazio hori iterazioaren inbariantea izango da.

Borne-adierazpena definitzeko, iterazioaren bukaera v eta w aldagaien menpe dagoela kontuan hartuko dugu. Sekuentzia horien luzera zenbat eta txikiagoa izan, orduan eta gertuago egongo da iterazioa bukaeratik. Beraz, honako borne-adierazpen hau egokia da:

$$E \equiv luzera(v) + luzera(w)$$

Hor, $luzera(v)$ eta $luzera(w)$ adierazpenek v eta w sekuentzien osagai kopurua adierazten dute.

Ondoren, inbariantetik abiatuta, programa iteratiboaren atal guztiak eratorriko ditugu:

- Hasieraketa. Iterazioa hasi aurretik inbariantea betetzeko, hurrengo esleipena egokia da:

```
u := <>;
v := s;
w := t;
```

  $nahastu(s,t) = \langle\,\rangle @ nahastu(s,t)$ egiazkoa da eta.

- Bukaera. *nahastu* funtzioaren definizioak bi kasu nabari dituenez, iterazioaren bukaera-baldintza bi kasu nabari horien baldintzen disjuntzioa izango da, funtzioaren parametroak inbarianteko aldagai berriekin ordezkatuz:

$$hutsa\_da(v) \vee hutsa\_da(w)$$

Hau da, iterazioa v edo w aldagaien balioa sekuentzia hutsa denean bukatuko da. Bukaera-baldintza bi kasuren baldintzen disjuntzioa denez, kasu bakoitza bere aldetik aztertu beharko da kasuan kasuko emaitza kalkulatzeko. Horretarako, inbariantean $nahastu(v,w)$ deia kasu bakoitzaren emaitzarekin ordezkatu behar da:

$$hutsa\_da(v) \rightarrow nahastu(s,t) = u@nahastu(\langle\,\rangle,w) = u@w$$
$$hutsa\_da(w) \rightarrow nahastu(s,t) = u@nahastu(v,\langle\,\rangle) = u@v$$

Beraz, n-ri esleitu behar zaion emaitza $u@w$ edo $u@v$ izango da, bukaeran betetzen den baldintzaren arabera:

```
if hutsa_da(v) then
  n := u @ w;
else
  n := u @ v;
end if;
```

- Iterazioaren gorputza. Inbariantean destolesketa/tolesketa teknika erabiliko dugu kasu errekurtsibo bakoitzeko. Lehenengo kasu errekurtsiboaren baldintza $lehena(v) \leq lehena(w)$ da, eta kasu errekurtsibo hori erabiliz inbariantearen destolesketa/tolesketa prozesua honako hau da:

$$\begin{aligned}
&nahastu(s,t)\\
&\quad = u@nahastu(v,w)\\
&\quad = u@(lehena(v) \bullet nahastu(hondarra(v),w)) && \text{(destolestu)}\\
&\quad = u@(\langle lehena(v)\rangle@nahastu(hondarra(v),w)) && \text{(@-ren ordez } \bullet\text{)}\\
&\quad = \underbrace{(u@\langle lehena(v)\rangle)}_{u'}@nahastu(\underbrace{hondarra(v)}_{v'},\underbrace{w}_{w'}) && \text{(tolestu)}
\end{aligned}$$

Beraz, lehenengo kasu errekurtsibotik ateratzen den aldibereko esleipena honako hau izango da:

```
(u,v,w) := (u @ <lehena(v)>,hondarra(v),w);
```

Aurreko esleipenak sekuentziatzeko ordena zuzena honako hau da:

```
u := u @ <lehena(v)>;
v := hondarra(v);
```

Bestela, `v := hondarra(v);` bestea baino lehenago exekutatuko balitz, $lehena(v)$ balioa galduko litzateke.

Bigarren kasu errekurtsiboari dagokionez, haren baldintza $lehena(v) > lehena(w)$ da eta destolesketa/tolesketa prozesua honela gelditukо da:

$$
\begin{aligned}
&nahastu(s,t)\\
&\quad = u@nahastu(v,w)\\
&\quad = u@(lehena(w) \bullet nahastu(v, hondarra(w))) && \text{(destolestu)}\\
&\quad = u@(\langle lehena(w)\rangle @nahastu(v, hondarra(w))) && \text{(@-ren ordez } \bullet\text{)}\\
&\quad = \underbrace{(u@\langle lehena(w)\rangle)}_{u'} @nahastu(\underbrace{v}_{v'}, \underbrace{hondarra(w)}_{w'}) && \text{(tolestu)}\\
&\quad = u' @nahastu(v', w')
\end{aligned}
$$

Hortaz, honako aldibereko esleipen hau lortuko da:

```
(u,v,w) := (u @ <lehena(w)>,v,hondarra(w));
```

Eta sekuentziatzen bada:

```
u := u @ <lehena(w)>;
w := hondarra(w);
```

Bi kasu errekurtsiboetatik ateratako esleipenak elkartuz, iterazioaren gorputza honako hau izango da:

```
if lehena(v) <= lehena(w) then
  u := u @ <lehena(v)>;
  v := hondarra(v);
else
  u := u @ <lehena(w)>;
  w := hondarra(w);
end if;
```

Azkenik, eraikitako funtzio iteratiboa honako hau da:

```
function nahastu_it(s,t:  sekuentzia(T))
return n:  sekuentzia(T) is
  u,v,w:  sekuentzia(T);
begin
  u := <>;
  v := s;
  w := t;
  while  { nahastu(s,t) = u@nahastu(v,w) }
   not ( hutsa_da(v) or hutsa_da(w) )
  loop   { E ≡ luzera(v)+luzera(w) }
    if lehena(v) <= lehena(w) then
      u := u @ <lehena(v)>;
      v := hondarra(v);
    else
      u := u @ <lehena(w)>;
      w := hondarra(w);
    end if;
  end loop;
  if hutsa_da(v) then
    n := u @ w;
  else
    n := u @ v;
  end if;
end nahastu_it;
{ n = nahastu(s,t) }
```

6.4.2. Adibidea: Fibonacci-ren zenbakiak

Adibide honekin, errekurtsio anizkoitza aztertuko dugu. Aurretik aipatu dugunez, errekurtsio anizkoitza erabiltzen denean inbariantean dei errekurtsibo bat baino gehiago egon daiteke. Horren ondorioz, tolesketa egin ahal izateko, destolestuko diren inbarianteko dei errekurtsiboak zein izango diren erabaki behar da. Destolesketa egiteko era guztiek ez dute tolesketa egiteko aukera ematen errekurrentzia-erlazioa edo inbariantea kontserbatuz. Are gehiago, batzuetan errekurrentzia-erlazio bat aurkitzea ezinezkoa da. Kontuan izan edozein errekurrentzia-erlaziotan dei errekurtsiboen kopurua finkoa dela.

Har dezagun fib funtzioaren honako definizio hau:

$$
\begin{array}{l}
\textit{fib}\colon\ oso \to oso \\
\underline{\text{Aurre}}\colon\ n \geq 0 \\
fib(n) = \left\{ \begin{array}{ll} n & \text{baldin } n \leq 1 \\ fib(n-1) + fib(n-2) & \text{bestela} \end{array} \right.
\end{array}
$$

Hor, $fib(n)$ Fibonacciren segidako n-garren elementua izango da. Definizio horretan kasu nabari bakar bat eta kasu errekurtsibo bakar bat daude, eta kasu errekurtsiboan bi dei errekurtsibo egingo dira. Aurreko adibideetan bezala, destolesketa/tolesketa teknika adierazpen konkretu bati aplikatuko diogu errekurrentzia-erlazioa lortzeko. Bi dei errekurtsibo daudenez, destolesketa-estrategia bat aukeratu behar dugu. Adibide honetan, argumentu handieneko dei errekurtsiboa destolestuko dugu. Argumentu txikieneko dei errekurtsiboa destolestuz, edo bi dei errekurtsiboak aldi berean destolestuz, ezin da errekurrentzia-erlazio bat aurkitu, tolestu ondoren ateratzen diren adierazpenetan dei errekurtsiboen kopurua finkoa izatea ez delako lortzen.

$$
\begin{array}{ll}
fib(5) & \\
\quad = fib(4) + fib(3) & \text{(destolestu)} \\
\quad = (fib(3) + fib(2)) + fib(3) & \text{(destolestu)} \\
\quad = 2 * fib(3) + fib(2) & \text{(tolestu)} \\
\quad = 2 * (fib(2) + fib(1)) + fib(2) & \text{(destolestu)} \\
\quad = 3 * fib(2) + 2 * fib(1) & \text{(tolestu)} \\
\quad = 3 * (fib(1) + fib(0)) + 2 * fib(1) & \text{(destolestu)} \\
\quad = 5 * fib(1) + 3 * fib(0) & \text{(tolestu)} \\
\quad = 5 * 1 + 3 * 0 & \\
\quad = 5 &
\end{array}
$$

Aurreko transformazioa kontuan hartuta, hainbat errekurrentzia-erlazio formula daitezke:

$$
\begin{array}{lll}
(1) & fib(n) & = x * fib(u) + y * fib(v) \\
(2) & fib(n) & = x * fib(u) + y * fib(u-1) \\
(3) & fib(n) & = x * fib(v+1) + y * fib(v)
\end{array}
$$

Programa iteratiboa diseinatzeko, azken errekurrentzia-erlazioa hartuko dugu inbariantetzat:

$$INB \equiv fib(n) = x * fib(v+1) + y * fib(v)$$

Interesa duen irakurleak beste programa iteratibo batzuk diseina ditzake lehenengo bi errekurrentzia-erlazioak inbariantetzat hartuz.

Orain arte egin dugun bezalaxe, aukeratutako inbariantetik abiatuta programa iteratiboaren atal guztiak eratorriko ditugu:

- Hasieraketa. Honako esleipen hauek hartuz,

  ```
  x := 0;
  y := 1;
  v := n;
  ```

  iterazioa exekutatzen hasi aurretik inbariantea bete egingo da, $fib(n) = 0 * fib(n+1) + 1 * fib(n)$ egiazkoa baita.

- Bukaera. $v = 0$ berdintza iterazioaren bukaera-baldintzatzat har daiteke, inbarianteko bi dei errekurtsiboak kasu nabariaren bidez ebazten direlako. Inbariantean, $fib(v+1)$ eta $fib(v)$ deiak kasu nabaritik ateratako emaitzarekin ordezkatuz, iterazioaren emaitza kalkulatuko da:

  $$v = 0 \;\rightarrow\; fib(n) = x * fib(0+1) + y * fib(0) = x * 1 + y * 0 = x$$

  Beraz, programa iteratiboaren emaitza itzultzeko, hurrengo esleipena erabiliko da:

  ```
  f := x;
  ```

  Kontuan izan $E \equiv v$ iterazioaren borne-adierazpentzat har daitekeela.

- Iterazioaren gorputza. Inbariantean destolesketa/tolesketa teknika erabiliko dugu argumentu handieneko dei errekurtsiboa destolestuz:

  $$\begin{aligned}
  & fib(n) \\
  & \quad = x * fib(v+1) + y * fib(v) \\
  & \quad = x * (fib(v) + fib(v-1)) + y * fib(v) && \text{(destolestu)} \\
  & \quad = \underbrace{(x+y)}_{x'} * fib(\underbrace{\quad v \quad}) + \underbrace{x}_{y'} * fib(\underbrace{v-1}_{v'}) && \text{(tolestu)} \\
  & \quad = \quad x' \quad * fib(v'+1) + \; y' \; * fib(\; v' \;)
  \end{aligned}$$

  Beraz, hurrengo aldibereko esleipena gelditzuko da:

  ```
  (x,y,v) := (x+y,x,v-1);
  ```

  Aldibereko esleipen hori sekuentziatzeko, aldagai laguntzaile bat erabiliko dugu:

```
    lag := x;
    x := x+y;
    y := lag;
    v := v-1;
```

Laburbilduz, lortu dugun programa iteratiboa honako hau da:

```
function fib_it(n:  Integer) return f:  Integer is
  x,y,v,lag:  Integer;
{ n ≥ 0 }
begin
  x := 0;
  y := 1;
  v := n;
  while  { fib(n) = x*fib(v+1)+y*fib(v) }
    not v = 0
  loop   { E ≡ v }
    lag := x;
    x := x+y;
    y := lag;
    v := v-1;
  end loop;
   f := x;
end fib_it;
{ f = fib(n) }
```

6.4.3. Adibidea: Zenbaki arruntezko sekuentzia bateko elementu handiena

Ondoren, zenbaki arruntezko sekuentzia ez-huts bateko elementu handiena itzultzen duen funtzioa hartuko dugu. Izan bedi funtzio horren honako espezifikazio ekuazional hau:

$handiena$: $sekuentzia(nat) \to nat$

(1) $handiena(\langle\,\rangle) =$ `errorea`

(2) $handiena(x \bullet s) =$

$$\begin{cases} x & \text{baldin } hutsa\_da(s) \\ handiena(x \bullet hondarra(s)) & \text{baldin } \neg\, hutsa\_da(s) \wedge \\ & \quad x > lehena(s) \\ handiena(s) & \text{bestela} \end{cases}$$

Ikus daitekeenez, funtzioaren definizioak errore-kasu bat du. Kasu hori kendu egingo dugu eta aurrebaldintza bezala sarrerako sekuentzia ez-hutsa izatea

ezarriko dugu. Hortaz, kasu nabari bat eta bi kasu errekurtsibo geldituko dira. Funtzio horren inplementazio errekurtsiboa honako hau da:

```
function handiena_er(s:  sekuentzia(Natural))
return e:  Natural is
{ ¬ hutsa_da(s) }
begin
  if hutsa_da(hondarra(s)) then
    e := lehena(s);
  elsif lehena(s) > lehena(hondarra(s)) then
    e := handiena_er(x • hondarra(hondarra(s)));
  else
    e := handiena_er(hondarra(s));
  end if;
end handiena_er;
{ e = handiena(s) }
```

Hasteko, destolesketa/tolesketa teknika adierazpen konkretu batean aplikatuko da errekurrentzia-erlazioa lortzeko. Destolesketa-urratsetan kasu errekurtsibo biak erabili beharko direnez, $handiena(\langle 6,\ 5,\ 8,\ 2,\ 7\rangle)$ adierazpena egokia da horretarako. Destolesketa/tolesketa prozesua honako hau izango litzateke:

$$
\begin{aligned}
&handiena(\langle 6,\ 5,\ 8,\ 2,\ 7\rangle)\\
&\quad = handiena(\langle 6,\ 8,\ 2,\ 7\rangle) && \text{(destolestu)}\\
&\quad = handiena(\langle 8,\ 2,\ 7\rangle) && \text{(destolestu)}\\
&\quad = handiena(\langle 8,\ 7\rangle) && \text{(destolestu)}\\
&\quad = handiena(\langle 8\rangle)\\
&\quad = 8
\end{aligned}
$$

Destolesketa-urratsen ondoren lortutako adierazpenek hasierako adierazpenaren egitura dute. Beraz, tolesketa-urratsak ez dira beharrezkoak eta lortutako errekurrentzia-erlazioa honako hau da:

$$handiena(s) \;=\; handiena(w)$$

Hor, s eta w zenbaki arruntezko sekuentziak dira, eta, gainera, w aldagai berria da. Kontuan izan adibide honetan $\bullet$ eragiketak sekuentzien nahasketaren adibidean aipatutako arazoa ez duela sortzen (ikus 6.4.1. atala). Lortutako errekurrentzia-erlazioa inbariantetzat hartuko dugu programa iteratiboan.

Gainera, aurreko destolesketa-prozesuan ikus daitekeenez, dei errekurtsiboaren sarrerako sekuentziaren luzera gero eta txikiagoa izango da. Beraz,

$$E \equiv luzera(w)$$

borne-adierazpentzat har daiteke. Hor, $luzera(w)$ adierazpenak w sekuentziaren osagai kopurua adierazten du.

Inbariantetik abiatuta, programa iteratiboaren osagai guztiak eratorriko ditugu:

- Hasieraketa. Hurrengo esleipenari esker

  ```
  w := s;
  ```

  iterazioa exekutatu aurretik inbariantea bete egingo dela ziurta daiteke; izan ere, $handiena(s) = handiena(s)$ egiazkoa da.

- Bukaera. Funtzioaren definizioak kasu nabari bakar bat duenez, iterazioa bukatzeko baldintza kasu nabari horren baldintza izango da parametroa inbariantean ipinitako aldagai berriarekin ordezkatuz:

  $$hutsa\_da(hondarra(w))$$

  Beraz, iterazioaren exekuzioa w sekuentzia elementu bakarrekoa denean bukatuko da. Emaitza kalkulatuko da inbariantean $handiena(w)$ deia kasu nabariak itzulitako balioarekin ordezkatuz:

  $$hutsa\_da(hondarra(w)) \rightarrow handiena(s) = handiena(w) = lehena(w)$$

  Hortaz, funtzio iteratiboaren emaitza $lehena(w)$ da:

  ```
  e := lehena(w);
  ```

- Iterazioaren gorputza. Inbariantean destolesketa/tolesketa teknika erabiliko dugu kasu errekurtsibo bakoitzeko.

  Lehenengo kasu errekurtsiboa: $lehena(w) > lehena(hondarra(w))$

  $$\begin{aligned} &handiena(s) \\ &\quad = handiena(\underbrace{lehena(w) \bullet hondarra(hondarra(w))}_{w'}) \quad \text{(destolestu)} \\ &\quad = handiena(\quad w' \quad) \end{aligned}$$

  Beraz, lehenengo kasu errekurtsiboari dagokion esleipena honako hau da:

  ```
  w := lehena(w) • hondarra(hondarra(w));
  ```

Bigarren kasu errekurtsiboa: $lehena(w) \leq lehena(hondarra(w))$

$$
\begin{aligned}
&handiena(s) \\
&\quad = handiena(\underbrace{hondarra(w)}_{w'}) && \text{(destolestu)} \\
&\quad = handiena(\quad w' \quad)
\end{aligned}
$$

Hortaz, bigarren kasu errekurtsibotik lortzen den esleipena honako hau da:

```
w := hondarra(w);
```

Aurreko bi esleipenak elkartuz, iterazioaren gorputza lortuko da:

```
if lehena(w) > lehena(hondarra(w)) then
  w := lehena(w) • hondarra(hondarra(w));
else
  w := hondarra(hondarra(w));
end if;
```

Ondorioz, eraikitako funtzio iteratiboa honako hau da:

```
function handiena_it(s:  sekuentzia(Natural))
return e:  Natural is
  w:  sekuentzia(Natural);
{ ¬ hutsa_da(s) }
begin
  w := s;
  while  { handiena(s) = handiena(w) }
   not (hutsa_da(hondarra(w)))
  loop   { E ≡ luzera(w) }
    if lehena(w) > lehena(hondarra(w)) then
      w := lehena(w) • hondarra(hondarra(w));
    else
      w := hondarra(w);
    end if;
  end loop;
  e := lehena(w);
end handiena_it;
{ e = handiena(s) }
```

6.4.4. Adibidea: Sekuentzia batetik elementu bat ezabatu

x elementua eta s sekuentzia emanda, *ezabatu* funtzioak s sekuentziatik x-ren agerpen guztiak ezabatuko ditu. Har dezagun *ezabatu* funtzioaren honako espezifikazio ekuazional hau:

$$\begin{array}{ll} & ezabatu\text{: } T \times sekuentzia(T) \to sekuentzia(T) \\ (1) & ezabatu(x, \langle\,\rangle) = \langle\,\rangle \\ (2) & ezabatu(x, y \bullet s) = \begin{cases} ezabatu(x,s) & \text{baldin } x = y \\ y \bullet ezabatu(x,s) & \text{bestela} \end{cases} \end{array}$$

Hor ikus daitekeen bezala, *ezabatu* funtzioaren definizioak kasu nabari bat eta bi kasu errekurtsibo ditu. Funtzio hori honela inplementa daiteke errekurtsiboki:

```
function ezabatu_er(x:  T, s:  sekuentzia(T))
return e:  sekuentzia(T) is
begin
  if hutsa_da(s) then
    e := <>;
  elsif x = lehena(s) then
    e := ezabatu_er(x, hondarra(s));
  else
    e := lehena(s) • ezabatu_er(x, hondarra(s));
  end if;
end ezabatu_er;
```

$\{\, e = ezabatu(x,s) \,\}$

Orain bertsio iteratibo bat eraikiko dugu. Hasteko, destolesketa/tolesketa teknika aplikatuko dugu $ezabatu(5, \langle 1,\ 5,\ 6,\ 5,\ 8\rangle)$ adierazpenean. Horretarako, $\bullet$ eragiketa ez-elkarkorra elkarkorra den sekuentzien kateaketarekin ($@$) ordezkatuko da (6.4.1. ataleko adibidean egin dugun bezala). Destolesketa/tolesketa prozesua honako hau izango da:

$$\begin{array}{ll} ezabatu(5, \langle 1,\ 5,\ 6,\ 5,\ 8\rangle) & \\ \quad = 1 \bullet ezabatu(5, \langle 5,\ 6,\ 5,\ 8\rangle) & \text{(destolestu)} \\ \quad = \langle 1\rangle @ ezabatu(5, \langle 5,\ 6,\ 5,\ 8\rangle) & \text{(tolestu)} \\ \quad = \langle 1\rangle @ ezabatu(5, \langle 6,\ 5,\ 8\rangle) & \text{(destolestu)} \\ \quad = \langle 1\rangle @ (6 \bullet ezabatu(5, \langle 5,\ 8\rangle)) & \text{(destolestu)} \\ \quad = \langle 1,\ 6\rangle @ ezabatu(5, \langle 5,\ 8\rangle) & \text{(tolestu)} \\ \quad = \langle 1,\ 6\rangle @ ezabatu(5, \langle 8\rangle) & \text{(destolestu)} \\ \quad = \langle 1,\ 6\rangle @ (8 \bullet ezabatu(5, \langle\,\rangle)) & \text{(destolestu)} \\ \quad = \langle 1,\ 6,\ 8\rangle @ ezabatu(5, \langle\,\rangle) & \text{(tolestu)} \end{array}$$

$= \langle 1,\ 6,\ 8 \rangle @\langle\ \rangle$
$= \langle 1,\ 6,\ 8 \rangle$

Hor ikus daitekeen bezala, lehenengo kasu errekurtsiboa erabiltzen duten destolesketa-urratsen ondoren tolesketa-urratsik ez da behar, lortutako adierazpenak inbariantearen egitura baitu. Destolesketa/tolesketa prozesu horretatik honako errekurrentzia-erlazioa lortuko da:

$$ezabatu(x, s) \ = \ u@ezabatu(x, w)$$

Hor, x elementua T motakoa da, s, u eta w T motako sekuentziak dira, eta u eta w aldagai berriak dira.

Aurreko adibidean gertatzen den bezala, destolesketa/tolesketa prozesuan dei errekurtsiboen sarrerako argumentua gero eta txikiagoa da. Horren ondorioz,

$$E \equiv luzera(w)$$

borne-adierazpentzat har daiteke.

Inbariantea oinarritzat hartuta, iterazioaren osagai guztiak honela eratorriko dira:

- Hasieraketa. Honako esleipen hauei esker

  ```
  u := <>;
  w := s;
  ```

  iterazioa exekutatu aurretik inbariantea beteko dela ziurta daiteke; izan ere, $ezabatu(x, s) \ = \ \langle\ \rangle @ezabatu(x, s)$ berdintza egiazkoa da.

- Bukaera. Funtzioaren definizioak kasu nabari bakar bat duenez, iterazioa bukatzeko baldintza kasu nabariaren baldintza izango da funtzioaren argumentua inbariantean ipinitako aldagai berriarekin ordezkatuz:

  $$hutsa\_da(w)$$

  Hortaz, iterazioaren exekuzioa w sekuentzia hutsa denean bukatuko da. Funtzio iteratiboaren emaitza inbariantean $ezabatu(x, w)$ deia kasu nabariak itzulitako balioarekin ordezkatuz kalkulatuko da:

  $$hutsa\_da(w) \rightarrow ezabatu(x, s) \ = \ u@ezabatu(x, \langle\ \rangle) \ = \ u@\langle\ \rangle \ = \ u$$

  Beraz, emaitza u izango da:

  ```
  e := u;
  ```

- Iterazioaren gorputza. Funtzioaren definizioak bi kasu errekurtsibo dituenez, destolesketa/tolesketa teknika inbariantean aplikatuko da bi kasu errekurtsiboak kontuan hartuz.

  Lehenengo kasu errekurtsiboa: $x = lehena(w)$

  $$\begin{array}{ll} ezabatu(x,s) & \\ \quad = u@ezabatu(x,w) & \\ \quad = \underbrace{u}_{u'}\ @ezabatu(x,\underbrace{hondarra(w)}_{w'}) & \text{(destolestu)} \\ \quad = u'\ @ezabatu(x,w') & \end{array}$$

  Beraz, lehenengo kasu errekurtsibotik honako aldibereko esleipena lortuko da:

  ```
  (u,w) := (u,hondarra(w));
  ```

  Aldibereko esleipen hori honako esleipen honen baliokidea da:

  ```
  w := hondarra(w);
  ```

  Bigarren kasu errekurtsiboa: $x \neq lehena(w)$

  $$\begin{array}{ll} ezabatu(x,s) & \\ \quad = u@ezabatu(x,w) & \\ \quad = u@(lehena(w) \bullet ezabatu(x, hondarra(w))) & \text{(destolestu)} \\ \quad = u@(\langle lehena(w)\rangle @ezabatu(x, hondarra(w))) & \text{(@-ren ordez } \bullet\text{)} \\ \quad = \underbrace{(u@\langle lehena(w)\rangle)}_{u'}\ @ezabatu(x,\underbrace{hondarra(w)}_{w'}) & \text{(tolestu)} \\ \quad = u'\ @ezabatu(x,w') & \end{array}$$

  Hortaz, bigarren kasu errekurtsiboari dagokion aldibereko esleipena honako hau da:

  ```
  (u,w) := (u @ <lehena(w)>,hondarra(w));
  ```

  Aldibereko esleipen hori sekuentziatu ondoren honako esleipenak geldituko dira:

  ```
  u := u @ <lehena(w)>;
  w := hondarra(w);
  ```

Kontuan izan `w := hondarra(w);` esleipena bestea baino lehenago exekutatuko balitz, behar dugun $lehena(w)$ balioa galdu egingo litzatekeela.

Aurreko esleipenak elkartuz, iterazioaren gorputza honela gelditu ko da:

```
if x = lehena(w) then
  w := hondarra(w);
else
  u := u @ <lehena(w)>;
  w := hondarra(w);
end if;
```

Laburbilduz, lortu den funtzio iteratiboa honako hau da:

```
function ezabatu_it(x:  T, s:  sekuentzia(T))
return e:  sekuentzia(T) is
  u,w:  sekuentzia(T);
begin
  u := <>;
  w := s;
  while  { ezabatu(x,s) = u@ezabatu(x,w) }
   not ( hutsa_da(w) )
  loop   { E ≡ luzera(w) }
    if x = lehena(w) then
      w := hondarra(w);
    else
      u := u @ <lehena(w)>;
      w := hondarra(w);
    end if;
  end loop;
  e := w;
end ezabatu_it;
{ e = ezabatu(x,s) }
```

6.5. BIBLIOGRAFIA-OHARRAK

R. M. Burstall eta J. Darlington ([23]) izan ziren funtzio errekurtsiboen transformazioan eta errekurtsibotik iteratiborako bihurketan aitzindariak. Lan horretan, transformazio-erregela batzuk aurkezten dira, kapitulu honen oinarriak diren destolesketa- eta tolesketa-erregelak barne. Proposamen bera erabili zuten programa-eskemak transformatzeko [27] artikuluan. Handik

aurrera, errekurtsibotik iteratiborako bihurketari buruzko lan asko egon dira helburu desberdinekin: programazio eta errekurtsibotik iteratiborako bihurketaren arloan ez ezik, konpilazioan, kode-optimizazioan, programa-analisian, ebaluazio partzialean eta programa-sintesian ere erabiltzen da destolesketa/tolesketa teknika. Besteak beste, [7, 8, 22, 64, 77] lanak aipa daitezke. Horrez gain, destolesketa/tolesketa teknika beste paradigma batzuetara zabaldu eta egokitu zen. Adibidez, [88] lanean destolesketa/tolesketa teknika programazio logikoaren paradigmara egokituta dago, eta [82, 83] lanetan teknikaren aldaera hori sakonki aztertzen da.

Ikuspegi didaktikotik, errekurtsibotik iteratiborako beste transformazio-metodo mota batzuk ere oso interesgarriak dira. Adibidez, dei-zuhaitza zeharkatzeko era simulatzen duten metodoak. Metodo horiek erabilgarriak dira errekurrentzia-erlazioa definitu ezin dugunean. [86] lan aitzindarian metodo mota hori era xehatuan aurkezten da.

Gaur egun, programen transformazioa —errekurtsibotik iteratiborako bihurketa barne— *birfaktorizazio* izenarekin ezagutzen diren tekniken barruan sailkatzen da ([42]). Teknika horien helburua kode-transformazioa da, kodearen barne-egitura aldatuz baina haren funtzionaltasuna edo jokabideari eutsiz. Gaur egun, garapen-ingurune integratu gehienek (ingelesez, IDE edo *integrated development environment*) birfaktorizazio-tresna automatikoak eskaintzen dituzte programazio-lengoaia desberdinetarako, besteak beste Eclipse[3] eta Visual Studio lengoaietarako[4].

6.6. ARIKETAK: ERREKURTSIBOTIK ITERATIBORAKO TRANSFORMAZIOA

1. Osatu *bitarra* funtzio errekurtsiboa iteratibo bihurtzeko garatu den honako ebazpen hau. Garapena destolestu/tolestu metodoari jarraituz egin da, ebazpenean proposatutako errekurrentzia-erlazioa erabiliz:

$$bitarra(x) = \begin{cases} x & \text{baldin } 0 \leq x \leq 1 \\ 10 * bitarra(x/2) & \text{baldin } bikoitia(x) \wedge x \geq 2 \\ 10 * bitarra(x/2) + 1 & \text{baldin } bakoitia(x) \wedge x \geq 2 \end{cases}$$

bitarra funtzioak zenbaki arrunt baten adierazpen bitarra kalkulatuko du.

Errekurrentzia-erlazioa:

$$bitarra(x) = u * bitarra(v) + w$$

3. `https://eclipse.org`
4. `https://www.visualstudio.com`

Hasieraketa:

$$bitarra(x) = u * bitarra(v) + w = \_\_\_\_ * bitarra(\_\_\_\_) + 0$$

```
u := ____;
v := ____;
w := 0;
```

Bukaera. Kasu nabari bakarra dago:

$$0 \leq v \leq 1 \;\rightarrow\; bitarra(x) \;=\; u * bitarra(v) + w \;=\; u * \_\_\_\_ + w$$

Beraz,

```
e := _______;
```

Bukaerako baldintza.

$$bukaera\_baldintza(u, v, w) \;\equiv\; 0 \leq v \leq 1$$

Iterazioaren gorputza. Bi kasu induktibo daude.
Lehenengo kasu induktiboa ($bikoitia(v) \wedge v \geq 2$):

$$\begin{aligned}
&bitarra(x)\\
&\quad = u * bitarra(v) + w\\
&\quad = u * (10 * bitarra(\tfrac{v}{2})) + w && \text{(destolestu)}\\
&\quad = \underbrace{_____}_{u'} * bitarra(\underbrace{\frac{v}{2}}_{v'}) + \underbrace{w}_{w'} && \text{(tolestu)}
\end{aligned}$$

Aldibereko esleipena:

```
(u,v,w) := (_______,v/2,w);
```

Aldibereko esleipenaren inplementazioa:

```
u := _______;
v := v/2;
```

Bigarren kasu induktiboa ($bakoitia(v) \wedge v \geq 2$):

$$
\begin{array}{ll}
bitarra(x) & \\
\quad = u * bitarra(v) + w & \\
\quad = \_\_\_\_\_ + w & \text{(destolestu)} \\
\quad = \underbrace{\_\_\_\_\_}_{u'} * bitarra(\_\_\_\_\_) + \underbrace{\_\_\_\_\_}_{w'} & \text{(tolestu)} \\
\quad = \underbrace{\quad u' \quad}_{} * bitarra(\quad v' \quad) + \underbrace{\quad w' \quad}_{} &
\end{array}
$$

Aldibereko esleipena:

```
(u,v,w) := (____,____,____);
```

Aldibereko esleipenaren inplementazioa:

```
w := ____;
u := ____;
v := ____;
```

Funtzio iteratiboa:

```
function bitarra_it(x:  Integer) return e:  Integer is
{ x ≥ 0 }
  u, v, w :  Integer;
begin
  u := ____;
  v := ____;
  w := 0;
  while  { INB ≡ bitarra(x) = u * bitarra(v) + w }
   v > 1
  loop
    if bikoitia(v) then
      u := ____;
      v := ____;
    else
      w := ____;
      u := ____;
      v := ____;
    end if;
  end loop;
  e := _______;
end bitarra_it;
{ e = bitarra(x) }
```

2. Erabili proposatutako errekurrentzia-erlazioa honako funtzio errekurtsibo hauetako bakoitza destolestu/tolestu metodoaren bidez iteratibo bihurtzeko.

2.1. *agerkop* funtzioak x digitua ($0 \leq x \leq 9$) y zenbaki arruntean zenbat aldiz agertzen den kalkulatuko du:

$$agerkop(x,y) = \begin{cases} 1 & \text{baldin } x = y \\ 0 & \text{baldin } 0 \leq y \leq 9 \wedge \\ & \quad x \neq y \\ agerkop(x, y/10) + 1 & \text{baldin } y \geq 10 \wedge \\ & \quad y \,\%\, 10 = x \\ agerkop(x, y/10) & \text{baldin } y \geq 10 \wedge \\ & \quad y \,\%\, 10 \neq x \end{cases}$$

Errekurrentzia-erlazioa:

$$agerkop(x,y) = agerkop(x,u) + v$$

2.2. *hur_baino_handi*: $sekuentzia(nat) \rightarrow nat$ funtzioak sekuentzia batean hurrengo elementua baino handiagoak diren zenbat elementu dauden kalkulatuko du.

$$\begin{array}{ll} (1) & hur\_baino\_handi(\langle\,\rangle) = 0 \\ (2) & hur\_baino\_handi(\langle x \rangle) = 0 \\ (3) & hur\_baino\_handi(x \bullet (y \bullet r)) = \\ & \qquad \begin{cases} 1 + hur\_baino\_handi(y \bullet r) & \text{baldin } x > y \\ hur\_baino\_handi(y \bullet r) & \text{bestela} \end{cases} \end{array}$$

Errekurrentzia-erlazioa:

$$hur\_baino\_handi(s) = hur\_baino\_handi(u) + v$$

Hor, s aldagaia $sekuentzia(nat)$ motakoa izango da eta espezifikazio ekuazionaleko hiru kasuak —$\langle\,\rangle$, $\langle x \rangle$ eta $x \bullet (y \bullet r)$— orokortuko ditu.

Laguntza: Espezifikazioko kasu nabari biak $luzera(s) \leq 1$ kasu nabari bakarrean bildu beharko dira; izan ere, $\neg B \equiv hutsa\_da(s)$ or $hutsa\_da(hondarra(s))$ bukaera-baldintza ez da zuzena, inbarianteak ez baitu $def(B)$ inplikatzen.

2.3. $f : nat^+ \times nat^+ \to nat^+$ funtzioa[5]:

$$f(m,n) = \begin{cases} 1 & \text{baldin } n = m \\ \frac{(n+1)*f(m,n+1)}{(m-n)} & \text{baldin } n < m \end{cases}$$

Errekurrentzia-erlazioa:

$$f(m,n) = \frac{u * f(m,v)}{w}$$

2.4. Errekurtsibitate bikoitzekoa den $f : nat^+ \to nat^+$ funtzioa:

$$f(n) = \begin{cases} n & \text{baldin } 1 \leq n \leq 2 \\ f(n-3) & \text{baldin } n \geq 3 \wedge bikoitia(n) \\ f(n-1) + f(n-2) & \text{baldin } n \geq 3 \wedge bakoitia(n) \end{cases}$$

Errekurrentzia-erlazioa:

$$f(n) = x * f(2*z) + y * f(2*z-1)$$

Laguntza: hasieraketan bikoitiak eta bakoitiak bereizi behar dira.

3. Bihurtu iteratibo honako funtzio errekurtsibo hauek destolestu/tolestu metodoa erabiliz:

3.1. *mugkop* funtzioak, $n \geq 1$ (disko kopurua) emanda, Hanoiko Dorreen problema n diskorentzat ebazteko egin behar diren mugimenduen kopurua kalkulatuko du.

$$mugkop(n) = \begin{cases} 1 & \text{baldin } n = 1 \\ 2 * mugkop(n-1) + 1 & \text{bestela} \end{cases}$$

3.2. *zkh* funtzioak osoak eta positiboak diren a eta b zenbakien zatitzaile komunetako handiena kalkulatuko du.

$$zkh(a,b) = \begin{cases} 1 & \text{baldin } a = 1 \vee b = 1 \\ a & \text{baldin } a = b \\ zkh(a-b,b) & \text{baldin } a > b \\ zkh(a,b-a) & \text{baldin } a < b \end{cases}$$

5. Teknikoki ondo datorkigulako, positiboak diren zenbaki arrunten datu-mota erabiliko dugu. Mota hori nat^+ bezala adieraziko dugu eta zero ez eta beste zenbaki arrunt guztiak izango ditu. Mota horren ordez *nat* erabiliko bagenu, hirugarren argumentua zero denerako errore-ekuazio bat definitu beharko genuke. Aldiz, nat^+ erabiliz ekuazio hori saihestuko dugu. nat^+ motaren espezifikazio aljebraikoa *nat* motarenaren antzekoa da. Eragiketa eraikitzaile bezala, *bat* eta *hurrengoa* erabil daitezke, esate baterako.

3.3. *zkh3* funtzioak osoak eta positiboak diren a, b eta c zenbakien zatitzaile komunetako handiena kalkulatuko du, $a \geq b \geq c$ izanda.

$$zkh3(a,b,c) = \begin{cases} a & \text{baldin } a = b = c \\ zkh3(a-c,b,c) & \text{baldin } a - c \geq b \\ zkh3(b,a-c,c) & \text{baldin } a - c \geq c \\ zkh3(b,c,a-c) & \text{bestela } (b \geq c > a - c) \end{cases}$$

3.4. n, *oin* eta e zenbaki osoak emanda, eta zenbaki horiek $z \geq 0$, $2 \leq oin \leq 9$ eta $e \geq 0$ baldintzak betetzen dituztela jakinda, $oin10$ funtzioak *oin* oinarrian dagoen z zenbakia hamar oinarrira bihurtuko du eta oin^e zenbakiarekin biderkatuko du. Beraz, e zero denean z zenbakia hamar oinarrira bihurtuko du.

$$oin10(n,oin,e) = \begin{cases} z * oin^e & \text{baldin } z < oin \\ (z \,\%\, 10) * oin^e + & \text{bestela} \\ \quad oin10(z/10,oin,e+1) & \end{cases}$$

3.5. $alderantzizkoa : sekuentzia(T) \to sekuentzia(T)$ funtzioak sekuentzia baten alderantzizkoa kalkulatuko du.

(1) $alderantzizkoa(\langle\,\rangle) = \langle\,\rangle$
(2) $alderantzizkoa(x \bullet s) = alderantzizkoa(s) @ \langle x \rangle$

3.6. *log_hurb* funtzioak x zenbakiaren logaritmoaren hurbilpen osoa *oin* oinarrian kalkulatuko du, $oin \geq 2$ eta $x \geq 1$ baldintzak betetzen badira.

$$log\_hurb(oin,x) = \begin{cases} 0 & \text{baldin } x = 1 \vee \\ & \quad x < oin \\ log\_hurb(oin,x/oin) + 1 & \text{bestela} \end{cases}$$

3.7. *aldikop*: $T \times sekuentzia(T) \to nat$ funtzioak elementu bat sekuentzia batean zenbat aldiz agertzen den zenbatuko du.

(1) $aldikop(x,\langle\,\rangle) = 0$
(2) $aldikop(x, y \bullet s) = \begin{cases} 1 + aldikop(x,s) & \text{baldin } y = x \\ aldikop(x,s) & \text{bestela} \end{cases}$

3.8. *bitan_zatitu*: $T \times sekuentzia(T) \to sekuentzia(T)$ funtzioak sekuentzia batean bi zati sortuko ditu, emandako elementu bat baino txikiagoak edo berdinak direnak hasieran ipiniz eta handiagoak bukaeran ipiniz.

$$
\begin{array}{ll}
(1) & bitan\_zatitu(x, \langle\,\rangle) = \langle\,\rangle \\
(2) & bitan\_zatitu(x, y \bullet s) = \\
& \quad \begin{cases} y \bullet bitan\_zatitu(x,s) & \text{baldin } y \leq x \\ bitan\_zatitu(x,s) @ \langle y \rangle & \text{bestela} \end{cases}
\end{array}
$$

3.9. *irauli*: $pila(T) \times pila(T) \to pila(T)$ funtzioak pila bat beste baten gainean irauliko du.

$$
\begin{array}{ll}
(1) & irauli(hutsa, q) = q \\
(2) & irauli(pilaratu(x,p), q) = irauli(p, pilaratu(x,q))
\end{array}
$$

3.10. *azken_in*: $zuhbit(T) \to T$ funtzioak zuhaitz bitar bat inordenan zeharkatuz lortzen den azkeneko adabegia itzuliko du.

$$
\begin{array}{ll}
(1) & azken\_in(hutsa) = \texttt{errorea} \\
(2) & azken\_in(errotu(e, ezk, esk) = \\
& \quad \begin{cases} e & \text{baldin } hutsa\_da(esk) \\ azken\_in(esk) & \text{bestela} \end{cases}
\end{array}
$$

3.11. *lehen_post*: $zuhbit(T) \to T$ funtzioak zuhaitz bitar bat postordenan zeharkatuz lortzen den lehenengo adabegia itzuliko du.

$$
\begin{array}{ll}
(1) & lehen\_post(hutsa) = \texttt{errorea} \\
(2) & lehen\_post(errotu(e, ezk, esk)) = \\
& \quad \begin{cases} e & \text{baldin } hutsa\_da(ezk) \wedge \\ & \quad\; hutsa\_da(esk) \\ lehen\_post(ezk) & \text{baldin } \neg hutsa\_da(ezk) \\ lehen\_post(esk) & \text{bestela} \end{cases}
\end{array}
$$

7. Programa iteratiboen eratorpena

Programen eratorpen formala softwarea garatzeko metodo zorrotz bat da eta programen egiaztapenerako teknikak era eraikitzailean erabiltzea du oinarri. Beste era batera esanda, programaren zuzentasuna formalki justifikatzeko balio duen arrazoibide logiko matematikoa oinarritzat hartuz eraikiko da programa, eta, hori horrela izanda, programa eta haren zuzentasunaren frogapena aldi berean lortuko dira. Planteamendu hau guztiz kontrajartzen zaio programatzaileak aurretik duen ezagutza euskarri bakartzat hartuz programa diseinatu eta, programaren garapena bukatu ondoren, lortutako programa hori zuzena ote den egiaztatzean datzan planteamenduari. Garatutako softwarearen fidagarritasuna nabarmen handitzen da programen eratorpen formalaren bidez, eta, gainera, fidagarritasun horren oinarriak diren propietateen aztarna gelditzen da.

Kapitulu honen helburua iterazioen eratorpen formala aurkeztea da. Hala ere, programen eratorpen formalaren metodoaren ideia orokorrak aurkeztuko dira lehendabizi (7.1. atalean). Iterazioak eratortzeko metodoaren urratsen aurkezpen xehatua egingo da 7.2. atalean, eta metodo hori adibide erraz bati aplikatuko zaio berehala. Iterazio bat eratortzeko inbariante bat idatzi beharko denez, 7.4. atalean inbarianteak idazteko erabilgarriak diren jarraibide batzuk azalduko ditugu. 7.2. atalean azaldutako metodoari jarraituz formalki eratorritako iterazioen adibideak aurkeztuko dira 7.5., 7.6., 7.8., eta 7.9. ataletan. 7.8.2. azpiatalean aurkeztuko den adibidea garatu ahal izateko, 7.8.1. atalean bektoreen osagaiei egindako esleipenei dagokien axioma aurkeztuko da. Ariketak dituzten lau (azpi)atal motz ere tartekatu dira (7.3., 7.5.4., 7.7. eta 7.8.3.). Ariketa horiek beraien aurretik dauden adibideak landu ondoren egiteko oso egokiak dira. Kapituluan zehar landutakoa laburbiltzen duten ariketez osatutako zerrenda luzeagoa dator 7.11. atalean. Azkeneko atalean (7.10.), programen eratorpen formalaren sorrera, bilakaerari buruzko datuak eta zerikusia duen bibliografia bildu dira.

7.1. PROGRAMEN ERATORPEN FORMALERAKO METODOA

Programa bat formalki eratortzeko abiapuntua programak egin beharko duen kalkulua formalizatzen duen aurre-ondoetako espezifikazioa izango da. Aurrebaldintza betetzen duen edozein egoeratatik abiatuz postbaldintza betetzen duen egoeraren batean bukatzen den programa bat diseinatzea izango da helburua. Lortu nahi den emaitza kalkulatzeko balio duen estrategia edo algoritmo batera bideratu beharko gaitu espezifikazioaren, eta, batez ere, postbaldintzaren azterketak. Sarrerako egoeratik irteerako egoera batera iristeko egin beharreko aldaketa burutzeko balio duen oinarrizko agindua esleipena da. Batzuetan esleipen-segida bat beharko da. Bururatutako estrategia kasu-azterketan oinarrituta ere egon daiteke, baldintzazko aginduak eta kasu desberdinak bereiziko dituzten asertzioak beharko direlarik. Problema ekintza desberdinen konposaketa sekuentzialaren bidez ebaztea ere gerta daiteke, ekintza bakoitza dagozkion tarteko asertzioen bidez espezifikatu ondoren. Beste batzuetan estrategia izan daiteke ekintza batzuk behin eta berriz errepikatzea, programa iteratibo bat lortuz. Estrategia iteratiboa baldin bada, estrategia hori inbariante baten eta E borne-adierazpen baten bidez formalizatu beharko da. Estrategia pentsatu ondoren, estrategia hori formalizatzen duten asertzio guztiak eta Hoareren sistema formala bera erabiliz zuzena den programa bat sortuko da era ordenatu, arrazoitu eta dokumentatuan. Programa diseinatzean jarraitutako estrategia eta arrazoibidearen aztarna era zehatzean jasoko dituen dokumentazioa sortuko du, gainera, metodoak. Garatutako softwarearen mantentze-faserako oso baliotsua izango da dokumentazio hori. Jarraian datozen bi azpiataletan (7.1.1. eta 7.1.2.) esleipenen konposaketa sekuentzial baten eta baldintzazko konposaketa baten eratorpen-adibideak garatuko ditugu. Ohartu beharrekoa da iterazioen gorputzak, hasieraketak eta abar, esleipen-segidez eta baldintzazko aginduz eratuta egon ohi direla gehienetan.

7.1.1. Adibidea: Bi aldagai indibidualen balioen trukaketa

Esleipenen konposaketa sekuentzialaren adibide bat garatuko da urratsez urrats azpiatal honetan eta, besteak beste, esleipenen eranste progresiboa, esleipen berri bakoitza erantsi ondoren kalkulatutako asertzioaren bidez formalizatutako helburu berria eta egokiak ez diren erabakiak detektatzeko era erakutsiko dira. x eta y bi aldagai indibidual emanda, beraien balioa trukatuko duen programa eratorriko dugu. Azpiatalaren bukaeran, kodearen eta espezifikazioaren berrerabilgarritasuna eta modularitatea aztertzeko erabiliko dugu adibide hau.

Problema honi dagokion aurre-ondoetako espezifikazioa honako hau da:

$$\varphi \equiv x = a \wedge y = b$$
$$\psi \equiv x = b \wedge y = a$$

Espezifikazio horretan a eta b-ren bidez x eta y-ren hasierako balioak adierazten dira. Balio-trukaketa esleipenak bakarrik erabiliz egingo denez, soluzioak ez du estrategia iteratiborik behar. Eratorpen-metodo honetan postbaldintza da zein agindu erantsi behar diren erabakitzen laguntzen duena. Beraz, badakigu $x = a \wedge y = b$ betetzen dela hasieran, eta $x = b \wedge y = a$ betetzea nahi dugu bukaeran. x-ri y esleitzen badiogu, $x = b$ edukitzea lortuko dugu, eta esleipenaren axioma erabiliz lortuko den asertzioa $\varphi_1 \equiv \psi_x^y \equiv y = b \wedge y = a$ izango da. Orain helburua $y = b \wedge y = a$ formula betearaztea da, baina hori ezinezkoa da, a eta b hasierako balioak berdinak izatea eskatzen baitu horrek. Ikusten den bezala, une honetan metodoak berak adierazten digu x-ri y balioa esleitzea ez dela egokia. Bestalde, y aldagaiak a balioa eduki dezan y-ri x esleitzeko aukera ere baztertu egin behar da, ia berdina den egoera batera baikaramatza. Arazo horrek berriro φ_1 kalkulatu baino lehen ipini beharreko esleipenera garamatza, eta, gainera, (esate baterako) x aldagaian gorde nahi dena beste balio bat dela suposatzera. Demagun beste balio hori lag izeneko aldagai laguntzaile batean dagoela. Beraz, x-ri lag esleituz aurrera egiten saiatuko gara. φ_1 berria $\psi_x^{lag} \equiv lag = b \wedge y = a$ izango da. φ betetzen dela jakinda φ_1 betearaztea da helburu berria. lag aldagaiari y eslei diezaiokegu $lag = b$ betetzea lortzeko. Esleipen horri dagokion asertzioa $\varphi_2 \equiv (\varphi_1)_{lag}^y \equiv y = b \wedge y = a$ izango da. Baina ezinezkoa den helburu bat dugu berriro ere. Beste aukerarekin saia gaitezke, hau da, x esleituko diogu y-ri. Esleipen horri dagokion asertzioa $\varphi_2 \equiv (\varphi_1)_y^x \equiv lag = b \wedge x = a$ izango da. Orain φ formulak $x = a$ inplikatzen du, baina ez $lag = b$. Horregatik, y esleituko diogu lag aldagaiari eta $\varphi_3 \equiv (\varphi_2)_{lag}^y \equiv y = b \wedge x = a$ asertzioa lortuko dugu. φ eta φ_3 baliokideak direnez, eratorpena bukatu dugu. Eratorritako programa honako hau da:

$\{ \varphi \equiv x = a \wedge y = b \}$
  $\{ \varphi_3 \}$
  `lag := y;`
  $\{ \varphi_2 \}$
  `y := x;`
  $\{ \varphi_1 \}$
  `x := lag;`
$\{ \psi \equiv x = b \wedge y = a \}$

Laburbilduz, φ aurrebaldintzaz eta ψ postbaldintzaz osatutako espezifikazio batekiko zuzena den esleipen-segida eratortzeko, φ-k ψ inplikatzen

duen ala ez egiaztatu beharko da lehenengo. Baiezko kasuan ez da esleipenik behar (hau da, agindurik gabeko programa zuzena da espezifikazio horrekiko). Bestalde, φ-k ez badu inplikatzen ψ, orduan ψ edo ψ-ren zati bat betearaziko duen esleipen bat eranstea beharrezkoa izango da. φ-k inplikatzen ez dituen ψ-ren osagaiek adieraziko digute egin beharreko esleipenak zeintzuk izan daitezkeen. Bat aukeratu beharko da. Esleipen berri bat erantsi ondoren, esleipenaren axioma erabiliz esleipen horri dagokion asertzioa kalkulatuko da. φ_1 dei diezaiokegu, adibidez, asertzio horri. Jarraian φ-k φ_1 formula berria inplikatzen duen ala ez egiaztatuko da. Helburu berria φ_1 betearaztea da. φ-k inplikatzen duen φ_j formula bat lortu arte jarraitu beharko da esleipen berriak eransten eta asertzio berriak kalkulatzen. Prozesu horretan lortzen den φ_k formularen batek eskuraezina den egoera bat adierazten badu, atzera jo eta lehenago hartutako erabakiak aldatu beharko dira. Aurreko adibidearen bidez ikus daiteke nola sortzen duen metodoak dokumentazioa eta nola laguntzen duen egokiak ez diren erabakiak detektatzen. Metodoari era zorrotzean jarraituz, esleipenak programan izango duten aurkako ordenan (behetik gora) lortuko direla ere azpimarratu nahi dugu.

Orain demagun programa hau berrerabili nahi dugula:

```
lag := y;
y := x;
x := lag;
```

Horretarako, haren sarrera-irteerako aldagaiak erabiliz parametrizatutako programa bat bezala adierazi beharko dugu: $trukaketa(x,y)$. Horrez gain, aurre-ondoetako espezifikazioa ere berrerabili ahal izateko, honako hau beteko dela baiezta dezakegu x eta y aldagai atxikitzat ez dituen edozein β formula hartuta:

$\{\ \beta^{y,x}_{x,y}\ \}$
```
  trukaketa(x,y);
```
$\{\ \beta\ \}$

Batuketa bezalako eragiketak nahasten dituzten balio-trukaketetara ere orokor daiteke ideia hori. Har dezagun honako espezifikazio hau:

$$\varphi \equiv x = a \wedge y = b$$
$$\psi \equiv x = b \wedge y = a + b$$

Espezifikazio horri dagokion programa honako hau izango litzateke:

```
lag := y;
y := x + y;
x := lag;
```

Programa hori *batura_trukaketa*(x,y) bezala parametrizatuz, x eta y aldagai atxikitzat ez dituen edozein β formularentzat honako hau beteko dela nabarmena da:

$\{\ \beta_{x,y}^{y,x+y}\ \}$
```
  batura_trukaketa(x,y);
```
$\{\ \beta\ \}$

Ohartu beharrekoa da $\psi_{x,y}^{y,x+y} \equiv y = b \wedge x + y = a + b$ dela eta formula hori φ-ren baliokidea dela. Era hauetako parametrizazioek moduluak definitzeko eta kodea berrerabili ahal izateko balio dute. Ildo horretatik joz, 7.6. atalean *batura_trukaketa* erabiliko da.

7.1.2. Adibidea: Zenbaki baten balio absolutua

Ebazteko orduan kasu-bereizketa eskatzen duen problema bat dugunean, kasu bakoitzari dagokion programa bere aldetik eratorri beharko da. Esate baterako, x eta y bi aldagairen balioen arteko aldearen balio absolutua z aldagaian kalkulatzeko, honako bi aurre-ondoetako espezifikazioei dagozkien programak eratorri beharko dira:

$$\varphi' \equiv x \geq y$$
$$\psi \equiv z =\mid x - y \mid$$

eta

$$\varphi'' \equiv x < y$$
$$\psi \equiv z =\mid x - y \mid$$

Lehenengo kasuan, φ' aurrebaldintzak ez du ψ postbaldintza inplikatzen. $x \geq y$ betetzen dela jakinda, z aldagaiari $x - y$ esleitu beharko zaio ψ bete dadin. Esleipenaren axioma erabiliz $\varphi_1 \equiv def(x-y) \wedge \psi_z^{x-y}$ formula lortuko da, hau da,

$$\varphi_1 \equiv x - y =\mid x - y \mid$$

φ' formulak φ_1 inplikatzen du, eta, horrenbestez, lehenengo kasuarekin bukatu dugu. Bigarren kasuan, φ'' aurrebaldintzak ere ez du ψ inplikatzen. $x < y$ betetzen denez, z aldagaiari $y - x$ esleitu beharko zaio ψ bete dadin. Esleipenaren axiomaren bidez lortuko den asertzio berria $\varphi_2 \equiv def(y-x) \wedge \psi_z^{y-x}$ izango da. Beraz,

$$\varphi_2 \equiv y - x =\mid x - y \mid$$

φ'' formulak φ_2 inplikatzen duenez, bigarren kasuarekin bukatu dugu, eta, gainera, kalkulatutako asertzioen bidez dokumentatuta gelditzen den honako baldintzazko aginduaren eratorpen hau ere bukatu dugu:

$\{\, \varphi \equiv True \,\}$
  if x > = y then
    $\{\, \varphi' \,\}$
    $\{\, \varphi_1 \,\}$
    z := x - y;
  else
    $\{\, \varphi'' \,\}$
    $\{\, \varphi_2 \,\}$
    z := y - x;
  end if;
$\{\, \psi \equiv z = \mid x - y \mid \,\}$

7.2. ITERAZIOEN ERATORPEN FORMALA

(φ, ψ) aurre-ondoetako espezifikazioaren bidez zehaztutako kalkulua egingo duen programa iteratibo baten eratorpen formalaren abiapuntua estrategia bat asmatzea edo pentsatzea izaten da. Estrategia horrek φ betetzen duen edozein egoeratatik abiatuta, ψ betetzen duen egoera batera iristeko balio beharko du. INB inbariante baten bidez eta E borne-adierazpen baten bidez formalizatu beharko da estrategia hori. Hoareren sistema formala erabiliz iterazio bat egiaztatzeko jarraitzen den prozesua kontuan hartzen duen era arrazoituan garatuko da programa. Hori dela-eta, espezifikazioak, inbarianteak eta borne-adierazpenak funtsezko zeregina izango dute programaren garatze-fasean. Honako eskema honetara egokitzen den programa bat lortzea da helburua:

$\{\, \varphi \,\}$
  // Hasiera
  while $\{\, \mathbf{INB} \,\}$
   B
  loop $\{\, \mathbf{E} \,\}$
    // Gorputza
  end loop;
  // Bukaera
$\{\, \psi \,\}$

Beraz, espezifikazioa, inbariantea eta E borne-adierazpena definitu ondoren, hasieraketak (`Hasiera`), begiztan jarraitzeko baldintza (`B`), prozesu errepikakorra bukatu ondoren exekutatu beharreko aginduak (`Bukaera`) eta iterazioak errepikatu beharko dituen aginduak (`Gorputza`) eratorri beharko dira. Programaren lau osagai horiek kalkulatzeko, *while*-aren erregelako lau premisak eta agindu iteratiboa bukatuko dela ziurtatzeko E borne-adierazpenak

bete behar dituen bi propietateak hartu beharko dira kontuan, betiere φ, ψ eta *INB* formulekiko. Guztira hiru urrats bereiziko ditugu:

1. Iterazioaren exekuzioa abiatu aurretik inbariantea betearazi behar duen `Hasiera` hasieraketaren kalkulua. Hasieraketa osatuko duten `Hasiera` sekuentziako aginduekin $\{\ \varphi\ \}$ `Hasiera` $\{\ INB\ \}$ hirukoteak zuzena izan beharko du.

   Hasieraketa kalkulatzeko *while*-aren erregelako lehenengo premisako inplikazioa hartuko da kontuan: $\varphi \rightarrow INB$. Inplikazio hori betetzen bada, ez da aldagairik hasieratu beharko. Inplikazio hori ez bada betetzen, aldagai bat edo gehiago hasieratu beharko dira. Gehienetan inplikazioaren lehenengo zatian (φ formulan) bigarren zatian (INB formulan) agertzen diren aldagai batzuei buruzko informaziorik ez egoteagatik ez da beteko inplikazioa. Helburua inplikazioa betetzea denez, inplikazioa betearaziko duten balioekin hasieratu beharko dira aldagai horiek. Inbarianteak eta aurretik pentsatutako estrategiak balioko dute erabakitzeko zein baliorekin hasieratu beharko diren aldagai horiek. Aldagai batzuen hasieraketa beste aldagai batzuen hasieraketaren menpekoa izatea ere ohikoa da. Teknikoki, hasieraketa berri bat erantsi ondoren, hasieraketa horri dagokion asertzioa kalkulatuko da esleipenaren axioma erabiliz, eta φ formulak formula berri hori inplikatzen duen ala ez egiaztatuko da. φ-k inplikatzen duen formula bat lortu arte jarraitu beharko da agindu berriak eransten.

2. B baldintzaren eta bukaerako aginduen kalkulua (`Bukaera`): errepikatzen jarraitzeko egokia eta zuzena den baldintza bat definitzea da B kalkulatzea. Bukaerari dagokionez, $\{\ INB \wedge \neg B\ \}$ `Bukaera` $\{\ \psi\ \}$ hirukotea zuzena izatea eragiten duten aginduak eratorri beharko dira.

   while-aren B baldintza kalkulatzeko, lehenengo $\neg B$ kalkulatuko da iterazioak noiz gelditu behar duen erabakiz. Gainera, $INB \rightarrow def(B)$ eta $(\ INB \wedge B\ ) \rightarrow E \in \mathbb{N}$ inplikazioak bete beharko dira, hau da, alde batetik, inbariantea betetzen bada, B-k definituta egon beharko du, eta, beste aldetik, inbarianteak eta B betetzeak bermatu behar dute E borne-adierazpenak balio bezala zenbaki arrunt bat (osoa eta ez-negatiboa) edukitzea. Iterazioa bukatu ondoren agindurik beharko ote den erabakitzeko, $(\ INB \wedge \neg B\ ) \rightarrow \psi$ inplikazioa hartu eta hasieraketen zatiarekin egin den bezala egin beharko da. Inplikazio hori betetzen bada, ez da behar bukaerako agindurik, baina ez bada betetzen, bukaerako aginduak eratorri beharko dira. `Bukaera` zatia osatuko duten agindu horiek inbariantea eta B-ren ukapena betetzen

dituen edozein egoeratatik abiatuz ψ betetzen duen egoeraren batera iristeko balio beharko dute.

3. Iterazioaren gorputzaren kalkulua: iterazioaren gorputzaren kalkulua $\{\, INB \wedge B \,\}$ `Gorputza` $\{\, INB \,\}$ propietateak eta $\{\, INB \wedge B \wedge E = z \,\}$ `Gorputza` $\{\, E < z \,\}$ propietateak zuzenak izan beharko dutela kontuan hartuz egingo da. Beste era batera esanda, iterazioaren gorputzaren exekuzioa inbariantea eta B baldintza betetzen dituen egoera batetik abiatzen denean, exekuzio horrek inbariantea kontserbatu egin beharko du, eta, gainera, E borne-adierazpenaren balioa txikiagotzea lortu beharko du.

   Iterazioaren gorputza, hau da, *while*-aren barruan joango diren aginduak kalkulatzeko $INB \wedge B \rightarrow INB$ inplikazioa eta $INB \wedge B \wedge E = z \rightarrow E < z$ inplikazioa hartu beharko dira abiapuntutzat, eta hasieraketak kalkulatzean bezala egin beharko da. $INB \wedge B \rightarrow INB$ inplikazioa beti beteko da, baina $INB \wedge B \wedge E = z \rightarrow E < z$ inplikazioa ez da inoiz beteko. Hori dela-eta, lehendabizi E-ren balioa txikiagotuko duen agindu bat eratorriko da. Gehienetan esleipen bat izango den agindu berri hori erantsi ondoren, esleipenaren axioma erabiliz INB formula eta $E < z$ formularekiko asertzioak kalkulatuko dira. $INB \wedge B$ formulak eta $INB \wedge B \wedge E = z$ formulak kalkulatu berri diren asertzio horiek hurrenez hurren inplikatzen badituzte, iterazioaren gorputza osatzen duten aginduen eratorpena bukatuta egongo da. Aldiz, inplikazio horiek ez badira betetzen, agindu gehiago erantsiz eta prozesua errepikatuz jarraitu beharko da.

Metodoa nola aplikatzen den erakusten duen adibide bat aurkeztuko dugu jarraian. Eratorriko den programak, osoa den x zenbaki ez-negatiboa emanda, x-ren erro karratuaren zati osoa kalkulatuko du r aldagaian. Honako bi formula hauek osatzen dute aurre-ondoetako espezifikazioa:

$$\varphi \equiv x \geq 0$$
$$\psi \equiv r \geq 0 \wedge r^2 \leq x \wedge (\, r+1 \,)^2 > x$$

Aukera bat da r aldagaia 0 balioarekin hasieratu eta $r+1$-en karratua x baino txikiagoa edo berdina den bitartean, r-ren balioa handituz joatea. Era horretara, r-ren karratua beti x baino txikiagoa edo berdina dela ziurtatu ahal izango dugu. Estrategia iteratibo hori honako inbariante honen bidez formaliza daiteke:

$$INB \equiv r \geq 0 \wedge r^2 \leq x$$

Ikus daitekeen bezala, postbaldintzatik konjuntzio bat ezabatuz lortu da inbariantea. $x - r^2$ inoiz ez denez negatiboa izango eta r-ren balioaren handitze

bakoitzarekin txikiagotuz joango denez, borne-adierazpentzat har dezakegu.

φ aurrebaldintzak ez du inbariantea inplikatzen, φ formulan ez baitago r aldagaiari buruzko informaziorik. r aldagaiak inbarianteak dioena bete dezan, nahikoa da r-ri 0 balioa esleitzearekin. Esleipenaren axioma erabiliz lortuko den asertzioa honako hau da:

$$\varphi_1 \equiv def(0) \wedge INB_r^0 \equiv 0 \geq 0 \wedge 0^2 \leq x$$

φ formulak φ_1 inplikatzen duenez (baliokideak dira), inbariantea betearazten duen `Hasiera = r := 0;` hasieraketa kalkulatuta daukagu.

Adibide honetan bigarren urratsa oso erraza da. $\neg B$ espresioa $(r+1)^2 > x$ izango da, eta, ondorioz, iterazioan jarraitzeko baldintza $(r+1)^2 \leq x$ izango da. Gainera, inbarianteak eta B-ren ukapenak ψ postbaldintza inplikatzen dutenez, ez da bukaerako agindurik behar.

Hirugarren urratsean badakigu jarraitu beharreko ideia iteratiboaren arabera zenbakiak era gorakorrean zeharkatu behar direla, eta, ondorioz, r-ren balioa handitu egin beharko da. Beraz, `r:= r+1;` esleipena erantsiko dugu eta esleipen horri INB eta $E < z$ formulekiko dagozkion φ_2 eta φ_3 asertzioak kalkulatuko ditugu:

$$\varphi_2 \equiv def(r+1) \wedge INB_r^{r+1} \equiv r+1 \geq 0 \wedge (\, r+1\,)^2 \leq x$$
$$\varphi_3 \equiv def(r+1) \wedge (\, E < z\,)_r^{r+1} \equiv x - (\, r+1\,) < z$$

$INB \wedge B \rightarrow \varphi_2$ inplikazioa eta $INB \wedge B \wedge E = z \rightarrow \varphi_3$ inplikazioa bete egiten direnez, iterazioaren gorputza lortu dugu. Eratorritako programa, bere dokumentazio guztiarekin, honako hau da:

```
{ φ ≡ x ≥ 0 }
  { φ1 }
  r := 0;
  while  { INB ≡ r ≥ 0 ∧ r² ≤ x }
   (r+1)*(r+1) <= x
  loop   { E ≡ x − r² }
    { φ2 }  { φ3 }
    r := r+1;
  end loop;
{ ψ ≡ r ≥ 0 ∧ r² ≤ x ∧ (r+1)² > x }
```

7.3. ARIKETAK: ZENBAKIEI BURUZKO ITERAZIOEN ERATORPENA

1. Formalki eratorri n zenbaki arruntaren faktoriala f aldagaian kalkulatuko duen programa. n-ren faktoriala $n!$ bezala adieraziko dugu.

   Hainbat estrategia erabil daitezkeenez, eratorri hiru programa honako hiru inbariante hauetako bakoitzaren bidez formalizatutako estrategiari jarraituz:

   1.1. $INB \equiv 0 \leq i \leq n \wedge f = i!$

   1.2. $INB \equiv 0 \leq i \leq n \wedge f = (n-i)!$

   1.3. $INB \equiv 0 \leq i \leq n-1 \wedge f = n!/(n-i-1)!$

2. Formalki eratorri $\{n \geq 0\}$ aurrebaldintza eta $\{\ y = n^2\ \}$ postbaldintzarekiko zuzena den eta $\{\ y + x^2 = n^2 \wedge 0 \leq x \leq n\ \}$ inbariantea kontserbatzen duen iterazioa. Gogoratu $(\ a \pm 1\ )^2 = a^2 \pm 2a + 1$ betetzen dela.

3. Formalki eratorri honako espezifikazioarekiko zuzena den

$$\varphi \equiv n \geq 1$$
$$\psi \equiv e = n! * n^n$$

   eta honako inbariante hau kontserbatzen duen

$$INB \equiv 0 \leq i \leq n \wedge e = i! * n^i$$

   iterazioa.

7.4. INBARIANTEAK SORTZEKO JARRAIBIDEAK

Aurre-ondoetako espezifikaziotik eta aukeratutako estrategia iteratibotik inbariantea sortzeko metodo orokorrik ez egon arren, inbariantea sortzeko lagungarriak diren jarraibide batzuk badaude. Erabilgarriak diren jarraibide batzuk azalduko ditugu atal honetan, eta, gainera, adibideak emanez jarraibide horien aplikazioa erakutsiko dugu.

Diseinatuko den iterazioaren urrats bakoitzaren aurretik gerta daitezkeen (edo onargarriak diren) konputazio-egoera guztiak era zehatzean deskribatu behar ditu inbarianteak. Postbaldintza aztertzea eta abiapuntu gisa erabiltzea lagungarria gertatzen da askotan inbariantea sortzean. Adibidez, demagun zenbaki osozko $A(1..n)$ bektore ez-hutsaren osagaien batura b aldagaian

kalkulatuko duen programa bat eratorri nahi dela. Honako formula hauen bidez adieraziko litzateke espezifikazioa:

$$\varphi \equiv n \geq 1$$
$$\psi \equiv b = \sum_{k=1}^{n} A(k)$$

Jarraian deskribatzen diren lau aukeren bidez erakusten den bezala, kalkulu hori burutzeko hainbat estrategia proposa daitezke.

1. Indize bezala i aldagaia erabiliz, ezkerretik eskuinera zeharkatu $A(1..n)$ bektorea eta iterazio berri bat hastera doanean, batuko den osagaia zein den adieraz dezala i aldagaiak.

   $$INB_1 \equiv (\, 1 \leq i \leq n+1 \,) \wedge b = \sum_{k=1}^{i-1} A(k)$$

   Xehetasunak zehaztean mutur biak finkatuko dira: elementurik batu gabe abiatuko da prozesua hasieran eta i aldagaiak $n+1$ balioa hartzen duenean n osagaien batura edukiko da.

2. Indize bezala i aldagaia erabiliz, ezkerretik eskuinera zeharkatu $A(1..n)$ bektorea eta batutako azkeneko osagaia zein den adieraz dezala i aldagaiak b aldagaiari elementu bat batzeko unea iristen denean.

   $$INB_2 \equiv (\, 0 \leq i \leq n \,) \wedge b = \sum_{k=1}^{i} A(k)$$

   Elementurik batu gabe abiatuko da prozesua hasieran eta osagai guztiak batuta egongo dira i aldagaiak n balioa hartzen duenean.

3. Bigarren aukeran aurkeztutako estrategia bera baina b aldagaian lehenengo osagaia duela abiatuko da iterazioa.

   $$INB_3 \equiv (\, 1 \leq i \leq n \,) \wedge b = \sum_{k=1}^{i} A(k)$$

4. Eskuinetik ezkerrera zeharkatu $A(1..n)$ bektorea eta iterazio bakoitzaren hasieran (hau da, b aldagaiari osagai bat batzeko unea iristen zaionean) batu beharreko osagaia zein den adieraz dezala i aldagaiak.

   $$INB_4 \equiv (\, 0 \leq i \leq n \,) \wedge b = \sum_{k=i+1}^{n} A(k)$$

   0 balioa hartzen duenean i aldagaiak, osagai denak batuta egongo dira.

Aukera bakoitzean aurkeztutako estrategia formalizatu da lau inbariante horien bidez. Lehenago ere esan dugun bezala, aukeratutako estrategia iteratiboa aztertzea da onena inbariantea lortzeko. Postbaldintzan oinarrituz ateratzen denez jarraitu beharreko estrategia, postbaldintza aztertzea oso lagungarria izan ohi da. Adibide horretan, postbaldintzatik atera daiteke aurkeztutako lau aukeretako bakoitzari dagokion inbariantea.

INB_1 lortu da ψ formulako n konstantea $i-1$ espresioarekin ordezkatuz eta i aldagaiak zeharkatuko duen tartea ipiniz. Tarte hori zeharkatzean i aldagaia 1etik hasiko da; izan ere, batu beharreko osagaia zein den zehaztuko du i aldagaiak eta b aldagaian bektoreko elementurik gorde gabe abiatzea erabaki da. Beraz, i-ren balioa 1 denean elementurik ez da batu oraindik, eta i aldagaiak $n+1$ balioa hartzen duenean bektoreko osagai denak batuta egongo dira. Inbariantea lortu eta gero, helburua den puntua eta oraingo puntuaren arteko aldea edo distantzia bezala plantea daiteke borne-adierazpena, hau da, $E \equiv n+1-i$. Bektorean eskuinera egin ahala E-ren balioa gero eta txikiagoa izango dela eta, gainera, haren balioa beti $\mathbb{N}$ multzokoa izango dela nabarmena da. *while* agindua bukatzeko zenbat iterazio gelditzen diren era zehatzean adierazten du E-k kasu honetan.

INB_2 lortu da ψ formulako n konstantea i aldagaiarekin ordezkatuz eta i aldagaiak zeharkatuko duen tartea zehaztuz. Hor 0 izango da i-ren hasierako balioa, eta n izango da i-k hartuko duen azken balioa; izan ere, batutako azken osagaia zein den adieraziko du i aldagaiak, eta b aldagaian bektoreko elementurik gorde gabe abiatzea erabaki da. Aurreko kasuko planteamendu berari jarraituz lortuko da borne-adierazpena. Horrela, $E \equiv n-i$ izango litzateke kasu honetan.

INB_3 INB_2-ren oso antzekoa da. Hemen i aldagaia 1 baliotik abiatuko da, b aldagaian bektoreko lehenengo osagaia gordeta edukiko baita hasieratik, eta hori da aurreko planteamenduarekin alderatuta aldatzen den gauza bakarra. Borne-adierazpena ere ez da aldatuko: $E \equiv n-i$.

Alderantzizko zentzuan zeharkatuko da bektorea laugarren estrategiari jarraitutakoan eta hori da, hain zuzen, aurreko estrategiekiko funtsezko aldaketa. Beraz, ψ formulako 1 konstantea $i+1$ espresioarekin ordezkatuz eta i aldagaia hartuz joango den balioez osatutako tartea erantsiz lortu da INB_4. Hor n-tik hasiko da i aldagaia; izan ere, batuko den hurrengo osagaia zein den adieraziko du i aldagaiak, eta b aldagaian bektoreko elementurik gorde gabe abiatzea erabaki da. Beraz, osagairik ez da batu oraindik i-ren balioa n denean, eta b aldagaian osagai guztien batura edukiko da i aldagaia 0 balioa hartzera iristen denean. Inbariantea lortu ondoren, helburua den puntua eta oraingo puntuaren arteko aldea edo distantzia bezala plantea daiteke borne-adierazpena, hau da, 0 eta i-ren artekoa. Balio horrek positiboa izan behar

duenez, $E \equiv i-0$ izango da, hau da, $E \equiv i$. Bektorean ezkerrera egin ahala E-ren balioa gero eta txikiagoa izango da, eta, gainera, haren balioa beti $\mathbb{N}$ multzokoa izango da. Agindu iteratiboa bukatzeko zenbat iterazio gelditzen diren era zehatzean adieraziko du E-k, aurreko hiru kasuetan bezalaxe.

Laburbilduz, aldagai berri bat (i aldagaia) duen espresio bat postbaldintzako konstante baten ordez (1 edo n) ipiniz lortu dira lau inbariante horiek. Inbarianteak lortzeko jarraibideetako bat da aldaketa hori egitea. Atal honetako gainerakoan beste jarraibide batzuk ere aurkeztuko ditugu.

Oro har, problemaren espezifikazioan eta estrategia iteratiboan oinarrituz inbariantea formulatzeko jarraibide batzuk eman daitezke. Ikusi dugun bezala, ψ postbaldintzako espresio bat (behar bada konstante bat) orokorragoa den (eta gehienetan ψ formulan agertzen ez den aldagai bat izan ohi duen) beste espresio batekin ordezkatzea da jarraibide horietako bat. ψ postbaldintzako konjuntzio bat kentzea izan daiteke beste jarraibide bat. Bi teknika horiek batera ere erabil daitezke, hau da, konjuntzio bat kendu eta, gainera, espresio bat orokorragoa den beste espresio batekin ordezkatu. Aurrebaldintzaren eta postbaldintzaren nahasketa bat ahultzean datza beste jarraibide edo aukera bat. Adibideak emanez azalduko ditugu lau jarraibide horiek orain.

7.4.1. Espresio bat orokorragoa den beste batekin ordezkatzea

Azpiatal honetan aurkeztuko diren bi adibideetan postbaldintzako konstante bat aldagai berri batekin ordezkatuko da, eta, gainera, aldagai horrek har ditzakeen balioen tartea ipiniko da.

Zenbaki osozko $A(1..n)$ bektore ez-hutseko balio handiena h aldagaian kalkulatuko duen programaren espezifikazioa hartuko dugu lehendabizi:

$$\begin{aligned} \varphi &\equiv n \geq 1 \\ \psi &\equiv h = \mathit{handiena}(A(1..n)) \end{aligned}$$

Hor $A(1..n)$ bektoreko balio handiena adierazten da $\mathit{handiena}(A(1..n))$-ren bidez. Aztertutako azken osagaia zein izan den adieraziko duen i aldagaia indize bezala erabiliz, ezkerretik eskuinera bektorea zeharkatzea da bektoreko balio handiena kalkulatzeko jarrai daitekeen estrategia bat. Bektorea hutsa ez denez, behin-behineko balio handiena bezala lehenengo posizioko osagaia hartuko da hasieran. ψ-ko n konstantea i aldagaiarekin ordezkatuz eta i-ren definizio-eremua finkatuz lortuko da inbariantea:

$$INB \equiv 1 \leq i \leq n \wedge h = \mathit{handiena}(A(1..i))$$

Bigarren adibide bezala, zenbaki osozko $A(1..n)$ bektore ez-hutsean dauden balio negatiboen kopurua kalkulatu beharko lukeen programaren espezifikazioa hartuko dugu:

$$\begin{array}{l} \varphi \equiv n \geq 1 \\ \psi \equiv neg = \mathcal{N}k\,(\,1 \leq k \leq n \wedge A(k) < 0\,) \end{array}$$

Aztertutako azken osagaia zein izan den adieraziko duen i aldagaia indize bezala erabiliz, ezkerretik eskuinera bektorea zeharkatzea da bektorean dauden balio negatiboen kopurua kalkulatzeko aukera daitekeen estrategia bat. Agindu iteratiboa hasi aurretik ez da bektoreko posiziorik aztertuko. ψ-ko n konstantea i aldagaiarekin ordezkatuz eta i-ren definizio-eremua zehaztuz lortuko da inbariantea:

$$INB \equiv 0 \leq i \leq n \wedge neg = \mathcal{N}k\,(\,1 \leq k \leq i \wedge A(k) < 0\,)$$

7.4.2. Konjuntzio bat ezabatzea

Postbaldintzako konjuntzio bat ezabatuz inbariantea lortzen lagun dezakeen jarraibidea bi adibide erabiliz azalduko dugu orain.

x balio positibora azpitik gehien hurbiltzen den 2ren berretura y aldagaian kalkulatu beharko lukeen programaren espezifikazioa honela adieraz daiteke:

$$\begin{array}{l} \varphi \equiv x \geq 1 \\ \psi \equiv ber2(y) \wedge y \leq x \wedge 2 * y > x \end{array}$$

Hor $\exists k\,(\,k \geq 0 \wedge y = 2^k\,)$ formulak definitzen du $ber2(y)$ predikatuaren esanahia. x-ren azpitik gehien hurbiltzen den 2ren berretura kalkulatzeko jarrai daitekeen estrategia bat hau da: 2^0tik hasi eta 2ren berreturak kalkulatuz joatea, 2ren hurrengo berreturak x gaindituko duela detektatutakoan geldituz. 2^0 den 1 baliotik hasi eta 2 balioaz behin eta berriz biderkatuz joatea nahikoa da 2ren berreturak kalkulatuz joateko. 2ren berretura bat 2 zenbakiaz biderkatzen den bakoitzean, 2ren hurrengo berretura lortuko da. ψ formulatik $2 * y > x$ konjuntzioa kenduz lor daiteke ideia hau formalizatuko duen inbariantea:

$$INB \equiv ber2(y) \wedge y \leq x$$

Konjuntzioa ezabatuz inbariantea lortzeko jarraibidearen bigarren adibide bat erakusteko, osoak eta positiboak diren x eta y bi zenbaki emanda, eta, gainera, y-ren balioa x-rena baino txikiagoa edo berdina izango dela jakinda, x eta y-ren zatitzaile komunetako handiena h aldagaian kalkulatu beharko

lukeen programaren espezifikazioa hartuko dugu:

$$\begin{array}{lcl} \varphi & \equiv & x \geq y \geq 1 \\ \psi & \equiv & 1 \leq h \leq y \ \wedge \ x \,\%\, h = 0 \ \wedge \ y \,\%\, h = 0 \ \wedge \\ & & \quad \forall k \, (\, h+1 \leq k \leq y \ \rightarrow \ (\, x \,\%\, k \neq 0 \ \vee \ y \,\%\, k \neq 0 \,) \,) \end{array}$$

$x \geq y$ betetzen dela jakinda, x eta y-ren zatitzaile komunetako handiena h aldagaian kalkulatzeko, h aldagaia y balioarekin hasieratu, h-ren balioa x eta y-ren zatitzailea den ala ez aztertu, eta, horrela ez bada, h-ren balioa txikiagotuz joatea da aukera bat. Aipatutako baldintzak betetzen dituzten x eta y bi zenbaki hartuta, badakigu gutxienez 1 balioak zenbaki biak zatituko dituela, ondorioz, zatitzaile komunetako handiena den balioa beti egongo dela eta haren bilaketa beti bukatuko dela, estrategia horri jarraitzen bazaio. $x \% h = 0 \ \wedge \ y \,\% h = 0$ konjuntzioa ψ-tik kenduz lor daiteke bilaketa hori burutzeko aipatutako ideia formalizatuko duen inbariantea:

$$INB \ \equiv \ 1 \leq h \leq y \ \wedge \ \forall k \, (\, h+1 \leq k \leq y \ \rightarrow \ (\, x \,\%\, k \neq 0 \ \vee \ y \,\%\, k \neq 0 \,) \,)$$

7.4.3. Konjuntzio bat kendu eta espresio bat orokorragoa den beste espresio batekin ordezkatzea

Batzuetan, aurreko bi azpiataletan aurkeztutako bi jarraibideak aplikatu beharko dira ψ postbaldintzatik abiatuz inbariantea lortzeko. Beraz, ψ-ko konjuntzio bat ezabatu beharko da, eta, gainera, ψ-ko espresio bat aldagai berri bat izango duen espresio orokorrago batekin ordezkatu beharko da, aldagai berriaren definizio-eremua ere erantsiz. Hiru adibide emango ditugu bi jarraibide hauek batera nola erabiltzen diren erakusteko.

$A(1..n)$ bektore ez-hutsean x balioaren lehenengo agerpenari dagokion indizea i aldagaian kalkulatuko duen programaren aurre-ondoetako espezifikazioa honela formaliza daiteke:

$$\begin{array}{lcl} \varphi & \equiv & n \geq 1 \\ \psi & \equiv & 1 \leq i \leq n+1 \ \wedge \ x \notin A(1..i-1) \ \wedge \ (\, i = n+1 \ \vee \ x = A(i) \,) \end{array}$$

$A(1..n)$ bektorean x ez agertzeko aukera ere onartzen da espezifikazio horretan. Kasu hori gertatzen bada, i aldagaiak $n+1$ balioarekin bukatu beharko du. x balioa bektorean agertzen bada, haren lehenengo agerpenari dagokion posizioa detektatzeko aukera bat honako hau da: indize bezala j aldagai laguntzailea erabiliz, ezkerretik eskuinera bektorea zeharkatu, iterazio bakoitzaren hasieran j aldagaiak iterazio horretan aztertuko den posizioaren hurrengoa adierazten duela, aztertuko den posizioa $j-1$ posizioa izanik. Alde batetik, x balioa bektorean agertzen bada, iterazioa gelditu egin beharko

da. Bestetik, x balioa bektorean ez bada agertzen, bektorearen eremutik kanpoko posizioak ez direla aztertuko, hau da, j aldagaiak $n+1$ balioa ez duela gainditukо ziurtatu beharko da. Agindu iteratiboa hasi aurretik ez da bektoreko posiziorik aztertuko. Agindu iteratiboa bukatu ondoren, x ez bada agertu aztertutako $1..n-1$ tartean, x balioa n posizioan agertzen den ala ez jakitea faltako da, eta, horregatik, agindu iteratibotik kanpo baldintzazko agindu bat ipini beharko da. Ideia hori formalizatuko duen inbariantea postbaldintzatik $1 \leq i \leq n+1 \wedge (i = n+1 \vee x = A(i))$ konjuntzioa ezabatuz eta $x \notin A(1..i-1)$ formulako i aldagaia $j-1$ espresioarekin ordezkatuz lor daiteke. Gainera, j-ren definizio-eremua eman beharko da:

$$INB \equiv 2 \leq j \leq n+1 \wedge x \notin A(1..j-2)$$

Inbariantea lortzeko jarraibide mota honetako bigarren adibide bezala, espezifikazio bera hartu eta x-ren lehenengo agerpenaren posizioa zein den arakatzeko bigarren era bat planteatuko dugu orain. Aztertua izango den hurrengo posizioa zein den adieraziko duen i aldagaia indize bezala erabiliz, bektorea ezkerretik eskuinera zeharkatuko da. Alde batetik, x agertzen bada, iterazioa gelditu egin beharko da. Beste aldetik, x bektorean ez bada agertzen, $n+1$ baino handiagoa den baliorik ezingo du hartu i-k. c aldagai boolearra erabil daiteke x agertzen denean iterazioa berehala bukatzeko. x aurkitzen ez den bitartean aldagai boolear horren balioa *faltsua* izango da, eta x-ren agerpen bat aurkitzen bada, *egiazkoa* balioa hartuko du.

Agindu iteratiboarekin hasi aurretik ez da bektoreko posiziorik aztertuko. Lehendabizi, $i = n+1 \vee x = A(i)$ konjuntzioa ken dezakegu kalkuluak egiteko aukera hau formalizatuko duen inbariantea lortzeko. Honako formula hau izango dugu ezabaketa hori egin ondoren:

$$1 \leq i \leq n+1 \wedge x \notin A(1..i-1)$$

Formula hori beste formula honen baliokidea da:

$$1 \leq i \leq n+1 \wedge (True \leftrightarrow x \notin A(1..i-1))$$

$True$ formula c aldagaiarekin ordezkatuko dugu formula horretan:

$$INB \equiv 1 \leq i \leq n+1 \wedge (c \leftrightarrow x \notin A(1..i-1))$$

Beraz, konjuntzio bat ezabatu da eta aldagai berri bat sartu da.

Hirugarren adibidea aurkeztuko dugu orain. Zenbaki osozko $A(1..n)$ bektore ez-hutseko balio handiena bikoitia den erabaki, eta erantzuna *hand_bik* aldagai boolearrean utzi behar duen programaren aurre-ondoetako espezifikazioa osatzen dute φ eta ψ honako bi formula hauek:

$$\begin{array}{lcl} \varphi & \equiv & n \geq 1 \\ \psi & \equiv & hand\_bik \leftrightarrow handiena(A(1..n)) \,\%\, 2 = 0 \end{array}$$

Demagun bektoreko baliorik handiena h aldagaian gordeko dela. Honela berridatz daiteke postbaldintza:

$$\begin{array}{lcl} \varphi & \equiv & n \geq 1 \\ \psi & \equiv & (hand\_bik \leftrightarrow h \,\%\, 2 = 0) \wedge h = handiena(A(1..n)) \end{array}$$

Indize bezala i aldagaia erabiliz bektorea ezkerretik eskuinera zeharkatuz joanda, zeharkatutako bektore-zatiko balio handiena h aldagaian edukitzeko, h eguneratuz joatea da bektoreko baliorik handiena bikoitia den erabakitzeko aukera bat. Aztertutako azken posizioa zein izan den adieraziko du i aldagaiak iterazio bakoitzaren hasieran. Behin-behineko baliorik handiena bezala $A(1..n)$ bektoreko lehenengo osagaia hartuko da agindu iteratiboarekin hasi aurretik. Bektore osoa zeharkatu ondoren, eta, horrenbestez, h aldagaian bektoreko baliorik handiena kalkulatu ondoren, *egiazkoa* balioa gordeko da *hand_bik* aldagaian h aldagaiak balio bikoitia baldin badu, eta, bestela, *faltsua* balioa gordeko da. Kalkulua egiteko ideia hori formalizatuko lukeen inbariantea ($hand\_bik \leftrightarrow h \,\%\, 2 = 0$) konjuntzioa ψ-tik ezabatuz eta n konstantea i aldagaiarekin ordezkatuz lor daiteke:

$$INB \equiv 1 \leq i \leq n \wedge h = handiena(A(1..i))$$

7.4.4. Aurrebaldintzaren eta postbaldintzaren nahasketa bat ahultzea

Aurrebaldintza eta postbaldintza ahulduz lortutako formulak elkartzean oinarritzen da inbarianteak lortzeko aurkeztuko dugun laugarren jarraibidea. Konjuntzioak ezabatuz eta espresio batzuk orokorragoak diren beste espresio batzuekin ordezkatuz lortzen da formulak ahultzea. Bi adibide emango ditugu jarraibide hau erakusteko.

x balioaren $A(1..n)$ bektore ez-hutseko lehenengo agerpenaren posizioa i aldagaian utzi behar duen programaren espezifikazioa hartuko dugu hasteko. Posizioren batean x balioa agertzen dela badakigu kasu honetan:

$$\begin{array}{lcl} \varphi & \equiv & n \geq 1 \wedge x \in A(1..n) \\ \psi & \equiv & 1 \leq i \leq n \wedge x \notin A(1..i-1) \wedge x = A(i) \end{array}$$

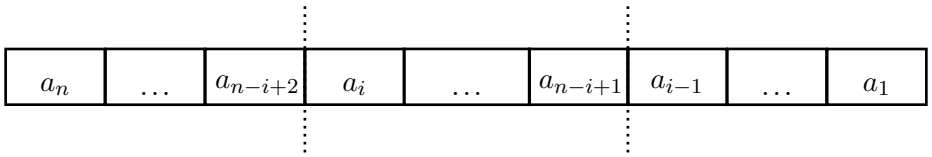
x balioa bektorean agertzen dela jakinda, aztertua izango den hurrengo posizioa zein den adierazten duen i aldagaia indize bezala erabiliz, bektorea ezkerretik eskuinera zeharkatzea da x-ren lehenengo agerpenaren posizioa zein den kalkulatzeko estrategia bat. Bilaketa x agertu bezain laster bukatuko da. Agindu iteratiboa hasi aurretik ez da bektoreko posiziorik aztertuko. φ eta ψ-ren nahasketa bat ahulduz lor daiteke kalkulua egiteko ideia hori formalizatuko duen inbariantea. Gehiago zehaztuz, $x \in A(1..n)$ espresioko 1 konstantea i aldagaiarekin ordezkatuko da φ formulan eta ψ-ko $x = A(i)$ zatia ezabatu egingo da.

$$INB \equiv \underbrace{1 \leq i \leq n \wedge x \notin A(1..i-1)}_{\psi \text{ ahulduz}} \wedge \underbrace{x \in A(i..n)}_{\varphi \text{ ahulduz}}$$

Bigarren adibide bezala, $A(1..n)$ bektorearen alderantzizkoa kalkulatu beharko duen programaren espezifikazioa hartuko dugu. Jarraian erakusten diren φ aurrebaldintzaren eta ψ postbaldintzaren bidez formaliza daiteke espezifikazio hori.

$$\begin{array}{rcl} \varphi & \equiv & A(1..n) = (\, a_1,\, a_2,\, \ldots,\, a_{n-1},\, a_n \,) \\ & \equiv & \forall j \,(\, 1 \leq j \leq n \,\rightarrow\, A(j) = a_j \,) \\ \psi & \equiv & A(1..n) = (\, a_n,\, a_{n-1},\, \ldots,\, a_2,\, a_1 \,) \\ & \equiv & \forall j \,(\, 1 \leq j \leq n \,\rightarrow\, A(j) = a_{n+1-j} \,) \end{array}$$

Indize bezala i aldagaia erabiliz, bektorea erdiraino ezkerretik eskuinera zeharkatzea eta i posizioko elementua bere posizio simetrikokoarekin ($n+1-i$ posiziokoarekin) trukatzea da bektorearen alderantzizkoa kalkulatzeko era bat. i aldagaiaren balioak bitartekoa den posizio bat adierazten duenean, egoera honela deskriba daiteke grafikoki:



Jarraian erakusten den inbarianteak egoera hori formalizatzen du:

$$\begin{array}{rcl} INB & \equiv & 1 \leq i \leq \frac{n}{2}+1 \wedge \\ & & \underbrace{A(1..i-1) = (\, a_n,\, a_{n-1},\, \ldots,\, a_{n-i+2} \,)}_{\text{postbaldintza ahulduz}} \wedge \\ & & \underbrace{A(i..n-i+1) = (\, a_i,\, a_{i+1},\, \ldots,\, a_{n-i+1} \,)}_{\text{aurrebaldintza ahulduz}} \wedge \\ & & \underbrace{A(n-i+2..n) = (\, a_{i-1},\, a_{i-2},\, \ldots,\, a_1 \,)}_{\text{postbaldintza ahulduz}} \end{array}$$

$$\begin{array}{rcl} & \equiv & 1 \leq i \leq \frac{n}{2}+1 \wedge \\ & & \quad \forall j\ (\ 1 \leq j \leq i-1\ \rightarrow\ A(j) = a_{n+1-j}\ ) \wedge \\ & & \quad \forall j\ (\ i \leq j \leq n-i+1\ \rightarrow\ A(j) = a_j\ ) \wedge \\ & & \quad \forall j\ (\ n-i+2 \leq j \leq n\ \rightarrow\ A(j) = a_{n+1-j}\ ) \end{array}$$

Beraz, n konstantea $i-1$ espresioarekin ordezkatu da postbaldintza lehenengo aldian ahultzean, eta 1 konstantea $n-i+2$ espresioarekin ordezkatu da postbaldintza bigarren aldian ahultzean. Bestalde, n konstantea $n-i+1$ espresioarekin eta 1 konstantea i aldagaiarekin ordezkatu dira aurrebaldintza ahultzeko. Gainera, i aldagaia berria denez, haren definizio-eremua gehitu da.

7.5. BEKTOREEKIN KALKULUAK EGITEN DITUZTEN ITERAZIOEN ERATORPENA

Bektoreak (beraien balioa aldatu gabe) zeharkatu eta aztertzen dituzten iterazioen eratorpenaren hiru adibide aurkeztuko dira atal honetan. Aurreko adibideetan baino xehetasun gutxiagorekin azalduko dira kalkuluak. Nabariak iruditzen ez zaizkion baieztapenei buruzko xehetasunak bere kabuz egiazta ditzake jakin-mina duen irakurleak.

7.5.1. Bektore bateko osagaien batura

Honela formula daiteke zenbaki osozko $A(1..n)$ bektore ez-hutsa emanda, haren osagaien batura b aldagaian kalkulatu beharko duen programaren aurreondoetako espezifikazioa:

$$\begin{array}{rcl} \varphi & \equiv & n \geq 1 \\ \psi & \equiv & b = \sum_{k=1}^{n} A(k) \end{array}$$

Egin beharreko kalkulua aurrera eramateko balio duten zenbait estrategia plantea daitezke espezifikazio horretatik abiatuta. Esate baterako, i aldagaia indize bezala erabiliz, bektorea ezkerretik eskuinera zeharkatzea da arruntena edo ohikoena. Oraindik batu gabe dagoen lehenengo osagaia zein den adieraziko du i aldagaiak. Gainera, agindu iteratiboa exekutatzen hasi aurretik ez da elementurik batuko. Beraz, 1 izango da i-ren hasierako balioa, eta $n+1$ izango da i aldagaiak hartuko duen azken balioa. Kontuan hartu i-ren balioa n izateak batu beharreko hurrengo osagaia n posiziokoa dela esan nahiko duela. Bestalde, 1 posiziotik n posiziora arteko osagai denak batuta edukiko dira i-ren balioa $n+1$ denean. Honako inbariante honek estrategia

hori formalizatzeko balio du:

$$INB \equiv 1 \leq i \leq n+1 \wedge b = \sum_{k=1}^{i-1} A(k)$$

Inbariantea lortu ondoren, i-k hartuko duen azken balioaren eta i-ren arteko aldea bezala defini dezakegu E borne-adierazpena, hau da, $E \equiv n+1-i$.

Hasteko, φ aurrebaldintzak ez du ezartzen inbariantea. Zehazkiago esanda, i eta b aldagaiei buruzko informaziorik ez du φ-k. Beraz, inbariantea *while* agindua exekutatzen hasi aurretik bete dadin, beharrezkoa da bi aldagai horiek hasieratzea. Aukeratutako estrategiaren arabera bektorea ezkerretik eskuinera zeharkatuko denez, i aldagaiari 1 balioa esleitzea nahikoa da $1 \leq i \leq n+1$ betetzeko, hau da inbariantea kontuan hartuz har dezakeen balio txikiena. Esleipen horri dagokion asertzioa $\varphi_1 \equiv INB_i^1 \equiv 1 \leq 1 \leq n+1 \wedge b = 0 \equiv 0 \leq n \wedge b = 0$ da. Orain b aldagaiak 0 balioa hartzea gelditzen da. φ aurrebaldintzak $\varphi_2 \equiv (\varphi_1)_b^0 \equiv 0 \leq n$ asertzioa inplikatzen duenez, zuzena den hasieraketa daukagu.

Bektorea ezkerretik eskuinera zeharkatzen ari garela eta inbariantearen arabera i-ren goiko muga $n+1$ dela jakinda, agindu iteratiboa bukatu egin beharko da i-ren balioa $n+1$ izatera iristen denean. Beraz, $\neg B \equiv i = n+1$, eta, ondorioz, $B \equiv i \neq n+1$. B baldintzak $INB \rightarrow def(B)$ eta $INB \wedge B \rightarrow E \in \mathbb{N}$ inplikazioak bete egiten ditu, hau da, B ondo definituta egongo da inbariantea betetzen den guztietan, eta E-k zenbaki arrunta den balioa hartuko du inbariantea betetzen bada eta B-ren balioa egiazkoa baldin bada. Bestalde, bukaerako agindurik ez da behar, $INB \wedge \neg B \rightarrow \psi$ inplikazioa ere betetzen baita.

Iterazioaren gorputza eratortzeko, hasteko badakigu i aldagaiaren balioari 1 gehitu beharko zaiola, eta, horrela, E-ren balioa txikiagotu egingo da iterazio bakoitzean. Esleipen hori eta INB eta $E < z$ asertzioak hartuz, $\varphi_3 \equiv (INB)_i^{i+1} \equiv 0 \leq i \leq n \wedge b = \sum_{k=1}^{i} A(k)$ eta $\varphi_4 \equiv (E < z)_i^{i+1} \equiv n-i < z$ asertzioak lortuko ditugu esleipenaren axioma erabiliz. $INB \wedge B \wedge E = z$ asertzioak φ_4 asertzioa inplikatzen du, baina $INB \wedge B$ asertzioak ez du φ_3 inplikatzen. Inbariantean $b = A(1) + A(2) + \ldots + A(i-1)$ dela esaten zaigu, eta φ_3-n, aldiz, $b = A(1) + A(2) + \ldots + A(i-1) + A(i)$ daukagu. Horregatik, $INB \wedge B \rightarrow \varphi_3$ inplikazioa ez da beteko. φ_3 betetzea denez helburua, b aldagaiaren oraingo balioari $A(i)$ batu behar zaiola ondoriozta dezakegu. Esleipen berri hau eta φ_3 hartuz $\varphi_5 \equiv def(A(i)) \wedge (\varphi_3)_b^{b+A(i)} \equiv (1 \leq i \leq n) \wedge b + A(i) = \sum_{k=1}^{i} A(k)$ kalkulatuko dugu. Orain bai $INB \wedge B \rightarrow \varphi_5$ bai $INB \wedge B \wedge E = z \rightarrow \varphi_6$ inplikazioak bete egiten direnez, $\varphi_6 \equiv def(A(i)) \wedge (\varphi_4)_b^{b+A(i)} \equiv 1 \leq i \leq n \wedge n-i < z$ izanda, iterazioaren gorputza lortuta dugu. Eratorritako programa honako hau da:

$\{\, \varphi \equiv n \geq 1 \,\}$
  $\{\, \varphi_2 \,\}$
  `b := 0;`
  $\{\, \varphi_1 \,\}$
  `i := 1`
  `while` $\{\, INB \equiv 1 \leq i \leq n+1 \wedge b = \sum_{k=1}^{i-1} A(k) \,\}$
   `i /= n + 1`
  `loop` $\{\, E \equiv n+1-i \,\}$
    $\{\, \varphi_5 \,\}$ $\{\, \varphi_6 \,\}$
    `b := b + A(i);`
    $\{\, \varphi_3 \,\}$ $\{\, \varphi_4 \,\}$
    `i := i + 1;`
  `end loop;`
$\{\, \psi \equiv b = \sum_{k=1}^{n} A(k) \,\}$

Beste aukera bat, agindu iteratiboa hasi aurretik osagairik ez batzea eta batu beharreko hurrengo osagaia zein den adierazten duen i aldagaia indize bezala erabiliz bektorea eskuinetik ezkerrera zeharkatzea da. Estrategia hori honako inbariantearen bidez formaliza daiteke:

$$INB \equiv 0 \leq i \leq n \wedge b = \sum_{k=i+1}^{n} A(k)$$

Inbariante hori erabiliz eta beste estrategiari dagokion programa eratortzean zehaztutako arrazoibide berari jarraituz, borne-adierazpena $E \equiv i$ izango litzateke. n balioarekin hasieratuko genuke i aldagaia eta 0rekin b aldagaia. Iterazioaren gorputzean b-ri $b + A(i)$ esleitu beharko genioke, eta i-ren balioa txikiagotu egin beharko litzateke. Beraz, honako programa hau lortuko genuke:

$\{\, \varphi \equiv n \geq 1 \,\}$
  `b := 0;`
  `i := n;`
  `while` $\{\, INB \equiv 0 \leq i \leq n \wedge b = \sum_{k=i+1}^{n} A(k) \,\}$
   `i /= 0`
  `loop` $\{\, E \equiv i \,\}$
    `b := b + A(i);`
    `i := i - 1;`
  `end loop;`
$\{\, \psi \equiv b = \sum_{k=1}^{n} A(k) \,\}$

7.5.2. Bektoreen arteko berdintasuna

$A(1..n)$ eta $B(1..n)$ bektore ez-hutsak emanda, berdinak diren erabaki eta erantzuna c aldagaian utziko duen programa eratorriko dugu jarraian.

Lortu behar den programak egin beharreko kalkulua deskribatzen du honako aurre-ondoetako espezifikazio honek:

$$\begin{array}{lcl} \varphi & \equiv & n \geq 1 \\ \psi & \equiv & c \leftrightarrow \forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = B(k) \,) \end{array}$$

i aldagaia indize bezala erabiliz bektorea ezkerretik eskuinera zeharkatzea da aukeratutako estrategia. Gainera, i aldagaiak aztertuko den hurrengo elementua zein den adieraziko du. Bestalde, erantzuna ezezkoa izango dela detektatzen bada uneren batean, gainerako posizioak aztertu gabe bukatu beharko du *while* aginduak. Beraz, i-ren hasierako balioa 1 izango da, eta haren azkeneko balioa gehienez n izango da. Iterazioa bukatu egingo da i-k n balioa hartzen duenean. Bukaerako erantzuna i aldagaiak hartuko duen azken balioari dagokion posizioan gertatzen denaren araberakoa izango da. Estrategia hori formalizatzen duen inbariantea honako hau da:

$$INB \equiv 1 \leq i \leq n \, \wedge \, \forall k \, (\, 1 \leq k \leq i-1 \, \rightarrow \, A(k) = B(k) \,)$$

Inbariantean jasotako informazioa erabiliz, i-k har dezakeen azken balioaren eta i beraren arteko aldea bezala defini dezakegu E borne-adierazpena, hau da, $E \equiv n - i$.

Inbariantea lehenengo aldiz betearaziko duten hasieraketak eratorriko ditugu lehendabizi. Inbariantearen arabera har dezakeen balio txikienarekin hasieratuko dugu i aldagaia, hau da, 1 balioarekin; izan ere, aukeratutako estrategia bektorea ezkerretik eskuinera zeharkatzea da. Esleipen horri dagokion asertzioa honako hau da: $\varphi_1 \equiv INB_i^1 \equiv 1 \leq n$. φ_1 kalkulatzeko $\forall k \, (\, 1 \leq k \leq 0 \, \rightarrow \, A(k) = B(k) \,)$ formula unibertsalaren definizio-eremua hutsa dela eta, ondorioz, formula hori $True$ formularen baliokidea dela kontuan hartu da. φ eta φ_1 baliokideak direnez, hasieraketen eratorpena bukatutzat eman dezakegu.

Bektorea ezkerretik eskuinera zeharkatzen ari garela eta inbariantearen arabera i-ren goiko muga n dela jakinda, i aldagaiak n balioa hartzen duenean iterazioa bukatu egin beharko da. Baina, $A(1..n)$ eta $B(1..n)$ bektoreetako posizioren batean desberdinak diren elementuak aurkitzen badira ere, *while* agindua bukatu egin beharko da. Beraz, $\neg B \equiv i = n \, \vee \, A(i) \neq B(i)$ eta $B \equiv i \neq n \, \wedge \, A(i) = B(i)$. Alde batetik, egiazta dezakegu inbarianteak B baldintzaren definizioa bermatzen duela eta, inbariantea betetzen denean eta B-ren balioa egiazkoa denean, E borne-adierazpenak balio bezala zenbaki arrunt bat hartuko duela. Bestetik, B-ren ebaluazioak faltsua itzultzen

duenean inbarianteak ez du postbaldintza beteko denik bermatzen. Horrek esan nahi du honako propietate hau beteko duen bukaerako aginduren bat behar dela: $\{\, INB \wedge \neg B \,\}$ `Bukaera` $\{\, \psi \,\}$. Iterazioaren bukaera $A(i)$ eta $B(i)$ balioen araberakoa izanda edo $i = n$ betetzen delako izanda, c aldagaian gorde beharreko balioa $A(i) = B(i)$ konparazioaren balioaren araberakoa izango da. Beraz, c aldagaiari konparazio horren balioa esleitu beharko diogula ondoriozta dezakegu. Horrekin bukaerako agindua kalkulatuta gelditzen da, inbarianteak eta B-ren ukapenak honako formula hau inplikatzen baitute:

$$\begin{array}{rcl} \varphi_2 & \equiv & def(A(i) = B(i)) \wedge \psi_c^{A(i)=B(i)} \\ & \equiv & 1 \leq i \leq n \wedge (A(i) = B(i) \leftrightarrow \forall k\, (\, 1 \leq k \leq n \rightarrow A(k) = B(k)\,)\,) \end{array}$$

Iterazioaren gorputzaren kalkulua i-ren balioa handituz hasiko da, horrela E borne-adierazpenaren balioa txikiagotuko baita. INB eta $E < z$ formulekiko esleipen horri dagozkion asertzioak honako hauek dira hurrenez hurren:

$$\begin{array}{rcl} \varphi_3 & \equiv & (INB)_i^{i+1} \equiv (\, -1 \leq i \leq n-1\,) \wedge \forall k\, (\, 1 \leq k \leq i \rightarrow A(k) = B(k)\,) \\ \varphi_4 & \equiv & (E < z)_i^{i+1} \equiv n - i - 1 < z \end{array}$$

Bai $INB \wedge B \rightarrow \varphi_3$ inplikazioa bai $INB \wedge B \wedge E < z \rightarrow \varphi_4$ inplikazioa bete egiten dira, eta horrek iterazioaren gorputza lortu dela esan nahi du. Eratorritako programa honako hau da:

$\{\, \varphi \equiv n \geq 1 \,\}$
  $\{\, \varphi_1 \,\}$
  `i := 1;`
  `while` $\{\, INB \equiv 1 \leq i \leq n \wedge \forall k\, (\, 1 \leq k \leq i-1 \rightarrow A(k) = B(k)\,)\,\}$
   `i /= n and A(i) = B(i)`
  `loop` $\{\, E \equiv n - i \,\}$
    $\{\, \varphi_3 \,\}$ $\{\, \varphi_4 \,\}$
    `i := i + 1;`
  `end loop;`
  $\{\, \varphi_2 \,\}$
  `c := (A(i) = B(i));`
$\{\, \psi \equiv c \leftrightarrow \forall k\, (\, 1 \leq k \leq n \rightarrow A(k) = B(k)\,)\,\}$

7.5.3. x-ren agerpen kopurua bektore batean

Osoa den x zenbakia eta zenbaki osozko $A(1..n)$ bektore ez-hutsa emanda, $A(1..n)$ bektorean x zenbat aldiz agertzen den kalkulatuko duen eta emaitza *kop* aldagaian utziko duen programa bat eratorriko da jarraian. Aurreondoetako espezifikaziotzat honako hau hartuko da:

$$\begin{array}{rcl} \varphi & \equiv & n \geq 1 \\ \psi & \equiv & kop = \mathcal{N}k\, (\, 1 \leq k \leq n \wedge A(k) = x\,) \end{array}$$

Aukera bat da indize bezala i aldagaia erabiliz bektorea ezkerretik eskuinera zeharkatuz joatea, i aldagaiak aztertuko den hurrengo elementua zein den adierazten duelarik. Gainera, ez da elementurik aztertuko agindu iteratiboa hasi aurretik. Honako inbariante honen bidez formaliza daiteke estrategia hori:

$$INB \equiv (\, 1 \leq i \leq n+1 \,) \wedge kop = \mathcal{N}k \,(\, 1 \leq k \leq i-1 \wedge A(k) = x \,)$$

Inbariantean jasotako informazioa erabilita, i aldagaiak hartuko duen azken balioaren eta i-ren uneko balioaren arteko aldea bezala defini dezakegu E borne-adierazpena: $E \equiv n+1-i$.

Lehendabizi hasieraketak eratorriko ditugu $\{\, \varphi \,\}$ `Hasiera` $\{\, INB \,\}$ propietateak egiazkoa izan beharko duela kontuan hartuz. Bektorea ezkerretik eskuinera zeharkatuko denez, $(\, 1 \leq i \leq n+1 \,)$ formula bete dadin i aldagaiari 1 balioa esleituko diogu, hau da, inbariantearen arabera i aldagaiak har dezakeen balio txikiena. Esleipenaren axioma erabiliz honako asertzioa lortuko da:

$$\varphi_1 \equiv INB_i^1 \equiv 0 \leq n \wedge kop = 0$$

φ aurrebaldintzak ez du φ_1 inplikatzen. Inplikazio hori betetzeko kop aldagaiak 0 balioa izan beharko luke. Hortaz, hurrengo hasieraketa kop aldagaiari 0 balioa esleitzea izango da. φ formulak esleipen horri dagokion $\varphi_2 \equiv (\varphi_1)_{kop}^0 \equiv 0 \leq n$ asertzioa inplikatu egiten du.

Bektorea ezkerretik eskuinera zeharkatuko denez eta, inbariantearen arabera, i-ren goiko muga $n+1$ denez, iterazioan jarraitzeko baldintza i-ren balioa eta $n+1$ balioa desberdinak izatea da. Inbariantea betetzen denean B definituta egongo dela eta inbariantea eta B betetzen direnean E-ren balioa negatiboa ez dela izango egiaztatzea berehalakoa da. Gainera, inbarianteak eta B-ren ukapenak postbaldintza inplikatzen dute, eta, ondorioz, bukaerako agindurik ez da behar.

Iterazioaren gorputzaren kalkulua i-ren balioa handituz hasiko da, horrela E-ren balioa txikiagotzea lortuko baita. $INB \wedge B$ formulak ez du honako asertzio hau inplikatzen:

$$\varphi_3 \equiv (INB)_i^{i+1} \equiv (\, 0 \leq i \leq n \,) \wedge kop = \mathcal{N}k \,(\, 1 \leq k \leq i \wedge A(k) = x \,)$$

Baina $INB \wedge B \wedge E = z$ asertzioak beste asertzio hau inplikatzen du:

$$\varphi_4 \equiv (E < z)_i^{i+1} \equiv n - i < z$$

φ_3 betearazteko helburuarekin bi kasu bereiztea beharrezkoa da. $A(i) = x$ baldin bada, inplikazioa ez da beteko; izan ere, inbarianteak dioenez, kop aldagaiak bektoreko $1..i-1$ tartean x zenbat aldiz agertzen den adierazten

du, baina φ_3 asertzioak dioenez, *kop* aldagaiak bektoreko $1..i$ tartean x zenbat aldiz agertzen den adierazten du. Beraz, *kop* aldagaiaren balioari 1 gehitu behar zaio. Bestalde, $A(i) \neq x$ betetzen bada, φ_3 ere beteko da. Hori horrela izanda, baldintzazko agindua eratorri dugu: *then* blokean `kop := kop + 1;` esleipen izango du, eta baldintza (B' bezala izendatuko dugu baldintza hori) $A(i) = x$ konparazioa izango da. Baldintzazko agindu hori zuzena da; izan ere, alde batetik, inbarianteak eta B baldintzak B'-ren definizioa inplikatzen dute, eta, bestetik, $A(i) = x$ betetzen denean $INB \wedge B$ formulak eta $INB \wedge B \wedge E < z$ formulak honako asertzio hauek inplikatzen dituzte hurrenez hurren:

$$\begin{array}{lcll} \varphi_5 & \equiv & (\varphi_3)^{kop+1}_{kop} & \equiv \quad (\ 0 \leq i \leq n\ ) \wedge \\ & & & \qquad (\ kop+1 = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = x\ )\ ) \\ \varphi_6 & \equiv & (\varphi_4)^{kop+1}_{kop} & \equiv \quad n - i < z \end{array}$$

Gainera, $A(i) \neq x$ betetzen denean $INB \wedge B$ formulak eta $INB \wedge B \wedge E < z$ formulak φ_3 eta φ_4 formulak inplikatzen dituzte hurrenez hurren.

Eratorritako programa honako hau da:

$\{\ \varphi \equiv n \geq 1\ \}$
  $\{\ \varphi_2\ \}$
  `kop := 0;`
  $\{\ \varphi_1\ \}$
  `i := 1;`
  `while` $\{\ INB \equiv (\ 1 \leq i \leq n+1\ ) \wedge kop = \mathcal{N}k\ (\ 1 \leq k \leq i-1\ \wedge\ A(k) = x\ )\ \}$
   `i /= n + 1`
  `loop` $\{\ E \equiv n+1-i\ \}$
    `if A(i) = x then`
      $\{\ \varphi_5\ \}$ $\{\ \varphi_6\ \}$
      `kop := kop + 1;`
    `// else (ez dago)`
      $\{\ INB \wedge i \neq n+1 \wedge A(i) \neq x \rightarrow \varphi_3\ \}$
      $\{\ INB \wedge i \neq n+1 \wedge E < v \wedge A(i) \neq x \rightarrow \varphi_4\ \}$
    `end if;`
    $\{\ \varphi_3\ \}$ $\{\ \varphi_4\ \}$
    `i := i + 1;`
  `end loop;`
$\{\ \psi \equiv kop = \mathcal{N}k\ (\ 1 \leq k \leq n\ \wedge\ A(k) = x\ )\ \}$

7.5.4. Ariketak: Bektoreekin kalkuluak egiten dituzten iterazioen eratorpena

1. Formalki eratorri $A(1..n)$ bektorean x balioa agertzen den ala ez erabaki eta erantzuna *agertzen_da* aldagai boolearrean itzuliko duen programa. Hor, x osoa den zenbakia izango da eta $A(1..n)$ zenbaki osozko bektore ez-hutsa izango da. Eratorritako programak eraginkorra izan beharko du, hau da, erantzuna baiezkoa izango dela ikusten bada uneren batean, gainerako posizioak aztertu gabe bukatu beharko du *while* aginduak.

   Aurrebaldintza eta postbaldintza bezala honako formula hauek hartu beharko dira:

   $$\begin{array}{lcl} \varphi & \equiv & n \geq 1 \\ \psi & \equiv & agertzen\_da \leftrightarrow \exists k\,(\,1 \leq k \leq n \wedge A(k) = x\,) \end{array}$$

   Zenbait estrategia kontsidera daitezkeenez, honako inbariante hauen bidez formalizatutako estrategia bakoitzeko eratorpen bat egin:

   1.1. $INB \equiv 1 \leq i \leq n \wedge x \notin A(1..i-1)$

   1.2. $INB \equiv 0 \leq i \leq n \wedge x \notin A(1..i)$

2. Formalki eratorri datu bezala zenbaki osozko $A(0..n)$ bektore ez-hutsa (beraz, $n \geq 0$) eta zero ez den x zenbaki osoa hartu eta

   $$A(0) + A(1) * x^1 + A(2) * x^2 + \cdots + A(n) * x^n$$

   espresioaren balioa p aldagaian lortuko duen programa. Eratorpen hori egiteko honako inbariante hau erabili beharko da:

   $$INB \equiv p = \sum_{k=j}^{n} x^{k-j} * A(k) \wedge 0 \leq j \leq n$$

3. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta bektore horretako hasierako sekzio bakoitzeko elementuen batura zero ez al den erabakiko duen eta erabaki horri dagokion balioa *ez_batu_zero* aldagai boolearrean utziko duen programa.

   Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek erabili beharko dira:

   $$\begin{array}{lcl} \varphi & \equiv & n \geq 1 \\ \psi & \equiv & ez\_batu\_zero \leftrightarrow \forall k\,(\,1 \leq k \leq n \rightarrow (\,\sum_{j=1}^{k} A(j)\,) \neq 0\,) \end{array}$$

Eratorpena honako inbariante honen bidez formalizatutako estrategiari jarraituz egin beharko da:

$$INB \equiv 1 \leq i \leq n \wedge batura = \sum_{k=1}^{i} A(k) \wedge ez\_batu\_zero \leftrightarrow \forall k \, (\, 1 \leq k \leq i \rightarrow (\sum_{j=1}^{k} A(j) \,) \neq 0 \,)$$

7.6. FIBONACCI-REN SEGIDA

Negatiboa ez den x zenbaki osoa emanda, Fibonacciren segidako x-garren terminoa kalkulatuko duen programa eratorriko dugu orain. Segida hori zenbaki arrunten gaineko honako funtzio honen bidez defini daiteke:

$$fib : nat \rightarrow nat$$
$$fib(n) = \begin{cases} n & \text{baldin } n \leq 1 \\ fib(n-1) + fib(n-2) & \text{baldin } n > 1 \end{cases}$$

Definizio horretan $fib(n)$ Fibonacciren segidako n-garren terminoa izango da.

Eratorri beharreko programari dagokion aurre-ondoetako espezifikazioa honako hau izango da:

$$\varphi \equiv x \geq 0$$
$$\psi \equiv f = fib(x)$$

Kalkulu hori burutzeko i eta g aldagai laguntzaileak erabiltzea izan daiteke estrategia bat. Zenbakiak era gorakorrean banan-banan zeharkatuz joateko erabiliko da i aldagaia eta f aldagaiak Fibonacciren segidako i-garren terminoa, hau da, $fib(i)$ izango du une bakoitzean. Segida horretako termino baten kalkuluak, oro har, aurreko bi terminoak behar izaten dituenez, g aldagaia $fib(i+1)$ gordetzeko erabiliko da. Estrategia hori formalizatzen duen inbariantea honako hau da:

$$INB \equiv (\, 0 \leq i \leq x \,) \wedge f = fib(i) \wedge g = fib(i+1)$$

E borne-adierazpena i aldagaiak hartuko duen azken balioaren eta i-ren artekoa aldea bezala definituko dugu, hau da, $E \equiv x - i$.

i, f eta g aldagaiak hasieratu behar dira lehenengo. Hasteko i-ren balioa erabakiko da, f eta g-ren balioak i-ren balioaren menpekoak baitira. Zenbakiak era gorakorrean zeharkatuko direnez, i aldagaiari 0 balioa ematea nahikoa dela ondoriozta dezakegu inbariantetik. Esleipen horri dagokion asertzioa honako hau izango da:

$$\varphi_1 \equiv INB_i^0 \equiv 0 \leq x \wedge f = fib(0) \wedge g = fib(1)$$

Asertzio horretan garbi ikusten da f eta g aldagaiak $fib(0)$ eta $fib(1)$ balioekin, hau da, 0 eta 1 balioekin hasieratu beharko direla hurrenez hurren. Esleipen horiei dagozkien asertzioak $\varphi_2 \equiv (\varphi_1)_f^0$ eta $\varphi_3 \equiv (\varphi_2)_g^1$ dira. φ formulak φ_3 inplikatzen duenez, zuzena den hasieraketa osatu dugu.

Zenbakiak era gorakorrean zeharkatuko direnez eta, inbariantearen arabera, i aldagaiaren goiko muga x denez, i-ren balioa x izatera iristen denean *while* aginduak bukatu egin beharko du. B baldintza, hau da, $i \neq x$ beti definituta egongo da. Gainera, erraz ikus daiteke inbariantea eta B betetzen direnean E borne-adierazpenaren balioa zenbaki arrunt bat izango dela. Bestalde, inbariantea betetzen denean baina B ez denean betetzen, ψ beteko da, eta, ondorioz, bukaerako agindurik ez da beharko.

Iterazioaren gorputza eratortzean, E-ren balioa txikiagotzeko i aldagaiaren balioa unitate batean handitu beharko da. Esleipen horretatik eta INB eta $E < z$ asertzioetatik honako bi asertzioak lortzen dira hurrenez hurren:

$$\begin{array}{lcll} \varphi_4 & \equiv & (INB)_i^{i+1} & \equiv \ (\ 0 \leq i+1 \leq x\ ) \wedge f = fib(i+1) \wedge g = fib(i+2) \\ \varphi_5 & \equiv & (E < z)_i^{i+1} & \equiv \ x - i - 1 < z \end{array}$$

Iterazioaren gorputzaren eratorpena ez da bukatu oraindik, inbarianteak eta agindu iteratiboaren baldintzak ez baitute φ_4 asertzioa inplikatzen: inbarianteak $f = fib(i)$ eta $g = fib(i+1)$ betetzen direla dio, baina φ_4 asertzioan $f = fib(i+1) \wedge g = fib(i+2)$ konjuntzioa daukagu. Konjuntzio hori bete dadin egin beharrekoa bat dator 7.1.1. azpiatalean aurkeztutako *batura_trukaketa* programak egiten duenarekin, eta, beraz, honako baieztapena betetzen dela ziurta dezakegu:

$\{\ (\varphi_4)_{f,g}^{g,f+g}\ \}$
```
  batura_trukaketa(f,g);
```
$\{\ \varphi_4\ \}$

Beste baieztapen hau ere betetzen dela ziurta dezakegu:

$\{\ (\varphi_5)_{f,g}^{g,f+g}\ \}$
```
  batura_trukaketa(f,g);
```
$\{\ \varphi_5\ \}$

$INB \wedge B$ formulak eta $INB \wedge B \wedge E = z$ formulak hurrenez hurren $(\varphi_4)_{f,g}^{g,f+g}$ eta $(\varphi_5)_{f,g}^{g,f+g}$ formulak inplikatzen dituztenez, iterazioaren gorputza eratorri dugu.

Eratorritako programa honako hau da:

$\{\, \varphi \equiv x \geq 0 \,\}$
  $\{\, \varphi_3 \,\}$
  `g := 0;`
  $\{\, \varphi_2 \,\}$
  `f := 0;`
  $\{\, \varphi_1 \,\}$
  `i := 0;`
  `while` $\{\, INB \equiv (0 \leq i \leq x) \wedge f = fib(i) \wedge g = fib(i+1) \,\}$
   `i /= x`
  `loop` $\{\, E \equiv x - i \,\}$
    $\{\, (\varphi_4)^{g,f+g}_{f,g} \,\}$ $\{\, (\varphi_5)^{g,f+g}_{f,g} \,\}$
    `batura_trukaketa(f,g);`
    $\{\, \varphi_4 \,\}$ $\{\, \varphi_5 \,\}$
    `i := i + 1;`
  `end loop;`
$\{\, \psi \equiv f = fib(x) \,\}$

7.7. ARIKETA: FIBONACCIREN TERMINOAK KALKULATZEKO BESTE ERA BAT

1. Formalki eratorri datu bezala negatiboa ez den j zenbakia hartu eta Fibonacciren segidako j-garren terminoa kalkulatuko duen iterazioa. Eratorpen hori honako inbariante hau erabiliz egin beharko da:

$$x * fib(w) + y * fib(w+1) = fib(j) \wedge 0 \leq w \leq j$$

   Aurre-ondoetako espezifikazioa idatzi lehendabizi. Borne-adierazpena w da.

7.8. BEKTOREAK ALDATZEN DITUZTEN PROGRAMEN ERATORPENA

Hasteko, bektore bateko bi osagairen arteko balio-trukaketaren problema aztertuko dugu lehenengo azpiatalean. Horretarako, bektoreen osagaiei egindako esleipenak zuzenak direla frogatzeko beharrezkoa den axioma aurkeztuko dugu lehendabizi. Lehenengo azpiatalean azaldutakoa erabiliz, bigarren azpiatalean herbeheretar banderaren problema (*«The problem of the Dutch National Flag»*, [33]) ebatziko dugu. Problema hori bektore baten osagaien sailkapen bat kontuan hartuz bektorea berrordenatzean datza.

7.8.1. Bektoreen osagaiak: esleipena eta balio-trukaketa

7.1.1. azpiatalean x eta y aldagaien balioak trukatzen dituen programa bat eratorri da. Bektoreen osagaien arteko balio-trukaketa zailagoa edo korapilatsuagoa denez, azpiatal horretan x eta y aldagai indibidualak direla azpimarratu da. Bektore bateko osagaien arteko balio-trukaketak ez ezik, osagai bati balio bat esleitzearen azpiproblemak ere tratamendu espezializatua behar du. Azpiatal honetan, bektoreen osagaien gaineko esleipenaren axioma aurkeztuko dugu lehenengo, eta gero bektore bateko bi osagairen arteko balio-trukaketa egiten duen programa bat eratorriko dugu.

3. kapituluan aurkeztutako esleipenaren axiomak `x := t` erako esleipenentzat balio du, hor x oinarrizko mota bateko aldagai indibiduala izanda. Bektore bateko osagai bati balio-esleipena egitean (`A(i) := t` erako esleipen bat, alegia), esleipenaren axiomaren erabilerak arazoak ematen ditu. Arazo horiek i indizeak A bektorearen eremuaren edo mugen barruan egon beharko duela bermatu beharretik haratago doaz. Problematika hori aurkezteko, 3. kapituluko esleipenaren axioma erabiliz `A(i) := t` erako esleipenak tratatzean ager daitezkeen bi motatako arazoak erakusten dituzten adibideak emango ditugu. Axioma hori propietate batzuk frogatzeko ahulegia dela ikusiko dugu hasteko. Adibidez, zenbaki osozko edozein $A(1..n)$ bektore ez-hutsentzat era nabarmenean zuzena den honako propietate honen zuzentasuna ezin da frogatu:

$\{\, 1 \le i \le n \,\wedge\, i = j \,\}$
`  A(i) := 0;`
$\{\, A(j) = 0 \,\}$

Demagun bektoreko osagaia ondo definituta dagoela (hau da, indizea bektorearen eremuaren edo mugen barruan dagoela) ziurtatuko lukeen baldintza ere ager dadin, esleipenaren axioma orokortu egin dugula. Postbaldintzatik abiatuz eta esleipenaren axioma erabiliz honako asertzio hau lortuko litzateke:

$$1 \le i \le n \,\wedge\, A(j) = 0$$

Baina $1 \le i \le n \wedge i = j$ aurrebaldintzak ez du inplikatzen asertzio hori. Postbaldintzan $A(i)$ ez denez era esplizituan agertzen, $A(i)$ ordezkatzeak ez du inolako eraginik eta hortik dator esleipenaren axiomaren ahultasuna. Honako beste adibide honetan ere antzekoa gertatzen da:

$\{\, n \ge 1 \,\wedge\, 1 \le i \le n \,\wedge\, \forall \ell\, (\, 1 \le \ell < i \,\rightarrow\, A(\ell) \ge 1\,) \,\wedge\, A(i) \le 0 \,\wedge$
$\qquad\qquad \forall \ell\, (i+1 \le \ell \le n \,\rightarrow\, A(\ell) \ge 1\,)\,\}$
`  A(i) := 1;`
$\{\, \forall \ell\, (\, 1 \le \ell \le n \,\rightarrow\, A(\ell) \ge 1\,)\,\}$

ψ postbaldintzan $A(i)$ era esplizituan ez denez agertzen, aldagai indibidualei egindako esleipenetan ez bezala, $\psi^1_{A(i)}$ ez da lortzen ordezkatze sistematiko baten bidez. Ohartu beharrekoa da aurrebaldintzako bi azpiformula unibertsalak eta $A(i) \geq 1$ era esplizituan ipiniz osatutako konjuntzioa idatz dezakegula emandako postbaldintzaren ordez, baliokideak baitira. Horrek adibide honetako arazoa konponduko luke, baina ez aurreko adibideko arazoa. Aurreko adibideko arazoa konpontzeko hirukote baliokide bat idatzi beharko litzateke, baina hirukote horretako postbaldintza ez litzateke izango hasieran emandako postbaldintzaren baliokidea. Gainera, oraindik larriagoa den bigarren arazo mota bat ere ager daiteke; izan ere, bektoreen osagaiei egindako esleipenak aldagai indibidualei egindakoak bezala tratatuz gero, zuzenak ez diren propietateen zuzentasuna frogatu ahal izango dugu. Har dezagun, adibidez, honako hirukote hau:

$\{\, n \geq 2 \wedge A(1) = 2 \wedge A(2) = 2 \,\}$
`A(A(2)) := 1;`
$\{\, A(A(2)) = 1 \,\}$

Hor $A(1..n)$ gutxienez bi osagai dituen zenbaki osozko bektore bat da. Hirukote hori faltsua dela ikustea erraza da, esleipena burutu ondoren $A(1) = 2 \wedge A(2) = 1$ beteko baita, eta, ondorioz, $A(A(2))$-ren balioa 2 izango da. Baina, hala eta guztiz ere, esleipenaren axioma erabiliz zuzena dela froga dezakegu (axioma horri indizeek bektorearen mugen barruan egon beharko dutela dioen baldintza erantsiz). Izan ere, aurrebaldintzak era nabarmenean inplikatzen duen $1 \leq 2 \leq n \wedge 1 \leq A(2) \leq n \wedge 1 = 1$ asertzioa lortuko genuke postbaldintzatik.

Aurreko adibideetan erakutsi den problematika $A(1..n)$ erako bektoreak $A(1)$, $A(2)$, …, $A(n)$ aldagai sinpleez osatutako aldagai konposatutzat hartzetik dator. Soluzioa $A(1..n)$ erako bektoreak balio egituratuak hartzen dituzten A aldagai sinpletzat hartzea da. Beste era batera esanda, bektore baten osagai bati egindako esleipena bektorearen balio osoaren aldaketa bezala ikusi edo ulertu behar da. Ikuspuntu horrekin bat datorren axioma eman ahal izateko, honako notazio berri hau behar dugu: A bektorearen definizio-eremua $1..n$ baldin bada, $alpha(A,i,t)$ espresioak definizio-eremu bera duen honako bektore hau adieraziko du:

$$alpha(A,i,t)(j) = \begin{cases} t & \text{baldin } j = i \\ A(j) & \text{baldin } j \neq i \text{ eta } 1 \leq j \leq n \end{cases}$$

Notazio hori erabiliz, $A(1..n)$ bektoreko osagai bati egindako esleipenari dagokion axioma honela formula daiteke:

(BEA) $\boxed{\{\, def(A(t_1)) \wedge def(t_2) \wedge \varphi_A^{alpha(A,t_1,t_2)} \,\}\ \texttt{A(t}_\texttt{1}\texttt{) := t}_\texttt{2}\texttt{;}\ \{\, \varphi \,\}}$

Bata bestearen atzetik burututako esleipenetara ere era errazean egoki daiteke notazio hori:

$\{\, \varphi \,\}$
  $\{\, \varphi_2 \equiv 1 \le j \le n \wedge 1 \le i \le n \wedge (\, \psi_A^{alpha(A,i,1)})_A^{alpha(A,j,2)} \,\}$
  `A(j) := 2;`
  $\{\, \varphi_1 \equiv 1 \le i \le n \wedge \psi_A^{alpha(A,i,1)} \,\}$
  `A(i) := 1;`
$\{\, \psi \,\}$

Hor $alpha(alpha(A,j,2),i,1)$ erako espresioak lortuko dira.

Honako era honetako aldibereko esleipenak hartzen baditugu:

`(A(`$\mathtt{j_1}$`),A(`$\mathtt{j_2}$`),...,A(`$\mathtt{j_k}$`)):= (`$\mathtt{t_1}$`, `$\mathtt{t_2}$`,...,`$\mathtt{t_k}$`);`

eta $i,\ell \in \{1,2,\ldots,k\}$ eta $i \neq \ell$ baldintzak betetzen dituzten (i,ℓ) bikote guztientzat j_i terminoaren balioa eta j_ℓ terminoaren balioa desberdinak izanda, ψ postbaldintza batekiko lortuko litzatekeen espresioa honako hau izango litzateke:

$$def(A(j_1)) \wedge \ldots \wedge def(A(j_k)) \wedge def(t_1) \wedge \ldots \wedge def(t_k) \wedge \psi_A^{alpha(A,[j_1,j_2,\ldots,j_k],[t_1,t_2,\ldots,t_k])}$$

Hor $alpha(A,[j_1,j_2,\ldots,j_k],[t_1,t_2,\ldots,t_k])$ espresioa honako definizio honen bidez adierazten den bektorea izango da:

$$alpha(A,[j_1,j_2,\ldots,j_k],[t_1,t_2,\ldots,t_k])(j) = \begin{cases} t_\ell & \text{baldin } j = j_\ell \\ A(j) & \text{baldin } j \notin [j_1,j_2,\ldots,j_k] \text{ eta } 1 \le j \le n \end{cases}$$

Atal honetan aurkeztutako notazioak erakusteko, gutxienez bi osagai dituen $A(1..n)$ bektoreko i eta j bi posizio desberdinetako balioak trukatzen dituen honako programa honen zuzentasuna egiaztatuko dugu jarraian:

$\{\, 1 \le i < j \le n \wedge A(i) = a \wedge A(j) = b \,\}$
  `lag := A(j);`
  `A(j) := A(i);`
  `A(i) := lag;`
$\{\, A(i) = b \wedge A(j) = a \,\}$

Postbaldintzatik abiatuz eta bektoreen osagaien gaineko esleipenaren axioma erabiliz, honako asertzio hau lortuko genuke hasteko:

$$\varphi_1 \equiv 1 \le i \le n \wedge alpha(A,i,lag)(i) = b \wedge alpha(A,i,lag)(j) = a$$

7.1. irudia. Herbeheretar bandera.

Asertzio horretan $alpha(A,i,lag)(i) = b$ espresioa $lag = b$ espresioaren baliokidea da eta $alpha(A,i,lag)(j) = a$ espresioa $A(j) = a$ espresioaren baliokidea da. φ_1 asertziotik abiatuz eta prozesu bera eginez, honako asertzioa lortuko da:

$$\varphi_2 \equiv 1 \leq j \leq n \wedge 1 \leq i \leq n \wedge alpha(alpha(A,j,A(i)),i,lag)(i) = b \wedge alpha(alpha(A,j,A(i)),i,lag)(j) = a$$

Espresio horretan $alpha(alpha(A,j,A(i)),i,lag)(i) = b$ berdintzaren esanahia $lag = b$ da eta $alpha(alpha(A,j,A(i)),i,lag)(j) = a$ berdintzaren esanahia $A(i) = a$ da. Bukatzeko, 3. kapituluko esleipenaren axioma erabiliz beste asertzio hau kalkulatuko litzateke:

$$\varphi_3 \equiv 1 \leq j \leq n \wedge 1 \leq i \leq n \wedge alpha(alpha(A,j,A(i)),i,A(j))(i) = b \wedge alpha(alpha(A,j,A(i)),i,A(j))(j) = a$$

Asertzio horretan $alpha(alpha(A,j,A(i)),i,A(j))(i) = b$ espresioak $A(j) = b$ dela adierazten du eta $alpha(alpha(A,j,A(i)),i,A(j))(j) = a$ berdintzak $A(i) = a$ dela adierazten du. Aurrebaldintzak φ_3 asertzioa inplikatzen duenez, programa zuzena dela frogatuta gelditzen da.

Programa hori parametrizatutako *bektore_trukaketa*(A,i,j) programatzat hartzen badugu, $\beta \rightarrow i \neq j$ inplikazioa betetzen duen eta, gainera, i eta j aldagai atxiki bezala ez dituen edozein β formularentzat honako hau egiazkoa izango da:

$\{ def(A(i)) \wedge def(A(j)) \wedge \beta_A^{alpha(A,[i,j],[A(j),A(i)])} \}$
`bektore_trukaketa(A,i,j);`
$\{ \beta \}$

Jarraian datorren 7.8.2. azpiatalean erabiliko da *bektore_trukaketa* eragiketaren propietate hori.

7.8.2. Herbeheretar banderaren problema

Gorriak, zuriak eta urdinak bezala era bateraezinean edo baztertzailean sailka daitezkeen elementuak dituen $A(1..n)$ bektorea hartu eta kolorearen arabera elementuak elkartzean edo taldekatzean datza herbeheretar banderaren

problema (ikus 7.1. irudia). Bektorearen ezkerreko aldean elementu gorriak, erdialdean zuriak eta eskuineko aldean urdinak taldekatuta uztea da helburua. Beraz, elkarrekiko baztertzaileak edo bateraezinak diren hiru propietate betetzen dituzten elementuak sailkatzea eskatzen duen problemaren aurrean gaude. Esate baterako, $A(1..n)$ zenbaki osozko bektorea baldin bada, gorria izatearen propietatea 1 baino txikiagoa izatearekin pareka genezake, zuria izatearen propietatea zenbaki lehena izatearekin (1 baino handiagoa, hortaz) eta urdina izatearen propietatea 1 baino handiagoa edo berdina den zenbaki ez-lehena izatearekin.

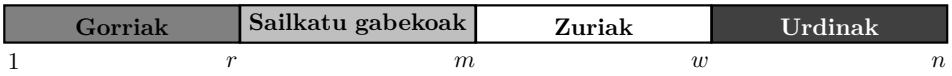
Jarraian zehazten den bezala formaliza daiteke herbeheretar banderaren problemari dagokion aurre-ondoetako espezifikazioa:

$$\begin{array}{lcl}\varphi & \equiv & n \geq 1 \wedge \forall k\,(\,1 \leq k \leq n \rightarrow A(k) = a_k\,) \wedge \mathit{hirukolore}(A(1..n)\,)\\ \psi & \equiv & 0 \leq r \leq w \leq n \wedge \mathit{permutazioa}(A(1..n),(a_1,a_2,\ldots,a_n)) \wedge\\ & & \quad \mathit{gorriak}(A(1..n),1,r) \wedge \mathit{zuriak}(A(1..n),r+1,w) \wedge\\ & & \quad \mathit{urdinak}(A(1..n),w+1,n)\end{array}$$

Formula horietan agertzen diren predikatuen definizioa honako hau da:

$$\begin{array}{l}\mathit{permutazioa}(A(1..n),(a_1,a_2,\ldots,a_n)) \equiv\\ \quad \forall k(\,1 \leq k \leq n \rightarrow \mathcal{N}j\,(\,1 \leq j \leq n \wedge A(k) = A(j)) =\\ \qquad\qquad\qquad\qquad \mathcal{N}i\,(\,1 \leq i \leq n \wedge A(k) = a_i\,)\,)\\ \mathit{hirukolore}(A(1..n)) \equiv\\ \quad \forall k\,(\,1 \leq k \leq n \rightarrow\\ \qquad (\,(\,\mathit{gorria}(A(k)) \wedge \neg\mathit{zuria}(A(k)) \wedge \neg\mathit{urdina}(A(k))\,) \vee\\ \qquad\ (\,\mathit{zuria}(A(k)) \wedge \neg\mathit{gorria}(A(k)) \wedge \neg\mathit{urdina}(A(k))\,) \vee\\ \qquad\ (\,\mathit{urdina}(A(k)) \wedge \neg\mathit{gorria}(A(k)) \wedge \neg\mathit{zuria}(A(k))\,)\,)\,)\\ \mathit{gorriak}(A(1..n),i,j) \equiv\\ \quad 1 \leq i \leq n \wedge i-1 \leq j \leq n \wedge \forall k\,(\,i \leq k \leq j \rightarrow \mathit{gorria}(A(k))\,)\\ \mathit{zuriak}(A(1..n),i,j) \equiv\\ \quad 1 \leq i \leq n \wedge i-1 \leq j \leq n \wedge \forall k\,(\,i \leq k \leq j \rightarrow \mathit{zuria}(A(k))\,)\\ \mathit{urdinak}(A(1..n),i,j) \equiv\\ \quad 1 \leq i \leq n \wedge i-1 \leq j \leq n \wedge \forall k\,(\,i \leq k \leq j \rightarrow \mathit{urdina}(A(k))\,)\end{array}$$

Ohartu beharrekoa da berrordenatuta geratuko den bektoreaz gain, r eta w aldagaiak ere emaitzak izango direla, berrordenatutako bektorean bereiziko diren hiru zatiak mugatzeko erabiliko baitira, eta, horregatik, ψ postbaldintzan agertzen dira. Bestalde, j-ren balioa i-rena baino txikiagoa izatea onartzen da *gorriak*, *zuriak* eta *urdinak* predikatuen definizioetan, eta baldintza hori betetzen denean definizio-eremu hutsa izango dute definizio horietako formula unibertsalek. *gorria*, *zuria* eta *urdina* predikatuei dagokienez, aurredefinitutzat hartuko ditugu, eta, lehen aipatu den bezala, edozein propietate adieraz dezakete.



7.2. irudia. Bektoreko egoeraren deskribapena.

Bektorean lau zati bereiztea da jarrai daitekeen estrategietako bat (ikus 7.2. irudia): lehenengo zatia elementu gorriz bakarrik eratuta egongo litzateke, bigarren zatian hiru koloreetako elementuak onartuko lirateke, hirugarren zatiak elementu zuriak bakarrik edukiko lituzke eta laugarren zatian elementu urdinak besterik ez lirateke egongo. Zati gorriko azken posizioa eta zati zuriko azken posizioa zeintzuk diren adierazten duten r eta w aldagaiez gain, m beste aldagai bat ere beharko da. Hiru koloreetako elementuak nahastuta dituen zatiaren azken posizioa zein den zehaztuko du m aldagaiak, eta zati hori eskuinetik ezkerrera zeharkatuz joateko erabiliko da.

Iterazio bakoitzean m posizioko elementua bere koloreari dagokion zatian kokatu beharko da. Kasu-bereizketa egitea eskatzen du horrek: gorria baldin bada, $r+1$ posizioko elementuarekin posizio-trukaketa egin beharko da eta r eguneratu beharko da; zuria baldin bada, posizio egokian dago eta m eguneratzearekin nahikoa izango da; urdina baldin bada, w posizioko elementuarekin posizio-trukaketa egin eta m eta w eguneratu beharko dira. Estrategia hori honako inbariantearen bidez formaliza daiteke:

$$\begin{aligned} INB \quad \equiv \quad & 0 \leq r \leq m \leq w \leq n \wedge \mathit{permutazioa}(A(1..n),(a_1,a_2,\ldots,a_n)) \wedge \\ & \mathit{hirukolore}(A(1..n)) \wedge \mathit{gorriak}(A(1..n),1,r) \wedge \\ & \mathit{zuriak}(A(1..n),m+1,w) \wedge \mathit{urdinak}(A(1..n),w+1,n) \end{aligned}$$

Inbariantea finkatu ondoren, m-k hartuko duen azken balioaren eta m-ren arteko aldea bezala defini dezakegu E borne-adierazpena, hau da, $E \equiv m-r$.

r, m eta w aldagaiak hasieratzea beharrezkoa da φ aurrebaldintza betetzen duen egoera batetik abiatuz inbariantea bete dadin. Hasteko r-ren balioa erabakiko dugu. r aldagaia 0 balioarekin hasieratzea nahikoa da $0 \leq r \leq m \leq w \leq n$ eta $\mathit{gorriak}(A(1..n),1,r)$ bete ditzan. Hori egin ondoren, $\varphi_1 \equiv INB_r^0$ asertzioa kalkulatuko genuke. w aldagaiak φ_1 formulak dioena bete dezan, hau da, $0 \leq m \leq w \leq n$ eta $\mathit{urdinak}(A(1..n),w+1,n)$ bete daitezen, n balioarekin hasiera dezakegu. m aldagaiak $0 \leq m \leq n$ eta $\mathit{zuriak}(A(1..n),m+1,n)$ bete behar dituela adierazten du $\varphi_2 \equiv (\varphi_1)_w^n$ asertzioak. m aldagaia n balioarekin hasieratzea da gelditzen zaigun aukera, bai baitakigu bektoreko posizioak eskuinetik ezkerrera zeharkatzeko erabiliko dela. φ aurrebaldintzak hasieraketa horri dagokion $\varphi_3 \equiv (\varphi_2)_m^n$ asertzioa inplikatu egiten du. Hor, bai $\mathit{gorriak}(A(1..n),1,0)$, bai $\mathit{zuriak}(A(1..n),n+1,n)$

eta bai $urdinak(A(1..n), n+1, n)$ terminoen azpian dauden konjuntzioz eratutako formulen osagaietako bat definizio-eremu hutseko formula unibertsala da.

Lehenago zehaztu den estrategiari jarraituz, eskuinetik ezkerrera bektoreko posizioak zeharkatuz joango da m. Beraz, m aldagaiaren balioa r izatera iristen denean agindu iteratiboak bukatu egin beharko du, r baita m-ren beheko muga inbariantearen arabera. Ondorioz, $B \equiv m \neq r$ izango da *while*-ean jarraitzeko baldintza. B definituta egongo da inbariantea betetzen den egoera guztietan, eta, gainera, E borne-adierazpena negatiboa ez izatea bermatzen du inbariantea eta B betetzeak. Bestalde, bukaerako agindurik ez da beharko, inbariantea betetzen denean eta B ez denean betetzen, ψ egiazkoa izango baita.

Iterazioaren gorputza osatuko duten aginduak eratorriko ditugu bukatzeko. Inbariantea kontserbatzea eta borne-adierazpenaren balioa beheratzea lortu behar du agindu-segida horrek bere exekuzioa B betetzen den egoera batean hasten denean. r-ren balioa handitu edo m-rena txikiagotu dezakegu E-ren balioak behera egin dezan. m posizioko elementuaren kolorearen arabera dago ekintza bat edo bestea egitearen egokitasuna. Hori dela-eta, kasu-azterketa egingo dugu:

- $A(m)$ gorria baldin bada, $r+1$ posizioan ipini behar da elementu hori eta, horregatik, r-ren balioa handitu egin beharko da. Gehikuntza horri dagozkion asertzioak $\varphi_4 \equiv INB_r^{r+1}$ eta $\varphi_5 \equiv (E < z)_r^{r+1}$ izango lirateke. $INB \wedge B \wedge E = z \wedge gorria(A(m))$ formulak φ_5 inplikatzen du, baina $INB \wedge B \wedge gorria(A(m))$ formulak ez du φ_4 inplikatzen. A bektoreko $r+1$ eta m posizioetako balioak lekuz trukatu beharko dira φ_4 bete dadin. Horrela, $r+1$ posizioraino elementu gorriak bakarrik egotea lortuko da, eta, gainera, bektoreak hasierako balioekiko permutazioa izaten jarraituko du. 7.8.1. azpiatalean definitu den *bektore_trukaketa* modulua erabil dezakegu trukaketa hori burutzeko. Beraz, honako hau beteko dela ziurta dezakegu:

  $\{ \varphi_6 \equiv def(A(r+1)) \wedge def(A(m)) \wedge (\varphi_4)_A^{alpha(A,[r+1,m],[A(m),A(r+1)])}$
  `bektore_trukaketa(A,r+1,m);`
  $\{ \varphi_4 \}$

  Eta beste hau ere beteko da:

  $\{ \varphi_7 \equiv def(A(r+1)) \wedge def(A(m)) \wedge (\varphi_5)_A^{alpha(A,[r+1,m],[A(m),A(r+1)])}$
  `bektore_trukaketa(A,r+1,m);`
  $\{ \varphi_5 \}$

$INB \wedge B \wedge gorria(A(m))$ formulak φ_6 inplikatzen duenez, eta, beste aldetik, $INB \wedge B \wedge E = z \wedge gorria(A(m))$ formulak ere φ_7 inplikatzen duenez, kasu hau osatu dugu.

- $A(m)$ zuria baldin bada, m-ren balioa txikiagotu egin beharko da. $\varphi_8 \equiv INB_m^{m-1}$ eta $\varphi_9 \equiv (E < z)_m^{m-1}$ dira txikiagotze horri dagozkion asertzioak. Alde batetik, $INB \wedge B \wedge zuria(A(m))$ formulak φ_8 inplikatzen du, eta, bestetik, $INB \wedge B \wedge E = z \wedge zuria(A(m))$ formulak ere φ_9 inplikatzen du. Beraz, besterik ez da egin behar.

- $A(m)$ urdina baldin bada, ezingo da gorriez osatutako zatia zabaldu, hau da, r ez da handitu behar, eta horrek m-ren balioa txikiagotu beharko dela adierazten digu. $\varphi_{10} \equiv INB_m^{m-1}$ eta $\varphi_{11} \equiv (E < z)_m^{m-1}$ dira INB inbariantea eta $E < z$ formularekiko txikiagotze horri dagozkion asertzioak. $INB \wedge B \wedge E = z \wedge urdina(A(m))$ formulak φ_{11} inplikatzen du, baina $INB \wedge B \wedge urdina(A(m))$ formulak ez du φ_{10} inplikatzen, $zuriak(A(1..n), m, w)$ betetzen dela ziurtatzen baitu φ_{10} formulak, baina $zuriak(A(1..n), m+1, w)$ betetzen dela eta m posizioan, ez elementu zuria baizik eta urdina dagoela ziurtatzen du $INB \wedge B \wedge urdina(A(m))$ formulak. Bektoreak hasierako balioen permutazioa izaten jarraituz m posizioko elementua zuria izan dadin, m posizioko elementu urdina eta w posizioko elementu zuria lekuz truka daitezke. Horrela, $permutazioa(A(1..n), (a_1, a_2, \ldots, a_n))$ propietateari eutsi egingo zaio eta m posizioan elementu zuria edukiko da. Formalki, 7.8.1. azpiatalean definitutako *bektore_trukaketa* modulua erabil dezakegu $A(1..n)$ bektoreko m eta w posizioetako balioak trukatzeko. Beraz, honako hau beteko dela ziurta dezakegu:

$$\{\, \varphi_{12} \equiv def(A(m)) \wedge def(A(w)) \wedge (\varphi_{10})_A^{alpha(A,[m,w],[A(w),A(m)])} \,\}$$
```
bektore_trukaketa(A,m,w);
```
$$\{\, \varphi_{10} \,\}$$

Eta beste hau ere beteko da:

$$\{\, \varphi_{13} \equiv def(A(m)) \wedge def(A(w)) \wedge (\varphi_{11})_A^{alpha(A,[m,w],[A(w),A(m)])} \,\}$$
```
bektore_trukaketa(A,m,w);
```
$$\{\, \varphi_{11} \,\}$$

Hala ere, $INB \wedge B \wedge urdina(A(m))$ formulak ez du φ_{12} asertzioa inplikatzen, $alpha(A, [m, w], [A(w), A(m)])$ bektoreko m eta w (biak barne)

posizioen arteko osagai guztiak zuriak izatea ez baita betetzen. Elementu urdina daukagu $alpha(A,[m,w],[A(w),A(m)])$ bektoreko w posizioan hain zuzen ere. Helburua m eta w posizioen artean elementu zuriak edukitzea denez, w-ren balioa txikiagotuz lortuko dugu helburu hori betetzea. Esleipenaren axioma erabiliz $\varphi_{14} \equiv (\varphi_{12})_w^{w-1}$ asertzioa eta $\varphi_{15} \equiv (\varphi_{13})_w^{w-1}$ asertzioa lortzen dira. $INB \wedge B \wedge urdina(A(m))$ formulak eta $INB \wedge B \wedge E = z \wedge urdina(A(m))$ formulak hurrenez hurren φ_{14} asertzioa eta φ_{15} asertzioa inplikatzen dituztenez, kasu honi dagokion eratorpena bukatutzat eman dezakegu.

Eratorritako programa honako hau da:

```
{ φ }
  { φ3 }
  m := n;
  { φ2 }
  w := n;
  { φ1 }
  r := 0;
  while  { INB }
   m /= r
  loop   { E }
    if gorria(A(m)) then
      { φ6 }  { φ7 }
      bektore_trukaketa(A,r+1,m);
      { φ4 }  { φ5 }
      r := r+1;
    elsif zuria(A(m)) then
      { φ8 }  { φ9 }
      m := m-1;
    else
      { φ14 }  { φ15 }
      w := w-1;
      { φ12 }  { φ13 }
      bektore_trukaketa(A,m,w);
      { φ10 }  { φ11 }
      m := m-1;
    end if;
  end loop;
{ ψ }
```

7.8.3. Ariketak: Bektoreak berrordenatzen

1. Formalki eratorri datu bezala positiboak diren zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta elementu bikoiti denak hasierako posizioetan (ezkerreko aldean) eta elementu bakoiti denak elkarren jarraian bukaeran (eskuineko aldean) ipiniz bektorea berrordenatuko duen programa. Eratorritako iterazioak begizta bakarrekoa izan beharko du.

2. Formalki eratorri zenbaki osozko $A(1..n)$ bektore ez-hutsa eta x zenbaki osoa hartu eta hasierako posizioetan (ezkerreko aldean) x baino txikiagoak diren elementuak, erdian x-ren berdinak diren elementuak eta bukaeran (eskuineko aldean) x baino handiagoak diren elementuak kokatuz bektorea berrordenatuko duen programa. Eratorritako iterazioak begizta bakarrekoa izan beharko du. Kontuan izan, bektoreko elementuak goranzko ordenan ordenatuta ipini beharrik ez dagoela.

7.9. ZENBAKI LAGUNAK

Bi zenbaki emanda, zenbaki horiek *lagunak* diren ala ez erabakiko duen programa iteratibo bat eratorriko dugu formalki atal honetan. Positiboak diren x eta y zenbaki osoak *lagunak* direla esango dugu, x-ren berezko zatitzaile positiboen[1] batura y baldin bada eta y-ren berezko zatitzaile positiboen batura x baldin bada. Esate baterako, 220 eta 284 zenbaki lagunak dira.

y zenbakia x baino handiagoa dela eta biak positiboak direla jakinda, *zlag* aldagai boolearrean x eta y zenbaki osoak lagunak diren ala ez erabakitzearen problemaren espezifikazioa honela formula daiteke:

$$\begin{array}{lcl} \varphi & \equiv & 1 \leq x \leq y \\ \psi & \equiv & zlag \leftrightarrow (\, x = \sum\limits_{\substack{1 \leq k \leq y-1 \,\wedge \\ y \,\%\, k = 0}} k \;\wedge\; y = \sum\limits_{\substack{1 \leq k \leq x-1 \,\wedge \\ x \,\%\, k = 0}} k) \end{array}$$

Jarrai daitekeen estrategia bat da j aldagaia erabiliz 1etik $y-1$-erainoko zenbakiak zeharkatuz joatea, tarte horretako zenbaki bakoitzak x eta y zatitzen dituen aztertzea eta hurrenez hurren x-ren eta y-ren zatitzaileen batura gordetzeko diren bx eta by aldagaiak eguneratzea. Dagoeneko aztertu den azpitartearen goiko muga zein den adieraziko du j aldagaiak. Hori delaeta, 0 izango da goiko muga hori hasieran. Gainera, kalkulatzeko prozesuan zehar zenbakietako baten zatitzaileen baturak beste zenbakia gainditzen badu zenbaki horiek lagunak ez direla jakingo dugu, eta bukatu egin beharko da prozesua.

1. $[1..x-1]$ tarteko x-ren zatitzaileak.

Honako inbariante honen bidez formaliza daiteke estrategia hori:

$$\begin{aligned} INB \;\equiv\; & 1 \le x \le y \;\wedge\; 0 \le j \le y-1 \;\wedge \\ & by = \sum_{\substack{1 \le k \le j \wedge \\ y \,\%\, k = 0}} k \;\wedge\; bx = \sum_{\substack{1 \le k \le j \;\wedge \\ x \,\%\, k = 0}} k \;\wedge \\ & x \ge \sum_{\substack{1 \le k \le j-1 \;\wedge \\ y \,\%\, k = 0}} k \;\wedge\; y \ge \sum_{\substack{1 \le k \le j-1 \;\wedge \\ x \,\%\, k = 0}} k \end{aligned}$$

Inbariantea definitu ondoren, j aldagaiak hartuko duen azken balioaren eta j beraren arteko aldea bezala defini dezakegu E borne-adierazpena, hau da, $E \equiv (y-1)-j$.

Hasteko, inbariantean dugun informaziotik eta j aldagaiak zenbakiak txikienetik handienera zeharkatuko dituela jakitetik, j aldagaia 0 balioarekin hasieratzea komeni dela ondoriozta dezakegu. Hori egin ondoren, by eta bx aldagaiak 0 balioarekin hasieratu behar direla adierazten digu esleipenaren axioma erabiliz lortutako $\varphi_1 \equiv INB_j^0$ formulak. Bi hasieraketa horietatik $\varphi_2 \equiv (\varphi_1)_{by}^0$ eta $\varphi_3 \equiv (\varphi_2)_{bx}^0$ formulak lortuko dira. φ formulak φ_3 inplikatzen duenez, hiru esleipen horiek burutu ondoren inbariantea bete egingo dela ziurta dezakegu.

j aldagaia $y-1$ baliora iristea edo baturetakoren batek (by-k edo bx-k) dagokion zenbakia (x edo y hurrenez hurren) gainditzea da iterazioa bukatzeko bete beharreko baldintza. Beraz, *while*-ean jarraitzeko baldintza hiru osagai dituen disjuntzio horren ukapena izango da:

$$B \;\equiv\; j \neq y-1 \;\wedge\; by \le x \;\wedge\; bx \le y$$

Inbariantea betetzen den egoera guztietan B definituta egongo da, eta, gainera, B eta inbariantea betetzen direnean E borne-adierazpenak zenbaki arrunt bat izango du balio bezala. Aldiz, inbariantea betetzen denean baina B faltsua denean, ψ ez da beteko, eta, ondorioz, ψ betearaziko duen agindu bat beharko da bukaeran. $\neg B$ espresio boolearra hiru arrazoi desberdinengatik bete daiteke, baina hiru kasuetan $zlag$ aldagaiaren balioak $by = x \;\wedge\; bx = y$ espresioaren balioarekin bat etorri behar du. Beraz, horri dagokion esleipena ipiniko da. Inbarianteak eta B-ren ukapenak osatutako konjuntzioak $\varphi_4 \equiv \psi_{zlag}^{by=x \;and\; bx=y}$ formula inplikatzen dutenez, agindu iteratiboa bukatu eta aipatutako bukaerako esleipena burutu ondoren ψ beteko dela ziurta dezakegu.

Azkenik, B betetzen den kasuetan, iterazioaren gorputzak inbariantea kontserbatu eta E borne-adierazpenaren balioa txikiagotu egin beharko du. Bigarren helburua j-ren balioa handituz lor daiteke. $\varphi_5 \equiv INB_j^{j+1}$ asertzioa eta $\varphi_6 \equiv (E < z)_j^{j+1}$ asertzioa aztertzen baditugu, bigarren helburua bete egiten dela ikus dezakegu, baina ez lehenengo helburua, hau da, inbariantea kontserbatzea. φ_5 betearazteko kasu-azterketa egin beharra dago: $j+1$ balioak y eta x biak zatitzen baditu eta, gainera, $j+1$ eta x desberdinak badira, bai by-ri eta bai bx-ri $j+1$ batu beharko zaie. Aldiz, $j+1$ balioak y bakarrik zatitzen badu edo y eta x biak zatitzen baditu baina $j+1=x$ betetzen bada, orduan by-ri bakarrik batu beharko zaio $j+1$. Bestalde, $j+1$ balioak x bakarrik zatitzen badu eta, gainera, $j+1 \neq x$ baldin bada, orduan bx-ri bakarrik batu beharko zaio $j+1$. Beste edozein kasutan ez da ezer egin beharko. Lehenengo kasuan, $\varphi_7 \equiv (\varphi_5)_{bx}^{bx+j+1}$, $\varphi_8 \equiv (\varphi_6)_{bx}^{bx+j+1}$, $\varphi_9\ (\varphi_7)_{by}^{by+j+1}$ eta $\varphi_{10} \equiv (\varphi_8)_{by}^{by+j+1}$ asertzioak lortuko dira, eta $INB \wedge B \wedge B_1$ eta $INB \wedge B \wedge E = z \wedge B_1$ formulek azkeneko biak (φ_9 eta φ_{10}) inplikatzen dituzte hurrenez hurren. Formula horietan $B_1 \equiv y \,\%\, (j+1) = 0 \wedge x \,\%\, (j+1) = 0 \wedge x \neq j+1$ da. Bigarren kasuan, $\varphi_{11} \equiv (\varphi_5)_{by}^{by+j+1}$ eta $\varphi_{12} \equiv (\varphi_6)_{by}^{by+j+1}$ asertzioak lortuko dira eta $INB \wedge B \wedge \neg B_1 \wedge B_2$ eta $INB \wedge B \wedge E = z \wedge \neg B_1 \wedge B_2$ formulek bi formula horiek inplikatzen dituzte hurrenez hurren. Formula horietan $B_2 \equiv y \,\%\, (j+1) = 0$ da. Hirugarren kasuan, $\varphi_{13} \equiv (\varphi_5)_{bx}^{bx+j+1}$ eta $\varphi_{14} \equiv (\varphi_6)_{bx}^{bx+j+1}$ asertzioak lortuko dira, eta $INB \wedge B \wedge \neg B_1 \wedge \neg B_2 \wedge B_3$ eta $INB \wedge B \wedge E = z \wedge \neg B_1 \wedge \neg B_2 \wedge B_3$ formulek bi formula horiek inplikatzen dituzte hurrenez hurren. Hor $B_3 \equiv x \,\%\, (j+1) = 0 \wedge x \neq j+1$ da. Laugarren kasuan, ez dago inolako aginduren beharrik; izan ere, $INB \wedge B \wedge \neg B_1 \wedge \neg B_2 \wedge \neg B_3$ formulak eta $INB \wedge B \wedge E = z \wedge \neg B_1 \wedge \neg B_2 \wedge \neg B_3$ formulek φ_5 eta φ_6 inplikatzen dituzte hurrenez hurren.

Eratorritako programa honako hau da:

```
{ φ }
  { φ3 }
  bx := 0;
  { φ2 }
  by := 0;
  { φ1 }
  j := 0;
  while  { INB }
   (j /= y-1) and (by <= x) and (bx <= y)
  loop   { E }
    if y % (j+1) = 0 and x % (j+1) = 0 and x /= j+1 then
      { φ9 }  { φ10 }
      bx := bx+j+1;
      { φ7 }  { φ8 }
      by := by+j+1;
    elsif y  %  (j+1) = 0 then
      { φ11 }  { φ12 }
      by := by+j+1;
    elsif x % (j+1) = 0 and x /= j+1 then
      { φ13 }  { φ14 }
      bx := bx+j+1;
    // else (ez dago)
      { INB ∧ B ∧ ¬B1 ∧ ¬B2 ∧ ¬B3 → φ5 }
      { INB ∧ B ∧ E = z ∧ ¬B1 ∧ ¬B2 ∧ ¬B3 → φ6 }
    end if;
    { φ5 }  { φ6 }
    j := j+1;
  end loop;
  { φ4 }
  zlag := (by = x and bx = y);
{ ψ }
```

7.10. BIBLIOGRAFIA-OHARRAK

E. W. Dijkstra-k [34] artikuluan aurkeztu zuen lehenengo aldiz kapitulu honetan landu den eratorpen formalerako metodoa. Gainera, adibide-bilduma eder eta luzea duen [33] liburuan gehiago landutako azalpena aurkeztu zuen E. W. Dijkstrak. Bestalde, [48] da metodoaren aurkezpen xehatua, arretatsua eta didaktikoa egiten aitzindaria den liburua. Liburu horrek zailtasun-maila

desberdinetako adibideak eta ariketa-bildumak ere baditu, eta ariketa horien soluzioak eranskin batean jasota datoz. Programen eratorpen formalari buruzko liburu asko agertu ziren 80ko hamarkadaren bukaera aldera. Liburu horietan metodoa era arretatsuan lantzen da, adibide eta ariketen bildumak aberastuz. Liburu horietatik batez ere [9, 35, 36, 58] nabarmendu nahiko genituzke. 90eko hamarkadan gaztelaniazko hainbat testuk, esate baterako [13] eta [79] liburuek, eratorpen formalari zati bat eskaini zioten eta testu horiek lagungarriak izan ziren programatzeko era metodikoa zabaltzeko. Geroago, programek «eraikitzetik zuzenak» izan behar dutela dioen ikuspegian oinarrituz, programak eratortzen edo kalkulatzen espezializatutako monografiak egin dira, bai ingelesez [10] bai gaztelaniaz [67, 68].

7.11. ARIKETAK: PROGRAMA ITERATIBOEN ERATORPENA

Atal honetan iterazio edo begizta bakarreko programak formalki eratortzea eskatzen da.

1. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta b aldagai boolearrean $A(1..n)$-ko elementu guztiak bikoitiak al diren erabakiko duen programa. Programak eraginkorra izan beharko du, hau da, bikoitia ez den elementuren bat aurkitzen bada, bukaerako erantzunak zein izan behar duen garbi egongo denez, *while* aginduak bukatu egin beharko du une horretan bertan.

2. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa eta 20 baino handiagoa edo berdina den x zenbaki osoa hartu eta q aldagai boolearrean $A(1..n)$-ko elementuren bat (gutxienez elementu bat) x-ren anizkoitza al den erabakiko duen programa. Eratorritako programak eraginkorra izan beharko du, hau da, erantzuna baiezkoa izango dela detektatzen bada uneren batean, gainerako posizioak aztertu gabe bukatu beharko du programak.

3. Formalki eratorri datu bezala osoa den x zenbakia eta 20 baino txikiagoa den baliorik ez duen zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta q aldagai boolearrean x zenbakia $A(1..n)$-ko elementu guztien anizkoitza al den erabakiko duen programa. Eratorritako programak eraginkorra izan beharko du, hau da, erantzuna ezezkoa izango dela detektatzen bada uneren batean, gainerako posizioak aztertu gabe bukatu beharko du iterazioak.

4. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta $B(1..n)$ bektorean $A(1..n)$-ren kopia sortuko duen programa.

   Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{array}{rcl} \varphi & \equiv & n \geq 1 \wedge \forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = a_k \,) \\ \psi & \equiv & \forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = B(k) = a_k \,) \end{array}$$

5. Formalki eratorri zenbaki datu bezala osozko $A(1..n)$ bektore ez-hutsa hartu eta $B(1..n)$ bektorean $A(1..n)$-ko elementuak alderantzizko ordenan kopiatuko dituen programa.

   Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{array}{rcl} \varphi & \equiv & n \geq 1 \wedge \forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = a_k \,) \\ \psi & \equiv & \forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = B(n-k+1) = a_k \,) \end{array}$$

6. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta $A(1..n)$ bektorean bertan elementuak alderantzizko ordenan ipiniko dituen programa.

   Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{array}{rcl} \varphi & \equiv & n \geq 1 \wedge \forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = a_k \,) \\ \psi & \equiv & \forall k \, (\, 1 \leq k \leq n \, \rightarrow \, A(k) = a_{n-k+1} \,) \end{array}$$

7. Formalki eratorri datu bezala osoa den x zenbakia eta zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta x balioa $A(1..n)$ bektorean lehenengo aldiz zein posiziotan agertzen den i aldagaian kalkulatuko duen programa. Bektorean x ez bada agertzen, i aldagaian $n+1$ balioa itzuli beharko du programak.

8. Formalki eratorri datu bezala osoa den x zenbakia eta gutxienez x-ren agerpen bat duen zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta x balioa $A(1..n)$ bektorean lehenengo aldiz zein posiziotan agertzen den i aldagaian kalkulatuko duen programa.

9. Formalki eratorri datu bezala gutxienez bi osagai dituen zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta beheranzko ordenan eta elkarren ondoan dauden elementuez osatutako lehenengo bikotearen lehenengo osagaiaren posizioa i aldagaian kalkulatuko duen programa. Beheranzko ordenan dauden elementuez osatutako bikoterik ez badago, i aldagaian n balioa itzuli beharko du programak.

10. Formalki eratorri datu bezala bakarrik zeroak eta batekoak dituzten zenbaki osozko $A(1..n)$ eta $B(1..n)$ bektore ez-hutsak hartu eta oinarri bitarrean egindako A eta B-ren batura $S(0..n)$ bektorean eta digituen baturek sortzen dituzten bururakoak $L(1..n)$ bektorean kalkulatuko dituen programa.

    Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{array}{rcl}
\varphi & \equiv & n \geq 1 \wedge \\
 & & \forall k \, (\, 1 \leq k \leq n \, \rightarrow \, (\, (\, A(k) = 0 \, \vee \, A(k) = 1 \,) \wedge \\
 & & \qquad\qquad (\, B(k) = 0 \, \vee \, B(k) = 1 \,) \,) \,) \\
\psi & \equiv & \forall k \, (\, 0 \leq k \leq n-1 \, \rightarrow \\
 & & \qquad L(k) = (\, A(k+1) + B(k+1) + L(k+1) \,)/2 \,) \wedge \\
 & & L(n) = 0 \, \wedge \, S(0) = L(0) \, \wedge \\
 & & \forall k \, (\, 1 \leq k \leq n \, \rightarrow \\
 & & \qquad S(k) = (\, A(k) + B(k) + L(k) \,) \, \% \, 2 \,)
\end{array}$$

11. Formalki eratorri datu bezala gutxienez bi osagai dituen zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta ondoan dauden A-ko bi elementuren arteko alde handienaren balio absolutua ah aldagaian itzuliko duen programa. Gainera, *pos* aldagaian alde handieneko bikoteko lehenengo osagaiaren indizea itzuli beharko du programak. Alde handien hori bikote bati baino gehiagori badagokie, ezkerretik hasita lehenengo dagoen bikoteari dagokion posizioa itzuli beharko du.

12. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa eta x zenbaki osoa hartu eta balio absolutuan x-rekiko alde handiena duen A-ko elementuari dagokion posizioa hp aldagaian itzuliko duen programa.

    Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{array}{rcl}
\varphi & \equiv & n \geq 1 \\
\psi & \equiv & 1 \leq hp \leq n \, \wedge \, \forall k \, (\, 1 \leq k \leq n \, \rightarrow \, | \, A(k) - x \, | \leq | \, A(hp) - x \, | \,)
\end{array}$$

13. Formalki eratorri datu bezala osoa den x zenbakia eta polinomio baten koefizienteak dituen zenbaki osozko $A(0..n)$ bektore ez-hutsa hartu eta polinomioaren balioa p aldagaian itzuliko duen programa.

    Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{array}{rcl}
\varphi & \equiv & n \geq 0 \\
\psi & \equiv & p = A(0) + \sum_{k=1}^{n} A(k) * x^k
\end{array}$$

Gainera, honako inbariante honen bidez formalizatutako estrategiari jarraituz eratorri beharko da programa hori:

$$INB \equiv 0 \leq i \leq n \wedge p = A(0) + \sum_{k=1}^{i} A(k) * x^k$$

14. Formalki eratorri datu bezala zero balioa ez duen x zenbaki osoa eta polinomio baten koefizienteak dituen zenbaki osozko $A(0..n)$ bektore ez-hutsa hartu eta polinomioaren balioa p aldagaian itzuliko duen programa.

    Aurrebaldintzatzat eta postbaldintzatzat bezala honako formula hauek hartu beharko dira:

$$\begin{aligned} \varphi &\equiv x \neq 0 \wedge n \geq 0 \\ \psi &\equiv p = \sum_{k=0}^{n} A(k) * x^k \end{aligned}$$

    Gainera, honako inbariante honen bidez formalizatutako estrategiari jarraituz eratorri beharko da programa hori:

$$INB \equiv 0 \leq i \leq n \wedge p = \sum_{k=i}^{n} A(k) * x^{k-i}$$

15. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta $B(1..n)$ bektoreko k posizio bakoitzean A-ko 1 posiziotik k posiziora arteko elementuen batezbesteko aritmetikoa kalkulatuko duen programa.

    Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{aligned} \varphi &\equiv n \geq 1 \\ \psi &\equiv \forall k \, (\, 1 \leq k \leq n \rightarrow B(k) = (\sum_{j=1}^{k} A(j) \,) / k \,) \end{aligned}$$

16. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta $B(1..n)$ bektoreko k posizio bakoitzean A-ko $n-k+1$ posiziotik n posiziora arteko elementu handiena kalkulatuko duen programa.

    Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{aligned} \varphi &\equiv n \geq 1 \\ \psi &\equiv \forall k \, (\, 1 \leq k \leq n \rightarrow B(k) = handiena(A(n-k+1..n)) \,) \end{aligned}$$

17. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta *ona* aldagai boolearrean $A(1..n)$ bektorea *onargarria* den erabakiko duen programa.

    Bektore batek honako bi baldintza hauek betetzen baditu, onargarria dela esango dugu:

    - Batura negatiboa duen hasierako sekziorik ez izatea.
    - Bektoreko elementuen batura zero izatea.

    Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{aligned}\varphi &\equiv n \geq 1\\ \psi &\equiv ona \leftrightarrow (\,\forall k\,(\,1 \leq k \leq n-1 \rightarrow (\,\sum_{j=1}^{k} A(j)\,) \geq 0\,) \wedge \\ &\qquad (\,\sum_{j=1}^{n} A(j)\,) = 0\,)\end{aligned}$$

18. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa hartu eta *hp* aldagai boolearrean $A(1..n)$ bektoreko balio handienaren lehenengo agerpenaren (ezkerretik hasita) posizioa kalkulatuko duen programa.

19. Formalki eratorri datu bezala osagai kopuru desberdina izan dezaketen zenbaki osozko $A(1..n)$ eta $B(1..m)$ bektore ez-hutsak hartu eta, A-ko elementuak goranzko ordenan daudela eta B-ko elementuak beheranzko ordenan daudela jakinda, $C(1..n+m)$ bektorean A-ko eta B-ko elementuen nahasketa, elementu denak goranzko ordenan ipiniz, kalkulatuko duen programa.

    Adibidea:

$$\begin{aligned}A(1..6) &= (3,7,8,8,15,20)\\ B(1..4) &= (11,6,5,3)\\ C(1..10) &= (3,3,5,6,7,8,8,11,15,20)\end{aligned}$$

20. Formalki eratorri datu bezala zenbaki osozko $A(1..n)$ bektore ez-hutsa eta x eta y zenbaki osoak hartu eta x eta y ordena horretan bektoreko elkarren ondoko bi posiziotan agertzen diren ala ez erabakiko duen eta erantzuna c aldagai boolearrean utziko duen programa.

    Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{aligned}\varphi &\equiv n \geq 1\\ \psi &\equiv c \leftrightarrow \exists k\,(\,1 \leq k \leq n-1 \wedge A(k) = x \wedge A(k+1) = y\,)\end{aligned}$$

Hainbat estrategia kontsidera daitezkeenez, honako inbariante hauen bidez formalizatutako estrategia bakoitzeko eratorpen bat egin:

a) $INB \equiv 1 \leq i \leq n \wedge$
$\forall k \, (\, 1 \leq k \leq i-1 \rightarrow (\, A(k) \neq x \vee A(k+1) \neq y \,) \,)$

b) $INB \equiv 0 \leq i \leq n-1 \wedge$
$\forall k \, (\, 1 \leq k \leq i \rightarrow (\, A(k) \neq x \vee A(k+1) \neq y \,) \,)$

21. Formalki eratorri datu bezala positiboa den x zenbaki osoa hartu eta x zenbaki *triangeluarra* den ala ez erabakiko duen eta erantzuna *tri* aldagai boolearrean utziko duen programa.

x zenbakia zenbaki triangeluarra dela esango dugu 1etik hasitako ondoz ondoko zenbakiz eratutako segida baten baturaren berdina baldin bada.

Aurrebaldintzatzat eta postbaldintzatzat honako formula hauek hartu beharko dira:

$$\begin{aligned} \varphi &\equiv x \geq 1 \\ \psi &\equiv tri \leftrightarrow \exists k \, (\, k \geq 1 \wedge x = \sum_{j=1}^{k} j \,) \end{aligned}$$

Honako inbariante honen bidez formalizatutako estrategiari jarraituz eratorri beharko da programa hori:

$$INB \equiv i \geq 0 \wedge batura = \sum_{j=1}^{i} j \wedge batura < x$$

8. Programa errekurtsiboen eratorpena

3. kapituluan azaldutako egiaztapen formalerako teknikan oinarritzen den programa iteratiboak eratortzeko metodo bat aurkeztu da 7. kapituluan. Era berdintsuan, 4. kapituluan azaldutako programa errekurtsiboen egiaztapen formalerako teknika oinarri bezala duen programa errekurtsiboak eratortzeko metodo bat aurkeztuko dugu kapitulu honetan. Beraz, metodo honen ardatz nagusiak Hoareren kalkulua eta indukzio-hipotesia izango dira. Lehenengo urratsa programa espezifikatzea izango da, eta horretarako parametroak zehaztu beharko dira. Kasu batzuetan, problemaren berezko parametrizazioak ez du ahalbidetzen soluzioaren definizio errekurtsiborik. Kasu horietan hasierako funtzioa (programa) parametro gehiago dituen beste funtzio batera orokor daiteke. Teknika hori murgilketa bezala ezagutzen da, eta eraginkortasuna hobetzeko ere erabil daiteke.

8.1. atalean, egiaztapenerako erabiltzen diren tekniketan oinarritzen den programa errekurtsiboen eratorpen formalerako metodoa aurkeztuko da, baita haren aplikazioaren bi adibide ere. 8.2. atalean, adibideekin ere, definizio errekurtsiboak errazten dituen murgilketaren teknika aurkeztuko da. 8.3. atalean, aurreko ataletan azaldutako adibideen antzekoak diren ariketa batzuk proposatuko dira eratorpen formala eta murgilketa lantzeko. 8.4. atalean, murgilketaren teknika erabiliz eraginkortasuna hobetzen deneko hiru adibide emango dira. 8.6. atalean, kapituluan zehar landutakoa jasotzen duten ariketa batzuk proposatzen dira. Bukatzeko, datu historiko batzuk eta erreferentzia aipagarrienak biltzen dira 8.5. atalean.

8.1. PROGRAMA ERREKURTSIBOEN ERATORPEN FORMALA

Atal honetan, programa errekurtsiboak eratortzeko metodo orokorra eta metodo hori nola aplikatzen den erakusten duten bi adibide aurkeztuko ditugu. Adibide horiek bi azpiataletan joango dira. Iterazioen eratorpen formalean bezala, `f` funtzio errekurtsibo bat formalki eratorri edo diseinatzeko, problemaren espezifikazio formaletik abiatu beharko da (φ aurrebaldintza eta ψ

postbaldintzatik, beraz):

```
function f(x̄ : T̄) return z:  R is
{ φ }
begin
  // Gorputza
end f;
{ ψ }
```

Problema bat errekurtsiboki ebazteak problema horren izaera bereko problema txikiagoen soluzioak erabiltzea eskatzen duenez, lehenengo urratsa (ez-formala oraindik) problemaren datuak txikiagoak edo errazagoak diren datuetan nola deskonposa daitezkeen aztertzea izango da, betiere hasierako problemaren soluzioa txikiagoak edo errazagoak diren datu horiei dagozkien soluzioak erabiliz kalkulatu beharko dela kontuan hartuz. Estrategia edo ideia orokorra planteatu ondoren, gerta daitezkeen kasu guztien erabateko azterketa egin beharko da, oinarrizko kasuak edo kasu nabariak (errekurtsibitaterik edo problema txikiagotan deskonposatzearen beharrik ez dutenak) eta kasu errekurtsiboak edo induktiboak zein diren zehaztuz. Kasuak baldintzen bidez zehaztu beharko dira, gerta daitezkeen kasu denak estaliz, hau da, kasu nabariei dagozkien baldintzak $B_{N_1}, B_{N_2}, \ldots, B_{N_\ell}$ baldin badira eta kasu induktiboei dagozkien baldintzak $B_{I_1}, B_{I_2}, \ldots, B_{I_m}$ baldin badira, honako inplikazio hau bete beharko da:

$$\varphi \rightarrow B_{N_1} \vee B_{N_2} \vee \ldots \vee B_{N_\ell} \vee B_{I_1} \vee B_{I_2} \vee \ldots \vee B_{I_m}$$

B_{N_j} oinarrizko kasu bakoitzarentzat honako hirukote hau zuzena izatea ahalbidetzen duen P_j programa eratorri beharko da:

$$\{ \varphi \wedge B_{N_j} \}\ \mathtt{P_j}\ \{ \psi \}$$

Kasu errekurtsiboei dagokienez, hasteko, B_{I_j} kasu bakoitzari dagokion dei errekurtsiboko $\bar{t}_j$ argumentuek aurrebaldintzaren arabera $\overline{x}$ sarrerako datuek bete beharko luketena betetzen duten ala ez egiaztatu beharko da, hau da, $\varphi \wedge B_{I_j} \rightarrow \varphi^{\bar{t}_j}_{\overline{x}}$. Gero, honako egitura hau izango duen indukzio-hipotesia planteatu beharko da kasu errekurtsibo bakoitzerako:

$$\{ \varphi^{\bar{t}_j}_{\overline{x}} \}\ \ \mathtt{w := f(\bar{t}_j);}\ \{ \psi^{\bar{t}_j, w}_{\overline{x}, z} \}$$

Hor ikusten den bezala, parametroak eta emaitza formalak dei konkretu bakoitzari dagozkion argumentuekin eta emaitzarekin ordezkatu behar dira, hurrenez hurren.

Gainera, indukzioa balidatzeko erabili beharko den E borne-adierazpena ere planteatu beharko da. Borne-adierazpen horrek datuak gero eta txikiagoak izango direla eta dei errekurtsiboen sekuentzia finitua izango dela erakusteko balio beharko du. Beraz, 4. kapituluan azaldu den bezala, B_{N_j} oinarrizko kasu bakoitzeko honako hau betetzen dela egiaztatu behar da:

$$(\varphi \wedge B_{N_j}) \rightarrow E \in \mathbb{N}$$

Eta B_{I_j} kasu errekurtsibo bakoitzeko honako hau betetzen dela egiaztatu behar da:

$$(\varphi \wedge B_{I_j}) \rightarrow (E^{\bar{t}_j}_{\bar{x}} \in \mathbb{N} \wedge E > E^{\bar{t}_j}_{\bar{x}})$$

Bestalde, kasu errekurtsibo bakoitzean dei errekurtsiboaren bidez lortutako w emaitzatik bukaerako z emaitza lortzeko beharrezkoak diren Q_j aginduak ere eratorri beharko dira, beti ere honako hirukote hau betearazteko helburuarekin:

$$\{ \varphi \wedge B_{I_j} \} \quad \texttt{w := f(}\bar{\texttt{t}}_\texttt{j}\texttt{); Q}_\texttt{j} \{ \psi \}$$

Urrats horri *konposaketa-urrats* deitzen zaio.

Laburbilduz, algoritmo errekurtsibo baten diseinuan honako fase hauek bereiz daitezke:

- Espezifikazioa: funtzioaren izena, parametroen eta emaitzaren izenak eta motak, aurrebaldintza eta postbaldintza.
- Kasu-azterketa: baldintzen definizioa eta espezifikazioaren arabera onargarriak diren kasu guztiak estaltzen direla egiaztatzea.
- Kasu nabarien diseinua: kasu nabari bakoitzari dagokion espezifikazioa zehaztu, eratorpena egin eta zuzentasuna bermatzea.
- Kasu errekurtsiboen diseinua: kasu errekurtsibo bakoitzari dagokion indukzio-hipotesia planteatu eta konposaketa eratortzea.
- Indukzioaren balidazioa: borne-adierazpena definitu eta kasu bakoitzean kasuari dagokion inplikazioa egiaztatzea.

Fase horien formalizazioa hirukote eta inplikazio logiko bezala adierazi da lehenago.

Jarraian datozen bi azpiataletan, metodo honen aplikazioaren bi adibide aurkeztuko ditugu.

8.1.1. Adibidea: Zatidura osoa

Osoak diren x eta y zenbakien arteko zatiketa osoa kalkulatuko duen *zat* funtzioaren eratorpen-prozesua erakutsiko da jarraian. Honako aurre-ondoetako espezifikazio hau hartuko da abiapuntutzat:

```
function zat(x,y:  Integer) return e:  Integer is
{ φ ≡ x ≥ 0 ∧ y ≥ 1 }
begin
  // Gorputza
end zat;
{ ψ ≡ e = x/y }
```

x eta y-ren arteko zatiketa osoa x objektu edo elementu y talde desberdinen artean banatzearen problema bezala plantea daiteke. Talde bakoitzari elementu kopuru bera eman beharko zaio eta y baino txikiagoa den elementu kopuru bat banatu gabe geldi daiteke. Soluzioa talde bakoitzari elementu bana emanez eta gelditzen diren elementuak ($x-y$ elementu) y taldeen artean banatuz lor daiteke. Beraz, soluzio errekurtsibo baten aurrean gaude. $x < y$ betetzen denean oinarrizko kasuan gaude, eta ez da objekturik banatuko taldeen artean. Beraz, soluzioa 0 izango da. Formalki

$$\{ x \geq 0 \wedge y \geq 1 \wedge x < y \}\ \texttt{e := 0;}\ \{ e = x/y; \}$$

Aldiz, $x \geq y$ betetzen bada, talde bakoitzari elementu bat eman diezaiokegu, eta gero $x-y$ objektu y taldeen artean banatzearen problema ebatzi. Garbi dago kasu biak bata bestearen osagarriak direla eta, ondorioz, erabateko deskonposaketa definitzen dutela. Kasu errekurtsiboan indukzio-hipotesia honako hau izango da:

$$\text{(IH)}\ \{ x-y \geq 0 \wedge y \geq 1 \}\ \texttt{e := zat(x-y,y);}\ \{ e = (x-y)/y; \}$$

`e := zat(x-y,y);` deiak $x-y$ elementu y talde desberdinen artean banatzearen emaitza uzten duenez e aldagaian, $\psi \equiv e = x/y$ formularen bidez espezifikatutako helburua lortzeko, talde bakoitzari beste elementu bat ematea gelditzen da. Beraz, `e := e+1;` esleipena erantsi beharko da. Borneadierazpen bezala $E \equiv x$ erabiliz indukzioa balidatzea oso erraza da. Lortutako funtzioa honako hau da:

```
function zat(x,y:  Integer) return e:  Integer is
{ x ≥ 0 ∧ y ≥ 1 }
begin
  if x < y then
    e := 0;
  else
    e := zat(x-y,y);
    e := e+1;
  end if;
end zat;
{ e = x/y }
```

8.1.2. Adibidea: Zatitzaile komunetako handiena

Positiboak diren bi zenbaki osoren zatitzaile komunetako handiena kalkulatuko duen funtzio errekurtsiboa eratorriko dugu orain honako espezifikazio honetatik abiatuz:

```
function zkh(x,y:  Integer) return h:  Integer is
{ φ ≡ x ≥ 1 ∧ y ≥ 1 }
begin
  // Gorputza
end zkh;
{ ψ ≡ zkh(x,y,h) }
```

Postbaldintzan agertzen den zkh predikatuak hirugarren argumentua lehenengo bi argumentuen zatitzaile komunetako handiena dela adierazten du, eta haren definizioa honako hau da:

$$\begin{aligned} zkh(a,b,c) \quad \equiv \quad & 1 \le c \le a \ \wedge\ 1 \le c \le b \ \wedge\ a\%c = 0 \ \wedge\ b\%c = 0 \ \wedge \\ & \forall k \,(\, (\, c+1 \le k \le a \ \wedge\ c+1 \le k \le b \,) \ \rightarrow \\ & \qquad (\, a\%c \neq 0 \ \vee\ b\%c \neq 0 \,) \,) \end{aligned}$$

Beraz, espezifikazioaren arabera, x eta y-ren zatitzaile komunetako handiena h aldagaian kalkulatuko da. Aurreko adibideko planteamenduari jarraituz, x eta y-ren zatitzaile komunetako handiena x zein y objektuak gehienez zenbat h talderen artean bana daitezkeen kalkulatzea da, objekturik sobratu gabe. x edo y-ren balioa 1 baldin bada, talde bakar baterako egin ahal izango da banaketa (hau da, h-ren balioa 1 izango litzateke). x eta y berdinak baldin badira, x (edo y) talderi x eta y objektu dituzten multzo bakoitzetik bat eman dakieke, eta, ondorioz, $h = x = y$. Bestalde, $x > y > 1$ betetzen bada, h balioak y-ren zatitzailea izan beharko du (eta y elementu h talderen artean

banatu beharko dira elementurik sobratu gabe). $z \geq 1$ betetzen duen z balioren batentzat $x = y + z$ betetzen denez, z balio horrek, hau da, $x - y$ balioak h balioaz zatigarria izan beharko du. Beraz, bilatu beharreko h balioak y eta $x - y$ balioen zatitzaile komunetako handiena izan beharko du. Eta hori da problema berria. $y > x > 1$ kasuan ere arrazoibide berari jarraituko litzaioke, baina kasu honetan x eta $y - x$ balioen zatitzaile komunetako handiena kalkulatu beharko da. Beraz, datuak zenbaki oso positiboak izango direla bermatzen duen aurrebaldintzatik abiatuz, lau kasu ditugu: bi kasu nabari ($x = 1$ *or* $y = 1$ eta $x = y > 1$) eta bi errekurtsibo ($x > y > 1$ eta $y > x > 1$). Lau kasu horiek aurrebaldintzak baimentzen dituen egoeren erabateko deskonposaketa osatzen dute. Zuzenak diren honako hirukoteak dagozkie kasu nabariei:

$$\{\, x = 1 \wedge y = 1 \,\}\ \texttt{h := 1;}\ \{\, zkh(x,y,h) \,\}$$
$$\{\, x = y > 1 \,\}\ \texttt{h:= x;}\ \{\, zkh(x,y,h) \,\}$$

Bestalde, kasu errekurtsibo bakoitzean indukzio-hipotesi bat beharko da:

$$\text{(IH1)}\ \{\, x - y \geq 1 \wedge y \geq 1 \,\}\ \texttt{h := zkh(x-y,y);}\ \{\, zkh(x-y,y,h) \,\}$$
$$\text{(IH2)}\ \{\, x \geq 1 \wedge y - x \geq 1 \,\}\ \texttt{h := zkh(x,y-x);}\ \{\, zkh(x,y-x,h) \,\}$$

Kasu errekurtsiboetan beste agindurik ez dela beharko ikustea erraza da; izan ere, bai $zkh(x-y,y,h)$ espresioak bai $zkh(x,x-y,h)$ espresioak $zkh(x,y,h)$ postbaldintza bermatzen dute agindu gehiagoren beharrik gabe. Gainera, garbi dago $E \equiv x + y$ espresioak indukzioa balidatzea ahalbidetzen duela. Eratorritako funtzioa honako hau da:

```
function zkh(x,y:  Integer) return h:  Integer is
{ x ≥ 1 ∧ y ≥ 1 }
begin
  if x= 1 or y = 1 then
    h := 1;
  elsif x = y then
    h := x;
  elsif x > y then
    h := zkh(x-y,y);
  else
    h := zkh(x,y-x);
  end if;
end zkh;
{ zkh(x,y,h) }
```

8.2. MURGILKETAREN TEKNIKA

Aurreko atalean azaldutako metodoaren lehenengo urratsean (espezifikazioan) problemaren berezko parametrizazioak soluzio errekurtsibo bat formulatzea ahalbidetzen ez duenean aplikatu ohi den metodo bat aurkeztuko dugu atal honetan. Arazo hori duen adibide erraz bat x zenbaki oso positiboaren zatitzaileen baturaren kalkulua da:

```
function zat_batu(x: Integer) return zb: Integer is
{ x ≥ 1 }
begin
  //
end zat_batu;
{ zb = Σ_{1≤k≤x ∧ x%k=0} k }
```

x zenbakiaren zatitzaileen batura ezin da (nolabait x baino txikiagoa den) z zenbaki baten zatitzaileen batura erabiliz adierazi. Oro har, x datu bezala duen problema bat emanda, x baino sinpleagoa den daturen batentzat ($x-1$, $x/2$, eta abar) problema beraren soluzioa kalkulatu, eta soluzio hori erabiliz adierazi ahal izango da x datuarentzako soluzioa. Baina x zenbaki bat emanda, haren zatitzaileen batura $x-1$ edo $x/2$ edo antzekoa den balioren baten zatitzaileak erabiliz adieraztea ezinezkoa dirudi. Problema hobeto aztertzen badugu, $1..x$ tartekoak diren x zenbakiaren zatitzaileen batura kalkulatu nahi dela ohartuko gara. Beraz, $1..x$ tartea ezkutuan gelditu da problemaren hasierako formulazioan. Kasu-azterketa egiten badugu, $1..x$ tartekoak diren x-ren zatitzaileen batura 1 gehi $2..x$ tartekoak diren x-ren zatitzaileen batura dela ikusiko dugu. Beraz, soluzio errekurtsiboa $1..x$ tartearekiko plantea daiteke, x-rekiko planteatu beharrean. Oro har, $y..x$ tartekoak diren x zenbakiaren zatitzaileen batura y (y balioa x-ren zatitzailea baldin bada) edo 0 (y balioa x-ren zatitzailea ez bada) gehi $y+1..x$ tartekoak diren x zenbakiaren zatitzaileen batura izango da. Kasu berezi bezala $y > x$ betetzen bada, batura 0 izango da, $y..x$ hutsa izateagatik.

Oro har, murgilketaren teknika algoritmo errekurtsiboen parametrizazioa berriz planteatzean datza (parametro berriak gehituz, gehienetan) errekurtsibitatea parametro horien bidez formulatu ahal izateko. Zatitzaileen baturaren kasuan, x-ren zatitzaileen bilaketa $y..x$ tartean egin behar dela adieraziko duen y parametroa gehituko dugu. Beraz, y baino handiagoak edo berdinak diren x-ren zatitzaileen batura kalkulatzea eskatzen duen problema orokortuaren aurrean gaude:

```
function zat_batu_mr(x,y:  Integer) return zb:  Integer is
```
$\{\, x \geq 1 \wedge y \geq 1 \,\}$
```
begin
  //
end zat_batu_mr;
```
$\{\, zb = \sum_{\substack{y \leq k \leq x \,\wedge \\ x\%k=0}} k \,\}$

zat_batu_mr funtzio berri hau *funtzio murgiltzailea* bezala ezagutzen da. Funtzio errekurtsibo hori aurreko ataleko metodoa erabiliz diseinatuko dugu. Hasierako funtzioa funtzio murgiltzaileari dei jakin bat eginez definituko da:

```
function zat_batu(x:  Integer) return zb:  Integer is
```
$\{\, x \geq 1 \,\}$
```
begin
  zb := zat_batu_mr(x,1);
end zat_batu;
```
$\{\, zb = \sum_{\substack{1 \leq k \leq x \,\wedge \\ x\%k=0}} k \,\}$

Lehen egin dugun arrazoibidearen arabera, *zat_batu_mr* funtzioaren kasu nabaria $x < y$ betetzen denean gertatzen da, eta kasu horretan garbi dago zb emaitzari asignatu beharreko balioa 0 dela. Kasu errekurtsiboa $x >= y$ da, eta kasu horri dagokion indukzio-hipotesia honako hau da:

(IH) $\{\, x \geq 1 \wedge y+1 \geq 1 \,\}$
```
  zb := zat_batu_mr(x,y+1);
```
$\{\, zb = \sum_{\substack{y+1 \leq k \leq x \,\wedge \\ x\%k=0}} k \,\}$

Beraz, jarraian erakusten den hirukotea zuzena izatea eragiten duen Q programa diseinatu beharko da konposaketa-urratsean:

$$\{\, zb = \sum_{\substack{y+1 \leq k \leq x \,\wedge \\ x\%k=0}} k \,\}\ \texttt{Q}\ \{\, zb = \sum_{\substack{y \leq k \leq x \,\wedge \\ x\%k=0}} k \,\}$$

Ondorioz, `Q` $\equiv$ `if x % y = 0 then  zb := y+zb; end if;` izango da. Indukzioa balidatzeko erabil dezakegun E espresioa $x - y$ izango da.

Guztira honako funtzio hau lortuko da:

```
function zat_batu_mr(x,y:  Integer) return zb:  Integer is
```
$\{\, x \geq 1 \wedge y \geq 1 \,\}$
```
begin
  if x<y then
    zb := 0;
  else
    zb := zat_batu_mr(x,y+1);
    if x % y = 0 then  zb := y+zb; end if;
  end if;
end zat_batu_mr;
```
$\{\, zb = \sum_{\substack{y \leq k \leq x \,\wedge \\ x\%k=0}} k \,\}$

Positiboa den x zenbaki oso bat lehena den ala ez erabakiko duen funtzio errekurtsiboa diseinatzeko egin beharrekoa aurreko adibidean egindakoaren antzekoa da. Kasu honetan, $y..x-1$ tartean x zenbakiaren zatitzailerik ba al dagoen erabaki beharko du funtzio murgiltzaileak. Diseinatu nahi den *lehena* izeneko funtzioak $2..x-1$ tarteari dagozkion parametro egokiekin deitu beharko dio funtzio murgiltzaileari. Beraz, y baino handiagoa edo berdina den x-ren berezko zatitzailerik ba al dagoen erabakiko duen funtzio murgiltzaileari *han_ber_zat* deitzen badiogu, haren espezifikazioa honako hau izango da:

```
function han_ber_zat(x,y:  Integer) return h:  Boolean is
```
$\{\, x \geq 1 \wedge y \geq 2 \,\}$
```
begin
⋮
end han_ber_zat;
```
$\{\, h \leftrightarrow 0 = \sum_{\substack{y \leq k \leq x-1 \,\wedge \\ x\%k=0}} k \,\}$

Hasieratik diseinatu nahi zen *lehena* funtzioa funtzio murgiltzaileari dei jakin bat eginez definituko da:

```
function lehena(x:  Integer) return e:  Boolean is
```
$\{\, x \geq 1 \,\}$
```
begin
  e := han_ber_zat(x,2);
end lehena;
```
$\{\, e \leftrightarrow 0 = \sum_{\substack{2 \leq k \leq x-1 \,\wedge \\ x\%k=0}} k \,\}$

han_ber_zat funtzioaren diseinua bukatzeko gelditzen dena ariketa bezala utziko da irakurlearentzat (ikus 8.3. atala).

8.3. ARIKETAK: ERATORPENA ETA MURGILKETA

1. Aplikatu programa errekurtsiboen eratorpen formalaren metodoa honako funtzio honen diseinua osatzeko:

   ```
   function han_ber_zat(x,y:  Integer)
   return h:  Boolean is
   ```
   $\{\, x \geq 1 \,\wedge\, y \geq 2 \,\}$
   ```
   begin
   ⋮
   end han_ber_zat;
   ```
   $\{\, h \;\leftrightarrow\; 0 = \sum_{\substack{y \leq k \leq x-1 \,\wedge \\ x\%k=0}} k \,\}$

2. Positiboa den x zenbaki oso bat *betea* dela esaten da, x-ren berezko zatitzaile positiboen batura x baldin bada. Gogoratu x-ren berezko zatitzaile positiboak $[1..x-1]$ tarteko x-ren zatitzaileak direla. Adibidez, 6 zenbakia betea da bere berezko zatitzaileak $1, 2$ eta 3 baitira eta $6 = 1+2+3$ betetzen baita.

   Definitu hiru parametro izango dituen *betea_mr* funtzio murgiltzailea honako murgilketa honek x betea den ala ez erabakitzearen problema ebatzi beharko duela kontuan hartuz:

   ```
   function betea(x:  Integer) return e:  Boolean is
   ```
   $\{\, x \geq 1 \,\}$
   ```
   begin
     e := betea_mr(x,1,0);
   end betea;
   ```
   $\{\, e \leftrightarrow x = \sum_{\substack{1 \leq k \leq x-1 \,\wedge \\ x\%k=0}} k \,\}$

   Funtzio murgiltzailea honako espezifikazio honetatik abiatuta diseinatu beharko da:

   ```
   function betea_mr(x,y,b:  Integer)
   return e:  Boolean is
   ```
   $\{\, x \geq 1 \,\wedge\, y \geq 1 \,\}$
   ```
   begin
   ⋮
   end betea_mr;
   ```
   $\{\, e \;\leftrightarrow\; x = b + \sum_{\substack{y \leq k \leq x-1 \,\wedge \\ x\%k=0}} k \,\}$

8.4. MURGILKETA ERAGINKORTASUNA LORTZEKO

8.2. atalean azaldutako murgilketaren teknika eraginkorragoak diren funtzio errekurtsiboak diseinatzeko nola erabil daitekeen erakutsiko dugu atal honetan. Zuzeneko soluzio errekurtsiboa onartzen duten funtzio batzuk eraginkorrak ez izatea gerta daiteke. Kasu horietan, berez murgilketaren teknikaren beharrik ez egon arren, murgilketa erabiliz eraginkorragoa den soluzioa lor daiteke. Eraginkortasun ezaren arrazoiak bi izan daitezke: alde batetik, deien sekuentzia (edo zuhaitza) bi aldiz zeharkatu behar izatea, beherantz dei errekurtsiboak egitean, eta gero gorantz maila bakoitzean sortutako informazio partziala erabiliz behin betiko emaitza osatzeko; eta, beste aldetik, kalkulu-errepikapena. Deien sekuentzia edo zuhaitza bi aldiz zeharkatzea eta kalkuluen errepikapena saihesten duten programak diseinatzeko erabil daiteke murgilketaren teknika. Jarraian datozen bi azpiataletan eraginkortasun eza eragiten duten bi arazo horien adibideak —adibide bat lehenengo kasurako eta bi adibide bigarren kasurako— erakutsiko ditugu eta eraginkorragoak diren funtzioak lortzeko murgilketa nola erabili ere azalduko dugu.

Liburu honetan murgilketaren teknika erabiliz eraginkorragoa den funtzioa lortzea programa errekurtsiboen diseinuaren (edo eratorpen formalaren) ikuspuntutik landuko dugu. Hala ere, batzuetan programa errekurtsiboen transformazio-metodo bezala ere aurkeztu ohi da (ikus, adibidez, [13, 79, 67, 68]).

8.4.1. Bukaerako errekurtsibitatea deien zuhaitzak bi aldiz ez zeharkatzeko

Honako funtzio honek ohiko definizio errekurtsiboa erabiliz kalkulatzen du n-ren faktoriala:

```
function faktoriala(n:  Integer) return f:  Integer is
{ n ≥ 0 }
begin
  if n = 0 then
    f := 1;
  else
    f := faktoriala(n-1);
    f := n*f;
  end if;
end faktoriala;
{ f = n! }
```

4. kapituluko 4.1. irudian, beherantz sortutako dei errekurtsiboen sekuentzia

eta bukaerako emaitza kalkulatu ahal izateko gorantz egin beharreko bidea erakusten dira. Horrek eraginkortasunean galera dakar, bai denbora aldetik bai espazio aldetik. Goranzko bidea egin beharrik ez izateko, beheranzko bidea egitean bukaerako emaitza lortzeko beharrezkoak diren balioen biderkadura gordez joango den parametro bat ipin daiteke. Oinarrizko kasura iritsitakoan, parametro berri horretan egongo da emaitza. Parametro berri hori ipintzeak problema orokortu egiten du eta funtzio berriak faktoriala bider parametro berri horren hasierako balioa kalkulatuko du. Beraz, honako espezifikazio honetatik abiatuz diseinatuko dugu funtzio murgiltzailea:

```
function faktoriala_mr(n,b:  Integer) return f:  Integer is
{ n ≥ 0 }
begin
⋮
end faktoriala_mr;
{ f = n! * b }
```

Faktoriala kalkulatzeko, b parametroaren ordez 1 balioa ipiniz deitu beharko zaio funtzio murgiltzaileari:

```
function faktoriala(n:  Integer) return f:  Integer is
{ n ≥ 0 }
begin
  f := faktoriala_mr(n,1);
end faktoriala;
{ f = n! }
```

faktoriala_mr funtzioaren eratorpenak honako hau utziko du:

```
function faktoriala_mr(n,b:  Integer) return f:  Integer is
{ n ≥ 0 }
begin
  if n = 0 then
    f := b;
  else
    f := faktoriala_mr(n-1,n*b);
  end if;
end faktoriala_mr;
{ f = n! * b }
```

Murgilketa emaitza partzialak gordetzeko balio duten parametroak (bat edo gehiago) ipintzeko erabiltzen den kasu hauetan, *bukaerako errekurtsibitate* deritzona duten funtzioak diseinatu ohi dira gehienetan. Funtzio batek

bukaerako errekurtsibitatea duela esan ohi da dei errekurtsiboa burutu ondoren beste kalkulurik egin beharrik ez dagoenean. Adibidez, 8.1.2. azpiataleko `zkh` funtzioak bukaerako errekurtsibitatea du, aldiz, 8.1.1. azpiataleko `zat` funtzioak ez du bukaerako errekurtsibitaterik, dei errekurtsiboaren emaitza lortu ondoren batekoa batu behar baitu.

8.4.2. Murgilketa kalkuluen errepikapena saihesteko

Programa errekurtsiboen eraginkortasun ezaren iturrietako bat kalkuluen errepikapena izan ohi da askotan. Kalkulu-errepikapen hori emaitza partzial batzuk hutsetik hasi eta behin eta berriz kalkulatzeak eragiten du. Arazo hau oso ondo erakusten duen adibide ezagun bat Fibonacciren segidako n-garren terminoa kalkulatzen duen funtzioa da:

$$fibonacci(n) = \begin{cases} n & \text{baldin } 0 \leq n \leq 1 \\ fibonacci(n-1) + fibonacci(n-2) & \text{baldin } n \geq 2 \end{cases}$$

Definizio horri dagokion funtzioaren diseinua berehalakoa da:

```
function fib(n:  Integer) return f:  Integer is
  x,y:  Integer;
{ n ≥ 0 }
begin
  if n <= 1 then
    f := n;
  else
    x := fib(n-1);
    y := fib(n-2);
    f := x+y;
  end if;
end fib;
{ f = fibonacci(n) }
```

Fibonacciren segidako n-garren terminoa kalkulatzeko, Fibonacciren segidako $(n-1)$-garren eta $(n-2)$-garren terminoak errekurtsiboki eta bakoitza bere aldetik kalkulatzen dira. Horrek sortzen duen dei errekurtsiboen zuhaitzean erraz ikus daiteke kalkuluak askotan errepikatzen direla. Esate baterako, $n = 6$ denean, $fibonacci(3)$ balioa 3 aldiz kalkulatzen da. Kasu honetan, murgilketa egiterakoan helburua $fib(n-1)$ eta $fib(n)$ biak batera kalkulatzea izango da. Beraz, funtzio murgiltzaileak bi emaitza sortuko ditu. Aurreko ataleko adibideetan, funtzio murgiltzailea definitzeko, hasierako funtzioari parametroak (hau da, datuak) gehitzen zitzaizkion. Gauzak errazteko, funtzio murgiltzaileak zenbaki osoz eratutako bikote bat itzuliko duela

kontsideratuko dugu, eta bikoteen gaineko esleipena erabiliko dugu (esleipen hori aldibereko esleipenaren baliokidea izango da). Lehendabizi, funtzio murgiltzailearen espezifikazioa honako hau baldin bada,

```
function fib_mr(n:  Integer)
return (f,g):  (Integer,Integer) is
{ n ≥ 1 }
begin
⋮
end fib_mr;
{ f = fibonacci(n) ∧ g = fibonacci(n−1) }
```

orduan hasierako funtzioa honako murgilketa honen bidez kalkulatuko da:

```
function fib(n:  Integer) return f:  Integer is
  y:  Integer;
{ n ≥ 0 }
begin
  if n = 0 then
    f := 0;
  else
    (f,y) := fib_mr(n);
  end if;
end fib;
{ f = fibonacci(n) }
```

Funtzio murgiltzailea 8.1. atalean azaldutako metodoa erabiliz formalki erator daiteke, honako hau lortuz:

```
function fib_mr(n:  Integer)
return (f,g):  (Integer,Integer) is
  x,y:  Integer;
{ n ≥ 1 }
begin
  if n = 1 then
    (f,g) := (1,0);
  else
  (x,y) := fib_mr(n-1);
  (f,g) := (x+y,x);
  end if;
end fib_mr;
{ f = fibonacci(n) ∧ g = fibonacci(n−1) }
```

Kasu nabariaren zuzentasuna hirukote honi lotuta dago:

$\{\ n = 1\ \}$
`(f,g) := (1,0);`
$\{\ f = fibonacci(n) \wedge g = fibonacci(n-1)\ \}$

Eta hirukote hori bete egiten da. Kasu errekurtsiboari dagokion indukzio-hipotesia honako hau da:

$\{\ n-1 \geq 1\ \}$
`(x,y) := fib_mr(n-1);`
$\{\ x = fibonacci(n-1) \wedge y = fibonacci(n-2)\ \}$

Hirukote hori `(f,g) := (x+y,x);` esleipenarekin konposatuz lortzen den agindu-segidak $\{\ f = fibonacci(n) \wedge g = fibonacci(n-1)\ \}$ postbaldintza betearazteko balio du, dei errekurtsiboaren ondoren $\{\ x+y = fibonacci(n) \wedge x = fibonacci(n-1)\ \}$ beteko baita. Indukzioa balidatzeko $E \equiv n$ espresioa erabil daiteke.

Jarraian azalduko dugun adibidean ere murgilketaren teknika erabiliko da kalkulu-errepikapena saihesteko. Zuhaitz bitar bateko adar bakoitzeko adabegi kopuruaren eta beste edozein adarretako adabegi kopuruaren arteko aldea gehienez batekoa baldin bada, zuhaitz bitar hori *orekatua* dela esango dugu. Beste era batera esanda, zuhaitz horretako adabegi bakoitzarentzat, haren ezkerreko eta eskuineko azpizuhaitzen sakoneren arteko aldea bat da gehienez.

Zuhaitz bitar bat *orekatua* den ala ez erabakitzen duen eragiketaren espezifikazio ekuazionala honela formula daiteke:

Mota: $zuhbit(oso)$
Lag: bool
Eragiketak:
$orekatua$: $zuhbit(oso) \to bool$
Ekuazioak:
(1) $orekatua(hutsa) = \mathsf{T}$
(2) $orekatua(errotu(e, ezk, esk)) =$
$(\ abs(sakonera(ezk) - sakonera(esk)) \leq 1\ ) \wedge$
$orekatua(ezk) \wedge orekatua(esk)$

Hor *abs* funtzioak zenbaki oso baten balio absolutua kalkulatzen du eta *sakonera* funtzioak zuhaitz bitar baten sakonera kalkulatzen du.

Demagun *abs* eta *sakonera* funtzioak inplementatuta daudela. Espezifikazio hori zuzenean baliatzen duen funtzio errekurtsibo bat diseinatzea erraza da:

```
function orek_da(a:  zuhbit(Integer))
return ore:  Boolean is
  sak_ezk,sak_esk:  Integer;
  ore_ezk,ore_esk:  Boolean;
begin
  if hutsa_da(a) then
    ore := true;
  else
    sak_ezk := sakonera(ezker(a));
    sak_esk := sakonera(eskuin(a));
    ore_ezk := orek_da(ezker(a));
    ore_esk := orek_da(eskuin(a));
    ore := (abs(sak_ezk - sak_esk) <= 1) and
                ore_ezk and ore_esk;
  end if;
end orek_da;
```

$\{\, ore \leftrightarrow orekatua(a) \,\}$

orekatua eragiketaren inplementazio hori ez da eraginkorra, a-ren hainbat azpizuhaitzen sakoneraren kalkulua errepikatu egiten baitu. Adabegi bat zenbat eta sakonago egon, orduan eta gehiagotan kalkulatuko da haren eskuineko eta ezkerreko azpizuhaitzen sakonera. Murgilketa aplikatzerakoan, zuhaitz baten sakoneraren kalkulua eta zuhaitz hori orekatua den ala ez erabakitzea batera eramango dira, eta horrela azpizuhaitz bakoitza behin bakarrik zeharkatuko da. Funtzio murgiltzaileak bi emaitza itzultzea eskatzen du horrek. Aurreko adibidean bezala, funtzio murgiltzaileak emaitza bezala bikote bat itzuliko duela kontsideratuko dugu (kasu honetan boolear batez eta zenbaki oso batez osatua) eta bikoteen gaineko esleipena (aldibereko esleipenaren baliokidea) erabiliko dugu. Lehendabizi, funtzio murgiltzailearen espezifikazioa honako hau baldin bada,

```
function orek_da_mr(a:  zuhbit(T))
return (ore,sak):  (Boolean,Integer) is
```

$\{\, \neg hutsa\_da(a) \,\}$

```
begin
⋮
end orek_da_mr;
```

$\{\, ore \leftrightarrow orekatua(a) \wedge sak = sakonera(a) \,\}$

orduan honako murgilketa honen bidez kalkulatuko da hasierako funtzioa:

```
function orek_da(a:  zuhbit(T)) return ore:  Boolean is
  sak:  Integer;
begin
  if hutsa_da(a) then
    ore := true;
  else
    (ore,sak) := orek_da_mr(a);
  end if;
end orek_da;
```
$\{\, ore \leftrightarrow orekatua(a) \,\}$

8.1. atalean azaldutako metodoari jarraituz erator daiteke funtzio murgiltzailea, honako programa hau lortuz:

```
function orek_da_mr(a:  zuhbit(T))
return (ore,sak):  (Boolean,Integer) is
  sak_ezk,sak_esk:  Integer;
  ore_ezk,ore_esk:  Boolean;
```
$\{\, \neg hutsa\_da(a) \,\}$
```
begin
  if hutsa_da(ezker(a)) then
    sak_esk := sakonera(eskuin(a));
    (ore,sak) := (sak_esk ≤ 1,1 + sak_esk);
  elsif hutsa_da(eskuin(a)) then
    sak_ezk := sakonera(ezker(a));
    (ore,sak) := (sak_ezk = 1,1 + sak_ezk);
  else
    (ore_ezk,sak_ezk) := orek_da_mr(ezker(a));
    (ore_esk,sak_esk) := orek_da_mr(eskuin(a));
    (ore,sak) := ((abs(sak_ezk - sak_esk) <= 1) and ore_ezk
                                    and ore_esk,
                      1 + handiena(sak_ezk,sak_esk));
  end if;
end orek_da_mr;
```
$\{\, ore \leftrightarrow orekatua(a) \wedge sak = sakonera(a) \,\}$

Lehenengo oinarrizko kasuari dagokion hirukotea honako hau da:

$\{\, \neg\, hutsa\_da(a) \wedge hutsa\_da(ezker(a)) \,\}$
```
  sak_esk := sakonera(eskuin(a));
  (ore,sak) := (sak_esk ≤ 1,1 + sak_esk);
```
$\{\, ore \leftrightarrow orekatua(a) \wedge sak = sakonera(a) \,\}$

Eta bigarren oinarrizko kasuari dagokion hirukotea honako hau da:

$$\{ \neg\, hutsa\_da(a) \wedge \neg\, hutsa\_da(ezker(a)) \wedge hutsa\_da(eskuin(a)) \}$$
```
  sak_ezk := sakonera(ezker(a));
  (ore,sak) := (sak_ezk = 1,1 + sak_ezk);
```
$$\{ ore \leftrightarrow orekatua(a) \wedge sak = sakonera(a) \}$$

a-ren azpizuhaitz biak ez-hutsak direnean, ezkerreko eta eskuineko azpizuhaitzei dagozkien indukzio-hipotesi biak honako hauek dira:

$$\{ \neg\, hutsa\_da(ezker(a)) \}$$
```
  (ore_ezk,sak_ezk) := orek_da_mr(ezker(a));
```
$$\{ ore\_ezk \leftrightarrow orekatua(ezker(a)) \wedge sak\_ezk = sakonera(ezker(a)) \}$$

eta

$$\{ \neg\, hutsa\_da(eskuin(a)) \}$$
```
  (ore_esk,sak_esk) := orek_da_mr(eskuin(a));
```
$$\{ ore\_esk \leftrightarrow orekatua(eskuin(a)) \wedge sak\_esk = sakonera(eskuin(a)) \}$$

4.4.1. azpiatalean era xehatuan azaltzen den bezala, kontserbazioaren axioma **(KA)** eta konjuntzioaren erregela **(KJE)** erabiliz honako hirukotearen zuzentasuna ziurta daiteke:

$$\{ \neg\, hutsa\_da(a) \wedge \neg\, hutsa\_da(ezker(a)) \wedge \neg\, hutsa\_da(eskuin(a)) \}$$
```
  (ore_ezk,sak_ezk) := orek_da_mr(ezker(a));
  (ore_esk,sak_esk) := orek_da_mr(eskuin(a));
```
$$\{ ore\_ezk \leftrightarrow orekatua(ezker(a)) \wedge sak\_ezk = sakonera(ezker(a)) \wedge$$
$$ore\_esk \leftrightarrow orekatua(eskuin(a)) \wedge$$
$$sak\_esk = sakonera(eskuin(a)) \}$$

Ondorioz, hirukote hori honako esleipen honekin konposatuz

```
(ore,sak) := ((abs(sak_ezk - sak_esk) <= 1) and ore_ezk and
          ore_esk, 1 + handiena(sak_ezk,sak_esk));
```

lortzen den programa zuzena da honako postbaldintza hau betearazteko:

$$\{ ore \leftrightarrow orekatua(a) \wedge sak = sakonera(a) \}$$

Izan ere, bi dei errekurtsibo horien ondoren honako bi formula hauek beteko dira:

$$\{ ((abs(sak\_ezk - sak\_esk) <= 1) \wedge ore\_ezk \wedge ore\_esk) \leftrightarrow$$
$$orekatua(a) \}$$

eta

$$\{\ (\ 1 + handiena(sak\_ezk, sak\_esk)\ ) = sakonera(a)\ \}$$

Indukzioaren balidazioa $E \equiv sakonera(a)$ borne-adierazpena erabiliz egin daiteke, borne-adierazpen horrek beti balio ez-negatiboa izango baitu, eta, gainera, hutsa ez den edozein a zuhaitz bitar emanda, haren ezkerreko eta eskuineko azpizuhaitzen sakonera a-ren sakonera baino txikiagoa izango da.

8.5. BIBLIOGRAFIA-OHARRAK

Ageriko errepikapenik gabe, konputazio kopuru infinitua deskribatzea ahalbidetzen duen teknika bezala definitu zuen errekurtsibitatea N. Wirth-ek gerora programazioaren eta lengoaien bilakaeran eragin handienetakoa izan zuen lanetako bat den [92] liburuan. Gaur egun, programazio-lengoaia guztiek dituzte errekurtsibitatea hainbat eratan erabiltzeko erraztasunak, hala nola algoritmoak eta datu-egiturak inplementatzeko edo objektu infinituak, objektu klaseak eta abar definitzeko. Gainera, programazio-lengoaien sintaxia eta semantika definitzeko, analizatzaileak eta konpilatzaileak definitzeko eta abarretarako ere erabilgarria da errekurtsibitatea. Lengoaia funtzionaletan gertatzen den bezala (estandartzat har daitekeen Haskell [56] lengoaian, adibidez), programazio-lengoaia batzuetan era guztietako objektuak eta funtzioak definitzeko oinarrizko mekanismoa errekurtsibitatea da. Haskellek ez du lengoaia agintzaileetakoak bezalako iterazioak eraikitzeko aurredefinitutako erarik. 50eko hamarkadaren bukaera aldera J. McCarthy-k LISP lengoaia ([70]) asmatu zuenean hasi zen lengoaia funtzionalen historia (ikus [56]). LISP lengoaia, funtzioen definizio errekurtsiboan eta errekurtsiboki definitutako zerrendetan oinarritzen da erabat. Indukzio-hipotesia oinarritzat hartuz algoritmo errekurtsiboen diseinuaren deskribapen xehatua eskaintzen aitzindarietakoa den liburu bat [86] da. Murgilketaren teknikari buruzko artikulu aitzindari bat [85] da. Gai hauek (gaztelaniaz) azaltzen dituzten geroagoko liburuen artetik honako hauek nabarmenduko ditugu (ordena kronologikoan): [13, 79, 21, 67, 68].

8.6. ARIKETAK: PROGRAMA ERREKURTSIBOEN ERATORPENA

1. Formalki eratorri datu bezala positiboa den x zenbaki osoa eta negatiboa ez den y zenbaki osoa hartu eta x^y kalkulatuko duen programa errekurtsiboa. Baina $x^y = x^{y-1} * x$ definizioa kontuan hartuz baino eraginkorragoa izango den programa bat lortu nahi da honako beste definizio hau erabiliz: y bikoitia baldin bada, $x^y = (x^{(y/2)})^2$ beteko da eta y bakoitia baldin bada, $x^y = (x^{(y/2)})^2 * x$ beteko da.

2. Formalki eratorri datu bezala $m \geq n \geq 0$ baldintza betetzen duten m eta n zenbaki osoak hartu eta honako konbinazio-zenbaki hau kalkulatuko duen programa errekurtsiboa:

$$\left(\begin{array}{c} m \\ n \end{array} \right) = \frac{m!}{n! * (m-n)!}$$

   Baina eratorpen hori bitarteko kalkuluetan oso handiak diren zenbaki osoak sortzen dituzten faktorialak kalkulatu gabe egin nahi da.

   Laguntza: Konbinazio-zenbakiak kalkulatzeko beste era bat honako hau da:

$$\left(\begin{array}{c} m \\ n \end{array} \right) = \left(\begin{array}{c} m-1 \\ n-1 \end{array} \right) + \left(\begin{array}{c} m-1 \\ n \end{array} \right)$$

3. Formalki eratorri datu bezala $c \leq n$ baldintza betetzen duten eta positiboak diren c eta n bi zenbaki oso hartu eta n elementuak adierazten duen luzera eta c bateko dituzten sekuentzia bitarren kopurua, hau da, $n - c$ zeroz eta c batekoz osatutako sekuentzien kopurua kalkulatuko duen programa errekurtsiboa.

4. Murgilketa erabiliz, formalki eratorri datu bezala bektore bat eta balio bat hartu eta balio hori bektore horretan agertzen al den erabakiko duen programa errekurtsiboa.

5. Murgilketa erabiliz, formalki eratorri datu bezala bektore bat hartu eta bektore hori palindromoa den ala ez erabakiko duen programa errekurtsiboa.

6. Murgilketa erabiliz, formalki eratorri datu bezala x balio bat eta $a_0 + a_1 * x + a_2 * x^2 + \ldots + a_n * x^n$ erako polinomio baten a_i koefizienteak dituen $A(0..n)$ bektorea hartu eta polinomioaren balioa kalkulatuko duen programa errekurtsiboa.

7. Har dezagun osoak diren zenbaki positiboentzat definitutako honako funtzio hau:

$$fucs(n) = \begin{cases} 1 & \text{baldin } n = 1 \\ fucs(n/2) & \text{baldin } n \geq 2 \wedge bikoitia(n) \\ fucs(n/2) + fucs((n/2)+1) & \text{baldin } n \geq 2 \wedge bakoitia(n) \end{cases}$$

Definizio hori zuzenean inplementatuz lortu den honako funtzio honek Fibonacciren terminoak kalkulatzen dituen funtzioaren antzekoa den kalkulu-errepikapena eragiten du.

```
function fucs_zuz(n:  Integer) return f:  Integer is
  x,y:  Integer;
{ n ≥ 1 }
begin
  if n = 1 then
    f := 1;
  elsif n % 2 = 0 then
    f := fucs_zuz(n/2);
  else
    x := fucs_zuz(n/2);
    y := fucs_zuz(n/2 + 1);
    f := x + y;
  end if;
end fucs_zuz;
{ f = fucs(n) }
```

Murgilketa erabiliz, diseinatu kalkulurik errepikatu gabe emaitza bera lortuko duen funtzio errekurtsiboa.

Laguntza: Funtzio murgiltzaileak bi zenbaki osoz eratutako bikote bat itzuli beharko du emaitza bezala.

8. Osoa den eta 2 eta 9 balioen artean dagoen *oin* zenbaki osoa, *oin* oinarrian dagoen eta negatiboa ez den n zenbaki osoa eta negatiboa ez den b zenbaki osoa emanda, n-ri hamar oinarrian dagokion balioa bider oin^b itzuliko du $oin10$ funtzioak. b-ren balioa 0 baldin bada, n-ri hamar

oinarrian dagokion balioa lortuko da.

$$oin10(n, oin, b) = \begin{cases} n * oin^b & \text{baldin } n < oin \\ (n\%10) * oin^b + & \text{baldin } n \geq oin \\ \quad oin10(n/10, oin, b+1) & \end{cases}$$

Jarraian erakusten den zuzenean egindako inplementazioak ez du bukaerako errekurtsibitaterik.

```
function oin10_zuz(n,oin,b:  Integer)
return h:  Integer is
{ 2 ≤ oin ≤ 9 ∧ n ≥ 0 ∧
        ∀k ( k ≥ 0 → ( ( (n/10^k)%10 ) ≤ oin − 1 ) ) ∧ b ≥ 0 }
begin
  if n < oin then
    h := n*(oin**b);
  else
    h := oin10_zuz(n/10,oin,b+1);
    h := (n % 10)*(oin**b) + h;
  end if;
end oin10_zuz;
{ h = oin10(n,oin,b) }
```

Inplementazio horretan $**$ eragilea berreketa da, hau da, $oin ** b$ espresioak oin^b adierazten du.

Murgilketa erabiliz, diseinatu kalkulu bera egiten duen eta bukaerako errekurtsibitatea duen funtzio errekurtsiboa.

9. Har dezagun sekuentzia baten alderantzizkoa kalkulatzen duen eragiketaren honako espezifikazio ekuazional hau:

Mota: $sekuentzia(T)$
Lag: (ez dago)
Eragiketak:
$alderantzizkoa$: $sekuentzia(T) \rightarrow sekuentzia(T)$
Ekuazioak:
(1) $alderantzizkoa(\langle\,\rangle) = \langle\,\rangle$
(2) $alderantzizkoa(x \bullet q) = alderantzizkoa(q) @ \langle x \rangle$

Espezifikazio horri jarraituz zuzenean lortutako honako inplementazioak ez du bukaerako errekurtsibitaterik:

```
function alder(s:  sekuentzia(T))
return e:  sekuentzia(T) is
begin
  if hutsa_da(s) then
    e := <>;
  else
    e := alder(hondarra(s)) @ <lehena(s)>;
  end if;
end alder;
```
$\{\ e = alderantzizkoa(s)\ \}$

Murgilketa erabiliz, diseinatu *alder* funtzioak burutzen duen kalkulu bera egiten duen eta bukaerako errekurtsibitatea duen funtzioa.

10. Zuhaitz bitar bat *ona* dela esango da, zuhaitz bitar horretan adabegi bakoitzaren balioa bere ezkerreko azpizuhaitzeko adabegien baturarekin eta bere eskuineko azpizuhaitzeko adabegien baturarekin bat baldin badator (bakoitza bere aldetik).

    Har dezagun zuhaitz bitar bat ona den ala ez erabakitzen duen eragiketaren eta zuhaitz bitar baten adabegien balioen batura kalkulatzen duen *batura* eragiketaren honako espezifikazio ekuazional hau:

    Mota: $zuhbit(oso)$
    Lag: bool
    Eragiketak:
    ona: $zuhbit(oso) \to bool$
    $batura$: $zuhbit(oso) \to oso$
    Ekuazioak:
    (1) $ona(hutsa) = \mathsf{T}$
    (2) $ona(errotu(e, ezk, esk)) = (\ batura(ezk) = e\ ) \wedge (\ batura(esk) = e\ ) \wedge ona(ezk) \wedge ona(esk)$
    (3) $batura(hutsa) = 0$
    (4) $batura(errotu(e, ezk, esk)) = e + batura(ezk) + batura(esk)$

    Diseinatu datu bezala zuhaitz bitar bat hartu eta zuhaitz bitar hori ona den ala ez erabakiko duen *ona_zuz* funtzioa. Eratorpen hori egitean, *ona* eragiketaren espezifikazio ekuazionala zuzenean itzuli eta *batura* funtzioa inplementatuta dagoela kontsideratu.

    Funtzio errekurtsibo hori eraginkortasun aldetik oso txarra izango da, azpizuhaitz bakoitza bi aldiz aztertuko baitu.

Murgilketa erabiliz, diseinatu zuhaitz bat ona den ala ez erabakiko duen funtzioa. Funtzio horrek datu bezala hartutako zuhaitzaren azpizuhaitz bakoitza behin bakarrik aztertu beharko du.

11. Zuhaitz bitar batean adabegi bakoitzaren balioa bere ezkerreko azpizuhaitzeko adabegiena baino handiagoa edo berdina baldin bada eta bere eskuineko azpizuhaitzeko adabegiena baino txikiagoa baldin bada, zuhaitz bitar hori *bilaketa-zuhaitza* dela esango da. Har dezagun zuhaitz bitar bat bilaketa-zuhaitza den ala ez erabakitzen duen eragiketaren honako espezifikazio ekuazional hau:

Mota: $zuhbit(oso)$
Lag: bool
Eragiketak:
$bilaketa$: $zuhbit(oso) \to bool$
Ekuazioak:
(1) $bilaketa(hutsa) = \mathsf{T}$
(2) $bilaketa(errotu(e, ezk, esk)) = bilaketa(ezk) \wedge$
$bilaketa(esk) \wedge e \geq handiena(ezk) \wedge e < txikiena(esk)$

Espezifikazio horretan *handiena* eta *txikiena* eragiketa laguntzaileek zenbaki osozko zuhaitz bitar bateko balio handiena eta txikiena lortzen dute hurrenez hurren.

bilaketa funtzioa zuzenean inplementatuz lortu den *bz_da* funtzioak kalkuluak errepikatzen ditu:

```
function bz_da(a:  zuhbit(Integer))
return b:  Boolean is
  hand_ezk,txik_esk:  Integer;
  x,y:  Boolean;
begin
  if hutsa_da(a) then
    b := true;
  else
    hand_ezk := handiena(ezker(a));
    txik_esk := txikiena(eskuin(a));
    x := bz_da(ezker(a));
    y := bz_da(eskuin(a));
    b := x and y and (erroa(a) >= hand_ezk) and
                 (erroa(a) < txik_esk);
  end if;
end bz_da;
```

$\{\, b \leftrightarrow bilaketa(a) \,\}$

a) Funtzio laguntzaileen espezifikazio aljebraikoari dagozkion ekuazioak eman:

Mota: $zuhbit(oso)$
Lag: (ez dago)
Eragiketak:
$handiena$: $zuhbit(oso) \to oso$
$txikiena$: $zuhbit(oso) \to oso$
Ekuazioak:
$\vdots$

b) Murgilketaren teknika erabiliz, diseinatu datu bezala hartutako zuhaitzaren azpizuhaitz bakoitza bi aldiz ez aztertzeagatik *bz_da* baino eraginkorragoa izango den funtzio baliokidea.

Laguntza: Definitu datu bezala zuhaitz bitar bat hartu eta zuhaitz hori bilaketa-zuhaitza den ala ez adierazten duen balio boolearra eta zuhaitzeko balio handiena eta txikiena adierazten duten bi zenbaki oso itzuliko dituen funtzio murgiltzailea. Horrenbestez, funtzio horrek hiru emaitza itzuli beharko ditu.

12. Har ditzagun (bektoreak adierazten dituzten) zenbaki osozko bi sekuentzia, esate baterako, $s = \langle s_1, s_2, \ldots, s_k \rangle$ eta $r = \langle r_1, r_2, \ldots, r_k \rangle$. Sekuentzia horien arteko biderketa eskalarraren definizioa honako hau da:

$$\sum_{i=1}^{k} r_i * s_i$$

Adibidez, $bid\_esk(\langle 3, 2, 1 \rangle, \langle 2, 5, 4 \rangle) = (3*2)+(2*5)+(1*4) = 20$. Sekuentziek luzera desberdina baldin badute, *bid_esk* eragiketak errorea sortuko du.

a) Espezifikatu *bid_esk* funtzioa ekuazioen bidez.

b) Idatzi ekuazioen bidez emandako *bid_esk* eragiketaren espezifikazioa zuzenean inplementatzen duen *bid_esk_zuz* funtzioa. Horretarako, aurrebaldintzatzat argumentu biak luzera bereko bi sekuentzia direla ezarri.

c) Murgilketaren teknika erabiliz, diseinatu *bid_esk_zuz* funtzioaren baliokidea den eta bukaerako errekurtsibitatea duen funtzio bat.

Bibliografia

[1] Rod Adams. *An Early History of Recursive Functions and Computability from Godel to Turing*. Docent Press, 2011.

[2] Suad Alagic and Michael A. Arbib. *The Design of Well-Structured and Correct Programs*. Springer, 1978.

[3] Allen L. Ambler, Donald I. Good, James C. Browne, Wilhelm F. Burger, Richard M. Cohen, Charles G. Hoch, and Robert E. Wells. Gypsy: A language for specification and implementation of verifiable programs. *SIGPLAN Not.*, 12(3):1–10, March 1977.

[4] Krzysztof R. Apt. Ten years of Hoare's logic: A survey. Part I. *ACM Trans. Program. Lang. Syst.*, 3(4):431–483, October 1981.

[5] Krzysztof R. Apt. Ten years of Hoare's logic: A survey. Part II: Nondeterminism. *Theor. Comput. Sci.*, 28:83–109, 1984.

[6] Krzysztof R. Apt, Frank S. de Boer, and Ernst-Rüdiger Olderog. Modular verification of recursive programs. *CoRR*, abs/0907.4316, 2009.

[7] Jacques Arsac. *Foundations of Programming*. Academic Press, 1985.

[8] Jacques Arsac and Yves Kodratoff. Some techniques for recursion removal from recursive functions. *ACM Trans. Program. Lang. Syst.*, 4(2):295–322, April 1982.

[9] Roland C. Backhouse. *Program Construction and Verification*. Prentice-Hall, 1986.

[10] Roland C. Backhouse. *Program Construction: Calculating Implementations from Specifications*. Wiley, 2003.

[11] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Report on the algorithmic language ALGOL 60. *Commun. ACM*, 3(5):299–314, May 1960.

[12] Frédéric Badeau and Arnaud Amelot. Using B as a high level programming language in an industrial project: Roissy VAL. In *Proceedings of the 4th International Conference on Formal Specification and Development in Z and B*, ZB'05, pages 334–354, Berlin, Heidelberg, 2005. Springer-Verlag.

[13] José Luis Balcázar. *Programación Metódica*. McGraw-Hill, 1993.

[14] Michael Balser, Wolfgang Reif, Gerhard Schellhorn, Kurt Stenzel, and Andreas Thums. Formal system development with KIV. In *Fundamental Approaches to Software Engineering, Third Internationsl Conference, FASE 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, pages 363–366, 2000.

[15] John Barnes and Praxis Critical Systems Limited. *High Integrity Ada: The SPARK Approach*. Programming Languages. Addison-Wesley, 1997.

[16] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt. *Verification of Object-oriented Software: The KeY Approach*. Springer-Verlag, Berlin, Heidelberg, 2007.

[17] Bernhard Beckert, Tony Hoare, Reiner Hähnle, Douglas R. Smith, Cordell Green, Silvio Ranise, Cesare Tinelli, Thomas Ball, and Sriram K. Rajamani. Intelligent systems and formal methods in software engineering. *IEEE Intelligent Systems*, 21(6):71–81, 2006.

[18] Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. Météor: A successful application of B in a large project. In *World Congress on Formal Methods*, pages 369–387, 1999.

[19] Michel Bidoit, Hans-Jörg Kreowski, Pierre Lescanne, Fernando Orejas, and Donald Sannella, editors. *Algebraic System Specification and Development, a Survey and Annotated Bibliography*, volume 501 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[20] Garrett Birkhoff and Saunders Mac Lane. *A Survey of Modern Algebra*. AKP classics. A.K. Peters, Wellesley, Massachusetts, 1997.

[21] Gilles Brassard and Paul Bratley. *Fundamentos de Algoritmia*. Fuera de colección Out of series. Pearson Educación, 1997.

[22] Manfred Broy and Bernd Krieg-Bruckner. Derivation of invariant assertions during program development by transformation. *ACM Trans. Program. Lang. Syst.*, 2(3):321–337, July 1980.

[23] Rod M. Burstall and John Darlington. A transformation system for developing recursive programs. *J. ACM*, 24(1):44–67, January 1977.

[24] Michel R. V. Chaudron, Jan Tretmans, and Klaas Wijbrans. Lessons from the application of formal methods to the design of a storm surge barrier control system. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *World Congress on Formal Methods*, volume 1709 of *Lecture Notes in Computer Science*, pages 1511–1526. Springer, 1999.

[25] Lori A. Clarke and David S. Rosenblum. A historical perspective on runtime assertion checking in software development. *SIGSOFT Softw. Eng. Notes*, 31(3):25–37, May 2006.

[26] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude - a High-performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic*. Springer-Verlag, Berlin, Heidelberg, 2007.

[27] John Darlington and Rod M. Burstall. A system which automatically improves programs. *Acta Informatica*, 6:41–60, 1976.

[28] Julius Wilhelm Richard Dedekind. *Was sind und was sollen die Zahlen?(What are and what should the numbers be?)*. Braunschweig, 1888.

[29] Jyotirmoy V. Deshmukh and E. Allen Emerson. Verification of recursive methods on tree-like data structures. In *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*. IEEE, 2009.

[30] Razvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report : The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. AMAST series in computing. World scientific, Singapore, River Edge, NJ, London, 1998.

[31] Edsger W. Dijkstra. Letters to the editor: Go to statement considered harmful. *Commun. ACM*, 11(3):147–148, March 1968.

[32] Edsger W. Dijkstra. Notes on structured programming. Circulated privately, April 1970.

[33] Edsger W. Dijkstra. *A Discipline of Programming*. Englewood Cliffs, N.J. : Prentice-Hall, 1st edition, 1976.

[34] Edsger W. Dijkstra. Guarded commands, non-determinancy and a calculus for the derivation of programs. In Friedrich L. Bauer and Klaus Samelson, editors, *Language Hierarchies and Interfaces, International Summer School, Marktoberdorf, Germany, July 23 - August 2, 1975*, volume 46 of *Lecture Notes in Computer Science*, pages 111–124. Springer, 1976.

[35] Edsger W. Dijkstra and W. H. J. Feijen. *A Method of Programming*. Addison-Wesley, 1988.

[36] Geoff Dromey. *Program Derivation - The Development of Programs from Specifications*. International computer science series. Addison-Wesley, 1989.

[37] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification I*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1985.

[38] Hartmut Ehrig and Bernd Mahr. Algebraic techniques in software development: A review of progress up to the mid nineties. In *Current Trends in Theoretical Computer Science: Entering the 21st Century*, pages 134–152. World Scientific, 2001.

[39] Gerwin Klein et al. sel4: Formal Verification of an OS Kernel. In JeannaÑeefe Matthews and Thomas E. Anderson, editors, *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October, 11-14, 2009*. ACM, 2009.

[40] Robert W. Floyd. Assigning meaning to programs. In *Proceedings of the Symposium on Applied Maths*, volume 19, pages 19–32. AMS, 1967.

[41] M. Foley and Charles Antony Richard Hoare. Proof of a recursive program: Quicksort. *Comput. J.*, 14(4):391–395, 1971.

[42] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 1999.

[43] Joseph A. Goguen. Some design principles and theory for obj-o, a language to express and execute algebraic specification for programs. In *Proceedings of the International Conference on Mathematical Studies of Information Processing*, pages 425–473, London, UK, 1979. Springer-Verlag.

[44] Joseph A. Goguen, Eric G. Wagner, and James W. Thatcher. An initial algebra approach to the specification, correctness, and implementation

of abstract data types. Technical Report RC 06487, IBM US Research Centers (Yorktown, San Jose, Almaden, US), 1976.

[45] Joseph A. Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing obj. In J. Goguen and G. Malcolm, editors, *Software Engineering with OBJ: algebraic specification in action*, Advances in Formal Methods, pages 3–167. Springer, 2000.

[46] Hermann H. Goldstine and John von Neumann. Planning and coding of problems for an electronic computing instrument. *Institute for Advanced Study, Princeton, N. J.*, 3, 1948. Reprinted in John Neumann's Collected Works, Vol. 5, A H Taub, Ed., Pergamon, London, 1963, pp 215-235.

[47] Gerald A. Gorelick. *A Complete Axiomatic System for Proving Assertions about Recursive and Non-recursive Programs*. Department of Computer Science: Technical report. University of Toronto, Department of Computer Science, 1975.

[48] David Gries. *The Science of Programming.* Springer, 1981.

[49] John V. Guttag. *The specification and application to programming of abstract data types*. PhD thesis, University Of Toronto (Canada), 1975.

[50] Anthony Hall. Seven myths of formal methods. *IEEE Software*, 7(5):11–19, September 1990.

[51] David Harel. *First-order Dynamic Logic*. Lecture notes in computer science. Springer, 1979.

[52] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, October 1969.

[53] Charles Antony Richard Hoare. Procedures and parameters: An axiomatic approach. *Symposium on Semantics of Algorithmic Languages*, pages 102–116, 1971.

[54] Charles Antony Richard Hoare and Niklaus Wirth. An axiomatic definition of the programming language PASCAL. *Acta Informatica*, 2(4):335–355, 1973.

[55] Peter V. Homeier and David F. Martin. Mechanical verification of mutually recursive procedures. In *Automated DeductionCADE-13*, pages 201–215. Springer, 1996.

[56] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. A history of Haskell: being lazy with class. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, HOPL III, pages 12–1–12–55, New York, NY, USA, 2007. ACM.

[57] Kathleen Jensen and Niklaus Wirth. *Pascal : User Manual and Report.* Lecture Notes in Computer Science. Springer, Berlin, 1974.

[58] Anne Kaldewaij. *Programming: The Derivation of Algorithms.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[59] Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[60] Thomas Kleymann. *Hoare Logic and VDM: Machine-Checked Soundness and Completeness Proofs.* PhD thesis, University of Edinburgh, 1998.

[61] Peter Ernst Lauer. Consistent formal theories of the semantics of programming languages. Technical Report TR 25-121, IBM. Laboratory Vienna (Vienne, AT), 1971.

[62] K. Rustan M. Leino. Developing verified programs with dafny. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1488–1490, Piscataway, NJ, USA, 2013. IEEE Press.

[63] Barbara Liskov and Stephen Zilles. Programming with abstract data types. In *Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages*, pages 50–59, New York, NY, USA, 1974. ACM.

[64] Yanhong A. Liu and Tim Teitelbaum. Systematic derivation of incremental programs. *Science Computing Programming*, 24(1):1–39, 1995.

[65] Jacques Loeckx and Kurt Sieber. *The Foundations of Program Verification, 2nd ed.* Wiley-Teubner, 1987.

[66] Ralph L. London, John V. Guttag, James J. Horning, Butler W. Lampson, James G. Mitchell, and Gerald J. Popek. Proof rules for the programming language euclid. *Acta Inf.*, 10:1–26, 1978.

[67] Narciso Martí-Oliet, Clara M. Segura Díaz, and José A. Verdejo López. *Especificación, Derivación y Análisis de Algoritmos: Ejercicios Resueltos.* Prentice Practica. Pearson Educación, 2006.

[68] Narciso Martí-Oliet, Clara M. Segura Díaz, and José A. Verdejo López. *Algoritmos Correctos y Eficientes*. Ibergarceta, 2012.

[69] Abraham H. Maslow. *The Psychology of Science: A Reconnaissance*. Maurice Bassett, 2004.

[70] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, Part i. *Communications of the ACM*, 3(4):184–195, April 1960.

[71] Bertrand Meyer. Eiffel: A language and environment for software engineering. *Journal of Systems and Software*, 8(3):199–246, 1988.

[72] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1988.

[73] Francis Lockwood Morris and Clifford B. Jones. An early program proof by Alan Turing. *IEEE Annals of the History of Computing*, 6(2):139–143, 1984.

[74] Peter D. Mosses. *CASL Reference Manual, The Complete Documentation of the Common Algebraic Specification Language*, volume 2960 of *Lecture Notes in Computer Science*. Springer, 2004.

[75] The Institute of Electrical and Eletronics Engineers. Ieee standard glossary of software engineering terminology. IEEE Standard, September 1990.

[76] Fernando Orejas. ¿Es posible construir software que no falle? *Blog El Año de Turing, El País*, 26 de Julio de 2012. `http://blogs.elpais.com/turing/2012/07/es-posible-construir-software-que-no-falle.html`.

[77] Helmut A. Partsch. *Specification and Transformation of Programs. A Formal Approach to Software Development*. Springer-Verlag, 1990.

[78] Giuseppe Peano. *Arithmetices Principia, Nova Methodo Exposita (The principles of arithmetic, presented by a new method)*. Fratres Bocca, Turin, 1889.

[79] Ricardo Peña. *Diseño de programas: Formalismo y Abstracción*. Pearson Education, 1993.

[80] Rózsa Péter. Über den zusammenhang der verschiedenen begriffe der rekursiven funktion. *Mathematische Annalen*, 110:612–632, 1934.

[81] Rózsa Péter. *Recursive Functions in Computer Theory*. Computers and their applications. Ellis Horwood, 1981.

[82] Alberto Pettorossi and Maurizio Proietti. Rules and strategies for transforming functional and logic programs. *ACM Computing Surveys*, 28:360–414, 1996.

[83] Alberto Pettorossi and Maurizio Proietti. Transformation of logic programs. In D. M. Gabbayand, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 5*, pages 697–787. Oxford University Press, 1998.

[84] G. J. Popek, J. J. Horning, B. W. Lampson, J. G. Mitchell, and R. L. London. Notes on the design of Euclid. *ACM SIGPLAN Notices*, 12(3):11–18, March 1977.

[85] Celestí Rosselló, José L. Balcázar, and Ricardo Peña. Deriving specifications of embeddings in recursive program design. *Structured Programming*, 10(3):133–145, 1989.

[86] Pierre-Claude Scholl. *Algorítmica y Representación de Datos. 2.Recursividades y Árboles*. Masson, 1986.

[87] Thomas Schreiber. Auxiliary variables and recursive procedures. In Michel Bidoit and Max Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 697–711. Springer Berlin Heidelberg, 1997.

[88] Hisao Tamaki and Taisuke Sato. Unfold/fold transformation of logic programs. In Sten-Åke Tärnlund, editor, *Proceedings of the Second International Logic Programming Conference, Uppsala University, Uppsala, Sweden*, pages 127–138, 1984.

[89] Alan M. Turing. Checking a large routine. In *High-Speed Automatic Calculating Machines University Mathematical Laboratory*, pages 67–69, 1949. A corrected version is printed in [73]. The original is reprinted in [91, pp. 70–72].

[90] David von Oheimb. Hoare logic for mutual recursion and local variables. In C.Pandu Rangan, V. Raman, and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 168–180. Springer Berlin Heidelberg, 1999.

[91] Michael R. Williams and Martin Campbell-Kelly, editors. *The Early British Computer Conferences*. Charles Babbage Institute reprint series for the history of computing. MIT Press, 1989.

[92] Niklaus Wirth. *Algorithms + Data Structures = Programs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1978.

[93] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36, October 2009.

[94] William A. Wulf, Ralph L. London, and Mary Shaw. An introduction to the construction and verification of alphard programs. *IEEE Trans. Software Eng.*, 2(4):253–265, 1976.

Glosarioa

adierazpen (espresio) 19, 50, 53, 81

aldagai 13, 17, 20, 21, 23

aldagai libre 23, 24, 30, 31

aldibereko esleipen 196, 198, 201

aldibereko ordezkapen 33, 35

algoritmo errekurtsibo 141

asertzio 17, 19–21, 23, 31, 33, 34, 37, 60, 81

aurrebaldintza (aurreko baldintza) 37, 48, 51, 91

aurre-ondoetako espezifikazio (aurre-ondoetako zehaztapen) 13, 14, 37, 39, 48

axioma 49, 50, 56

baldintza 14

baldintzaren erregela 65, 67

baldintzazko agindu 64, 66, 68

begizta 74

bektore (array) 20, 22, 29

bukaera-baldintza 196, 201

datu-mota abstraktu 150, 155

dedukzio 149, 150, 173

Informatika sailean argitaratu diren beste liburu batzuk

Algoritmika
Rosa Arruabarrena
1997an argitaratua
ISBN: 84-86967-82-1

Ordenadore bidezko irudigintza
Joseba Makazaga, Asier Lasa
1998an argitaratua
ISBN: 84-86967-90-2

Oinarrizko programazioa. Ariketa-bilduma
Arantza Diaz de Illarraza, Kepa Sarasola
1999an argitaratua
ISBN: 84-8438-002-5

Zirkuitu elektriko eta elektronikoen oinarrizko analisia
Olatz Arbelaitz, Txelo Ruiz
2001ean argitaratua
ISBN: 84-8438-018-1

LINUX Sistemaren eta sarearen administrazioa
Iñaki Alegria
2003an argitaratua
ISBN: 84-8438-040-8

Sistema Digitalen Diseinu-hastapenak. Oinarrizko kontzeptuak eta adibideak
Olatz Arbelaitz eta beste
2005ean argitaratua
ISBN: 84-8438-069-6

Softwarearen ingeniaritza [I. atala: Softwarearen garapenaren zenbait arlo]
Jose Ramon Zubizarreta
2006an argitaratua
ISBN: 84-8438-085-8

Softwarearen ingeniaritza [II. ATALA: Garapen monolitikotik hiru mailako arkitekturara bezero/zerbitzariak bisitatuz]

Jose Ramon Zubizarreta
2009an argitaratua
ISBN: 978-84-8438-165-5

Linux: Sistemaren eta sarearen administrazioa 2. argitaraldia (Debian eta Ubuntu)

Iñaki Alegria eta Roberto Cortiñas
2008an argitaratua
ISBN: 978-84-8438-178-5

TAPE Testu-analisirako Perl erremintak

Aitzol Astigarraga eta beste
2009an argitaratua
ISBN: 978-84-8438-233-1

TCP/IP Sareak (3. argitaraldia)

Jose Mª Ribadeneyra Sicilia
2009an argitaratua
ISBN: 978-84-8438-235-5

Robot mugikorrak. Oinarriak

Aitzol Astigarraga eta Elena Lazkano
2011n argitaratua
ISBN: 978-84-8438-377-2