2 Programen espezifikazioa

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 programen espezifikazioak: batetik, datuek bete behar dituzten propietateak adierazten dira; bestetik, emaitzaren ezaugarriak deskribatzen dira. Bi parte horiek elkarren osagarri dira, eta bien konposaketak osotasuna ematen dio deskribapenari.

Datuen eta emaitzaren arteko erlazio hori funtzio baten gisara uler daiteke: datuak lirateke funtzioaren sarrera, eta emaitza, berriz, funtzioaren irteera. Espezifikazioa funtzioaren deskribapen zehatza da, horretara. Programa, ordea, funtzioaren inplementazioa da, hau da, datuetatik abiatuta emaitzak lortzeko modu jakin bat. Eta, noski, espezifikazio bat emanda, programa bat baino gehiago inplementa daitezke espezifikazio hori betetzen dutenak.

Helburuak

Espezifikazioa programagintzako hainbat alorretan erabiltzen da.

Programen diseinuan berebiziko garrantzia du. Izan ere, edozein programaren diseinurako premiazkoa da ebatzi beharreko problemaren ezagutza zehatza (anbiguetaterik gabea). Ezagutza hori diseinuari ekin aurretik eduki behar da, diseinatu ahala problemaren deskribapenaz jabetzea akats-iturri larria delako. Problema espezifikatzeak ideiak egituratzera behartzen gaitu eta aspektu nabariak agerian uztera, bidenabar. Aspektu hauek, espezifikazioan ez bada, diseinuan agertzen dira edo, are okerrago, ez dira agertzen, eta orduan programa desegokia izango da, ez baitu nahi dugun problema ebatziko.

Programak eta aplikazio informatikoak talde-lanean egin ohi dira. Hori horrela, espezifikazioa komunikaziorako bitarteko ezinbestekoa da. Modulutako banaketa sistemaren eginbeharrak zehatz-mehatz ezagutuz egin behar da, eta modulu bakoitzeko diseinatzaileak, inolako anbiguetaterik gabe ezagutu behar ditu problema orokorra eta beste moduluekiko elkarrekintzak.

Programen testak diseinatzeko ere ezinbestekoak dira espezifikazioak: ezin dira testeatu programak beren eginkizuna zehaztasunez ezagutzen ez bada.

Bestalde, ez dugu ahantzi behar programak behin egin eta askotan irakurtzen direla. Programa bat mantentzeko beharrezkoa da dokumentazio aproposa erabiltzea, edonork programa horrek zer egiten duen zehazki uler dezan aparteko ahaleginik gabe. Gerora

ere, aldaketaren bat burutu beharko denean, ezinbestekoa izango da dokumentazioa. Kontuan hartu behar da, gainera, aldaketak egiten dituena eta hasierako diseinua egin zuena ez direla pertsona bera izaten normalean.

Programen egiaztapenerako ere espezifikazioak dira abiapuntua, azken batean programak egiaztatzea espezifikazioak betetzen dituztela formalki frogatzea baita, hau da, diseinatu zeneko eginbeharrak betetzen dituela frogatzea. Beraz, egiaztapenean oinarritzen diren diseinu-metodologien oinarria ere badira espezifikazioak.

Programen eratorpena eta sintesi automatikoa egiten duten aplikazioek ere espezifikazioetan dute oinarria. Arlo honetan, espezifikazio-lengoaia exekutagarriak ere badira, hau da, programazio-lengoaia bilakatu diren espezifikazio-lengoaiak, nolabait problemaren deskribapena egikaritzen dutenak.

Hala bada, besteak beste, diseinuan, talde-lanean, probetan, mantentze-lanetan, egiaztapenean, eratorpenean eta sintesian baliagarriak dira guztiz espezifikazioak.

Ezaugarriak

Esana dugu espezifikazioei zehatz eta argi izatea eskatzen zaiela, jatorrizko problemaren ezagutza anbiguetaterik gabea ekarri behar dutelako. Oro har, espezifikazio egokiek propietate hauek izan beharko lituzkete:

- argitasuna (ulerterraza)
- laburtasuna (erredundantziarik gabea)
- doitasuna (anbiguotasunik ez)
- murriztasunaren eta orokortasunaren arteko oreka

Ezaugarri horiek betetzen dituzten deskribapenak espezifikazio egokitzat jotzen dira. Ez dago molde bakarra espezifikazioak idazteko. Formatu eta lengoaia desberdinak balia daitezke espezifikatzean.

Bi sailetan banatu ohi dira espezifikazio-lengoaiak: formalak eta informalak.

Lengoaia bat formaltzat hartuko dugu baldin eta bere sintaxia eta semantika formalki definituta badaude (zehaztasun matematikoz). Espezifikazio ez-formalerako teknikek, aldiz, edozein abstrakzio funtzionalen aspektu nabariak klausulen bidez adierazteko formatua finkatzen dute. Moldea bai, baina klausulak adierazteko lengoaia formalik ez dute ezartzen. Espezifikatzen duenaren esku geratzen da, hortaz, notazio matematiko eta/edo teknikoaren hautaketa, lengoaia naturalaren laguntzaz deskribapen argiak, laburrak eta zehatzak idatzi ahal izateko.

Helburuak baldintzatuko du lengoaiaren formaltasun-maila: taldeen arteko komunikaziorako, test-diseinurako edo dokumentaziorako lengoaia ez oso formala nahikoa izan daiteke; baina programen egiaztapenerako, eratorpenerako, sintesirako edo espezifikazioen exekuziorako ezinbestekoa da lengoaia formala izatea, espezifikazioak matematikoki eta/edo automatikoki tratatzen baitira.

2.2 Asertzioak: egoera-multzoak adierazteko formulak

Aurreko atalean esan dugunez, espezifikazioak datu-sarreren eta emaitzen arteko erlazioa adierazteko balio du: azken batean, *zer* egiten duen programa batek. Baina

espezifikazioak ez du adierazten zer egin behar den erlazio hori lortzeko; hau da, *nola* kalkulatzen diren emaitzak datu-sarreretatik abiatuz. Kontuan hartuta, gainera, espezifikazio bat era desberdinetan inplementa daitekeela, espezifikazioak ez du deskribatzen programa edo ebazpen jakin baten barruko antolamendua.

Programa bat aginduen segidatzat har daiteke. Agindu bakoitzak programaren aldagai baten edo batzuen balioak aldatzen ditu. Aldagaien hasierako balioak ezagutzen baditugu, orduan ezagut daiteke zein den agindu baten exekuzioaren emaitza. Horrela, datu-sarreratik emaitzetara gertatzen den transformazio-prozesua esplizituki errepresenta daiteke.

Programaren *egoera* deituko diogu programaren aldagaiek une jakin batean dituzten balioen multzoari. Beraz, agindu bakoitzak egiten duena programaren egoera aldatzea da.

Horrela bada, exekuzioaren une jakin bateko *konputazio-egoerak* (edo *egoerak*) programa bateko aldagai guztien balioen deskribapena dira:

$$\{aldagaiak\} \mapsto \{balioak\}$$

Konputazio-egoeraren kontzeptua hurrengo adibidearekin deskribatuko dugu. Demagun $x = 31$, $y = 7$ eta hurrengo P programa ditugula:

```
- - Datu-sarrerak: x, y zenbaki osokoak
- - Emaitzak: z, h zenbaki osokoak
begin
(1)
   z := 0;
(2)
   h := x;
(3)
   while h >= y loop
      z := z+1;
   (4)
      h := h-y;
   (5)
   end loop;
(6)
end ;
```

non x eta y-ren zatidura (z) eta hondar (h) osoa kalkulatzen den. Programa honen hasierako konputazio-egoera honako hau da:

(1) $\{\ x = 31\ \wedge\ y = 7\ \}$

Programan agertzen diren beste aldagaiei (hau da, z eta h) oraindik ez diegu baliorik eman, eta horregatik hasierako konputazio-egoeran ez dira agertzen: edozein balio izan dezakete. Lehenengo agindua exekutatu ondoren (`z := 0;`), z aldagaiak 0 balioa hartzen du. Beraz, bigarren konputazio-egoera

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

da. Era berean, bigarren agindua (`h := x;`) exekutatutakoan gelditzen den programaren konputazio-egoera

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

da. Ondoren, *while* aginduaren exekuzioa hasten da. Iterazioak aurrera egin ahala, aldagaiek honako taula honetan adierazitako balioak hartzen dituzte:

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

Azkenik, *while* aginduaren exekuzioa bukatu ondoren, programaren konputazio-egoera

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

da. Horrenbestez, programak honako funtzio hau inplementatzen du

$$\{\ x = 31\ \wedge\ y = 7\ \} \mapsto \{\ x = 31\ \wedge\ y = 7\ \wedge\ z = 4\ \wedge\ h = 3\ \}$$

non $\{\ x = 31\ \wedge\ y = 7\ \}$ hasierako egoera den eta $\{\ x = 31\ \wedge\ y = 7\ \wedge\ z = 4\ \wedge\ h = 3\ \}$ bukaerako egoera den.

Jakina, aipatutako funtzioa ez da orokorra, datu-sarrera jakin bat baino kontuan hartzen ez duelako. Hau da, x aldagaiak 31 balioa hartzen ez badu (edo y aldagaiak 7), aurreko funtzioak ez du bukaerako balio egokia adieraziko. Gainera, kontuan izan behar da programa batek datu-sarreren kopuru infinitua har dezakeela. Horregatik, konputazio-egoerak erabiltzea ez da nahikoa programa baten espezifikazio orokorra adierazteko.

Gabezia horri aurre egin eta adierazpide orokorrago baten bila, *asertzioak* erabiltzen dira. Azken batean, asertzioak espresio logikoak dira, prometan iruzkinen modura txertatzen direnak eta programako aldagaiek puntu jakin batean betetzen dituzten propietateak adierazten dituztenak. Hortaz, asertzio bakoitzak programaren puntu batean gerta daitezkeen egoeren multzoa (infinitua izan daitekeena) errepresentatzen du. Asertzioa egiazko egiten duten egoerak dira puntu horretan errepresentatzen direnak. Egoera horiek *asertzioa betetzen* dutela esango dugu. Adibidez,

$$\phi \equiv \{\ x = y \times z + h\ \}$$

asertzioa hartuz gero, asertzio hori:

- $\{\ x = 31\ \wedge\ y = 7\ \wedge\ z = 4\ \wedge\ h = 3\ \}$ egoeran *egiazkoa* da (eta egoera horrek asertzio hori betetzen du).
- $\{\ x = 31\ \wedge\ y = 7\ \wedge\ z = 4\ \wedge\ h = 0\ \}$ egoeran *faltsua* da.
- $\{\ x = 31\ \wedge\ y = 7\ \}$ egoeran *zehaztugabea* da.

Asertzioen bidez, programa baten espezifikazioa adieraz daiteke. Adibidez, bektore bateko elementuen batura kalkulatzen duen hurrengo programarekin egiten den bezalaxe.

```
begin  { n ≥ 1 }
   i := 0;
   x := 0;
   while i < n loop  { x = Σ_{j=1}^{i} A(j) }
      i := i+1;
      x := x+A(i);
   end loop;
end ; { x = Σ_{j=1}^{n} A(j) }
```

Asertzioak adierazteko, lengoaia formala erabiliko dugu, lehen mailako logika, hain zuzen ere. Hau da, asertzio bakoitza lehen mailako logikako formula bat izango da.

Lehen mailako logika (*predikatu-logika* eta *predikatu-kalkulua* izenak ere hartzen ditu) lehen mailako lengoaien inferentzia aztertzeko sistema formal bat da. Lehen mailako lengoaiak sinbolo (aldagai eta konstante) indibidualak erabiltzen dituzten funtzioez eta predikatuez eta, aldi berean, aldagai indibidualen kuantifikatzaileez osatutako lengoaia formalak dira. Oro har, guk erabiliko ditugun indibiduoak zenbaki arruntak izango dira. Hau da, formulek zenbaki arruntei buruzko propietateak edo erlazioak adieraziko dituzte, edo, bestela esanda, gure domeinua $\mathbb{N}$ da. Salbuespen gisa, batzuetan karaktereak (komatxo artean adierazitakoak) ere erabiliko ditugu, *string*-ekin lan egiteko.

2.2.1 Asertzioak idazteko lehen mailako logikaren sintaxia

Lengoaia formal guztiek sintaxia eta semantika jakin bat dute. Asertzioen idazkeran erabiliko dugun lengoaiaren sintaxiari dagokionean, formulak idazteko alfabetoa ADA programazio-lengoaiari egokituko diogu eta, besteak beste, honako sinbolo hauek erabiliko ditugu:

- Aldagai-sinboloak: x, y, z, ...
- Konstante-sinboloak: 0, 1, ..., a, b, c, ...
- Funtzio-sinboloak: $+$, $-$, mod, ..., $\sum$, $\prod$, ...
- Predikatu-sinboloak: $<$, $=$, $>$, ..., *bikoiti*, *bakoiti*, ...

Aldagai-, konstante- eta funtzio-sinboloak indibiduoak adierazteko erabiltzen ditugu, *terminoak* osatuz. Predikatuak baliatzen dira formulak osatzeko (konektatzaileekin eta kuantifikatzaileekin konbinatuz). Hau da, *terminoek* indibiduoak adierazten dituzte, eta formulek indibiduoen arteko propietateak. *Terminoak* errekurtsiboki defini daitezke:

- Termino sinpleak: aldagaiak eta konstanteak (funtzio 0-tarrak). Bektoreetako elementu bakoitza ere termino sinple bat da.
- Termino konposatuak: funtzio n-tarrak n terminori aplikatuta ($n > 0$).

Beste edozein adierazpen ez da terminoa izango. Adibidez,

$$x,\ 0,\ 1,\ a,\ true,\ false,\ A(i)$$

termino sinpleak dira, eta

$$x + 1,\ x\ mod\ 2,\ \sum_{i=1}^{j} A(i)$$

termino konposatuak dira. Adibideetan ikus daitekeenez, bektoreko osagaiak ere terminoak dira. Definizioak sinplifikatzeko, bektore bakoitza aldagai multzo baten laburduratzat hartuko dugu eta, horregatik, bektore bateko osagai bakoitza aldagai arrunt bezala erabil daiteke.

Era berean, formulak ere errekurtsiboki defini daitezke:

- Formula sinpleak edo atomoak: predikatu n-tarrak n terminori aplikatuta. *true* eta $false$ atomoak dira.
- Formula konposatuak: $\neg$ (negazioa, ukapena), $\wedge$ (konjuntzioa, eta), $\vee$ (disjuntzioa, edo), $\rightarrow$ (inplikazioa, baldintza) eta $\leftrightarrow$ (baliokidetza, baldintza bikoitza) eragile logikoak, eta $\forall$ (unibertsala) eta $\exists$ (existentziala) kuantifikatzaileak erabiltzen dira.

Ukapena konektatzaile bakuna da (formula bakar bati aplikatzen zaio), eta konjuntzioa, disjuntzioa, baldintza eta baldintza bikoitza bitarrak (bi formulari aplikatzen zaie) dira. Esate baterako, hurrengo espresioek

$$x < y,\ z = 2, A(i) > A(j) + 1,\ bakoiti(x),\ bikoiti(2)$$

atomoak adierazten dituzte, eta

$$\begin{gathered}\neg(\ x = y\ ) \\ x > 0\ \wedge\ x \leq 10 \\ y = 1\ \vee\ x = 5\ \rightarrow\ \neg bakoiti(x) \\ \forall z\ (\ bakoiti(z)\ \vee\ bikoiti(z)\ )\end{gathered}$$

formula konposatuak dira. Formula konposatuetan, aldagai bat *atxikia* da kuantifikatzaile baten eraginpean badago. Atxikiak ez diren aldagaiak *libreak* dira. Kontuan hartu *true* eta $false$ konstanteak bai termino bai atomotzat hartuko ditugula.

2.2.2 Asertzioak idazteko lehen mailako logikaren semantika

Lehen mailako logikaren semantika estandarra erabiliko dugu $\mathbb{N}$ domeinuarekin. Batetik, konektatzaileari buruz, ukapenaren interpretazioa $\neg true \equiv false$ eta $\neg false \equiv true$ da, eta konektatzaile bitarren egia-taula honako hau da:

ϕ	ψ	$\phi \wedge \psi$	$\phi \vee \psi$	$\phi \rightarrow \psi$	$\phi \leftrightarrow \psi$
$false$	$false$	$false$	$false$	$true$	$true$
$false$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$true$	$true$	$false$	$false$

Bestetik, kuantifikatzaileei buruz, $\forall x\ \phi$ egiazkoa da ϕ betetzen bada edozein $x \in \mathbb{N}$ baliotarako, eta $\exists y\ \psi$ egiazkoa da ψ betetzen bada $y \in \mathbb{N}$ balioren baterako.

Hala ere, askotan formula batek adierazi behar duen propietatea ez dute $\mathbb{N}$ multzoko elementu guztiek betetzen, baizik eta $\mathbb{N}$-ren azpi-multzo batek. Horrelakoetan, aldagai atxikiei esleitu ahal zaizkien balioak murriztu egin behar dira. Horretarako, patroi edo egitura ezagun bat erabiltzen dugu. Lehenik, x aldagai atxikiak har ditzakeen balioak adierazten dira formula baten bidez. Formula horrek *domeinu* izena hartzen

du eta $D(x)$ adierazpenarekin errepresentatzen dugu. Normalean, formula hau sinplea da (atomo bat), baina formula konposatuak ere erabil daitezke. Adibidez:

$$1 \leq x \leq 10$$
$$x > 10$$
$$1 < x \leq 100 \ \wedge \ x \neq 50$$

Bigarrenik, domeinuko elementuek bete behar duten propietatea adierazten da. Erabiltzen den formulak *propietate* izena hartzen du eta $P(x)$ adierazpenarekin errepresentatzen dugu. Bi elementu hauekin osatzen den formulak honako egitura hau du

$$D(x) \otimes P(x)$$

non $\otimes$-k eragile logiko bat errepresentatzen duen kuantifikatzailearen arabera. Kuantifikatzaile unibertsalarekin baldintza ($\rightarrow$) erabiltzen da, eta kuantifikatzaile existentzialarekin konjuntzioa ($\wedge$):

$$\forall x \ (\ D(x) \ \rightarrow \ P(x) \ )$$
$$\exists x \ (\ D(x) \ \wedge \ P(x) \ )$$

Edozein formuletan kuantifikatzaileren bat agertzen denean, aurreko patroiak erabiltzea derrigorrezkoa da.

Kasu berezi gisa, aldagai atxikiak edozein $\mathbb{N}$ multzoko elementu har badezake (domeinu unibertsala), orduan $D(x) \equiv true$ da eta patroiak horrela sinplifika daitezke: $\forall x \ P(x)$ eta $\exists x \ P(x)$.

Era berean, aldagai atxikiak baliorik hartu ezin badu (domeinu hutsa), orduan $D(x) \equiv false$ da eta formula osoa sinplifikatzen da:

$$\begin{array}{rcl} \forall x \ (\ false \ \rightarrow \ P(x) \ ) & \equiv & true \\ \exists x \ (\ false \ \wedge \ P(x) \ ) & \equiv & false \end{array}$$

Hau da, horrelakoetan, kuantifikatzaile unibertsala ($\forall$) duen formula *true* bihurtzen da (konjuntzioaren elementu neutroa), eta kuantifikatzaile existentziala ($\exists$) duen formula, $false$ (disjuntzioaren elementu neutroa).

Izatez, kuantifikatzaileak laburduratzat har daitezke, errepikatzen diren eragiketak orokortu eta biltzeko balio baitute. Horrela ikusita, kuantifikatzaile unibertsalak konjuntzioa orokortzen du, eta kuantifikatzaile existentzialak, disjuntzioa:

Kuantifikatzailea	Eragiketa orokortua
$\forall$	$\wedge$
$\exists$	$\vee$

Adibidez:

$$\begin{array}{rcl} \forall x \ (\ 1 \leq x < 4 \ \rightarrow \ A(x) < 8 \ ) & \equiv & A(1) < 8 \ \wedge \ A(2) < 8 \ \wedge \ A(3) < 8 \\ \exists y \ (\ 2 < y \leq 5 \ \wedge \ B(y) \geq 1 \ ) & \equiv & B(3) \geq 1 \ \vee \ B(4) \geq 1 \ \vee \ B(5) \geq 1 \end{array}$$

Horrela, kuantifikatzaileak erabiliz, propietate batzuk era labur batean adieraz daitezke:

- *"*$A(1..10)$ *zenbaki osoko positiboen bektorea da"*:

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

- *"$A(1..10)$-k zehatz-mehatz zenbaki negatibo bat du"*:

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

- *"Badago zenbaki bat 5-en zatigarria dena"*:

$$\exists x \; (\; x \; mod \; 5 = 0 \;) \qquad (\; \mathbb{N} \text{ domeinuarekin: } D(x) \equiv true \;)$$

- *"Edozein zenbaki bikoiti zenbaki ez-lehena da"*:

$$\forall x \; (\; x \; mod \; 2 = 0 \; \rightarrow \; \neg lehena(x) \;)$$
$$\text{non } lehena(x) \equiv \forall i \; (\; 1 < i < x \; \rightarrow \; x \; mod \; i \neq 0 \;)$$

Kuantifikatzailearen bidez formulak laburtzen diren antzera, terminoak ere laburtu daitezke. Horretarako, funtzio-sinbolo hauek erabiltzen dira: $\sum$ (batukaria), $\prod$ (biderkaria) eta $\mathcal{N}$ (kontakaria).

Batukarien eta biderkarien idazkera berdina da. Lehenik, aldagai atxikiaren domeinua espezifikatzen da eta, bigarrenik, aldagai atxikia erabiltzen duen termino bat adierazten da:

$$\begin{array}{rcl} \sum_{x=i}^{j} f(x) & \equiv & f(i) + f(i+1) + \ldots + f(j-1) + f(j) \\ \prod_{x=i}^{j} f(x) & \equiv & f(i) \times f(i+1) \times \ldots \times f(j-1) \times f(j) \end{array}$$

Honakoetan ere badira nabarmendu beharreko kasu partikularrak. Domeinua hutsa denean, batukariak 0 balioa hartzen du (batuketaren elementu neutroa), eta biderkariak, 1 (biderketaren elementu neutroa).

Hirugarrenak, kontakariak, adierazitako propietatea betetzen duten domeinuko elementuen kopurua itzultzen du. Kasu honetan, kuantifikatzaile existentzialarekin gertatzen den bezalaxe, $D(x) \wedge P(x)$ patroia erabiltzen da, baina zenbaki arrunt bat bueltatzen du. Adibidez:

$$\mathcal{N}x \; (\; 1 \leq x \leq 7 \; \wedge \; bikoiti(x) \;) \; = \; 3$$

Gainera, batukarian bezalaxe, domeinua hutsa denean, kontakariak 0 bueltatzen du (batuketaren elementu neutroa).

Ondoren, batukaria, biderkaria eta kontakaria erabiliz adieraziko ditugu propietate batzuk, adibide gisa:

- *"$A(1..10)$ bektoreko elementu guztien batura 50 baino handiago da"*:

$$(\sum_{i=1}^{10} A(i) \;) \; > \; 50$$

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

$$(\prod_{i=1}^{k} A(i) \;) \; = \; x$$

- *"3 elementua 5 aldiz agertzen da $A(1..10)$ bektorean"*:

$$\mathcal{N}i \; (\; 1 \leq i \leq 10 \; \wedge \; A(i) = 3 \;) \; = \; 5$$

2.2.3 Logikaren interpretazioa programen testuinguruan

Testuinguru honetan, konputazio-egoerek adieraziko dute formula bateko aldagai libreen balioa. Horregatik, konputazio-egoera konkretu batean, formula batek balioa (*egiazkoa* edo *faltsua*) duela esango dugu (edo *definituta* dagoela), baldin eta soilik baldin bere aldagai libre guztiek balio bat badute egoera horretan. Adibidez, konputazio-egoera $\{ x = 3 \ \wedge \ y = 4 \}$ bada:

- $x \geq y$ faltsua da.
- $x + 1 = y \ \wedge \ y > 2$ egiazkoa da.
- $(x \ mod \ 2 = 0) \ \leftrightarrow \ (y \ mod \ 2 = 0)$ faltsua da.
- $\exists z \ (\ z > 0 \ \wedge \ z + x = y \ )$ egiazkoa da.
- $\forall z \ (\ z > 1 \ \rightarrow \ z + x > y \ )$ egiazkoa da.

Gainera, $A = [\ 2, \ 5, \ 3, \ 6, \ 1, \ 8 \ ]$ bektorea dugularik:

- $\exists i \ (\ 1 \leq i \leq 6 \ \wedge \ (\ x = A(i) \ \vee \ y = A(i) \ ) \ )$ egiazkoa da.
- $\forall i \ (\ 1 \leq i \leq 6 \ \rightarrow \ A(i) \leq x + y \ )$ faltsua da.
- $\exists i \ (\ 1 \leq i \leq 6 \ \wedge \ x = A(i) \ ) \ \wedge \ \exists j \ (\ 1 \leq j \leq 6 \ \wedge \ y = A(j) \ )$ faltsua da.
- $\exists i \ (\ 1 \leq i \leq 6 \ \wedge \ \forall j \ (\ i < j \leq 6 \ \rightarrow \ A(i) > A(j) \ ) \ )$ egiazkoa da.

Programa bateko asertzioak idazterakoan, badira kontuan hartu beharreko arau eta murriztapen batzuk. Hona hemen murriztapen nagusiak:

- **Programakoak diren aldagai libreen eta programakoak ez diren aldagai atxikien (laguntzaileen) erabilera asertzioetan**.

  Asertzioetan aldagai libreak eta atxikiak ager daitezke, lehen ere esan dugunez. Aldagai libre eta atxikien erabilera estuki lotuta dago programako aldagaiekin, eta erabilera okerrak erabat aldaraz dezake formulen esanahia.

  Kontuan hartu behar da, batetik, asertzio batek aldagai libreak baldin baditu eta aldagai libre horiek ez badira programan agertzen, asertzio horrek ez duela zentzurik izango: programaren konputazio-egoerarik ez du definitzen eta, beraz, zehaztugabea da.

  Bestetik, asertzio batek aldagai atxikiak baldin baditu eta aldagai atxiki horiek programan agertzen badira, asertzio horrek ez du zentzurik izango. Identifikadoreen talka saihestu egin behar da. Ez da erabili behar aldagai atxiki batentzat programan agertzen den aldagai baten izena.

  Arau garrantzitsua da honako hau: programan agertzen den edozein aldagaik librea izan behar du asertzioetan. Eta, bestalde, ez dira inoiz nahastu behar programako aldagaien eta aldagai atxikien identifikadoreak.

  Adibidez, ekar dezagun hona bektore bateko elementuen batura kalkulatzen duen programa:

```
begin  { n ≥ 1 }
   i := 0;
   x := 0;
   while i < n loop  { x = Σ_{j=1}^{i} A(j) }
      i := i+1;
      x := x+A(i);
   end loop;
end ; { x = Σ_{j=1}^{n} A(j) }
```

Nabarmentzekoa da asertzioetan erabiltzen diren aldagai libre guztiak ($A(1..n)$, n, i, x) programan erabiltzen diren aldagaiak direla. Eta, bestetik, batukarian erabiltzen den j aldagai atxikia ez da ageri programan. Zentzua galduko lukete asertzioek batukariko aldagai atxiki moduan, adibidez, i edo x identifikadoreak erabili izan balira.

- **Formula bat ondo definituta dago konputazio-egoera batean baldin eta soilik baldin:**
  - **bere aldagai libre guztiek balio bat badute egoera horretan, eta**
  - **formulako espresio guztien ebaluazioek balio definitua itzultzen badute egoera horretan.**

  Aldagai libreek balioren bat hartu ondoren agertu behar dute asertzioetan. Balio hori datu-sarreratik datorkiena izan daiteke, edo aldagaiak hasieratzetik datorkiena.

  Aurreko programara joz berriz, aldagai libre guztiek ($A(1..n)$, n, i, x) asertzioetan agertzen direnerako badute balioren bat. Ez luke zentzurik izango, esate baterako, hasierako asertzioa $\{\ n \geq 1 \wedge\ i = 0 \wedge\ x = 0\ \}$ izateak, artean i eta x aldagaiak hasieratu gabe daude-eta.

***def* predikatua**

Terminoak eta formulak ondo definituta egotea ezinbesteko baldintza da programen konputazio-egoerari buruzko propietateak adierazi ahal izateko. Hori dela eta, hemendik aurrera, asertzioetan erabiltzen ditugun terminoak eta formulak ondo definituta daudela adierazteko *def* predikatua erabiliko dugu.

def predikatuak terminoak edo formulak har ditzake argumentu gisa, eta egiazko balioa itzuliko du baldin eta soilik baldin argumentua (terminoa edo formula) ondo definituta badago konputazio-egoera batean.

2.2.4 Arrazonamendua

Esana dugu programak exekutatzen direnean aldagaien balioak aldatzen direla esleipenen bidez. Egoera-aldaketa horiek asertzioetan jasotzen dira: asertzio batek aldaketa aurreko egoera adieraziko du, eta beste batek aldaketa ondorengo egoera. Asertzio batetik besterako trantsizio horiek, azken batean, logikan oinarritutako arrazonamendua dute oinarrian. Komeni zaigu, bada, modu sendoan egitea aldaketak eta horien inguruko arrazoiketak. Zeregin horretan erabiliko dugun oinarrizko eragiketa *ordezpena* da. Funtsean, formula bateko aldagaien ordez terminoak jartzea da *ordezpenak* egitea.

Hona hemen *ordezpena*ren definizio formala: izan bitez ϕ formula, x aldagaia eta t terminoa, ϕ_x^t adierazpenak ϕ formulako x-ren agerpen libre guztien ordez t terminoa jarrita lortzen den formula errepresentatzen du. Transformazio sinple hori egitean bada kontuan hartu beharreko murriztapen bat: t terminoak ezin du eduki ϕ formulan kuantifikatuta (atxikia) agertzen den aldagairik. Adibidez:

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

ez da zuzena.

Ondoren ikusiko dugu nola sinplifikatzen diren ordezpen batzuk:

- $(\ x^3 \geq 0 \ \wedge \ x + z < 5 \ )_x^{A(i)} \ \equiv \ (\ A(i)^3 \geq 0 \ \wedge \ A(i) + z < 5 \ )$
- $(\ \prod_{i=1}^{k} A(i) \ )_k^8 \ \equiv \ \prod_{i=1}^{8} A(i)$
- $(\ \exists i \ (\ i > x \ \wedge \ bikoiti(i) \ ) \ )_x^{x+y} \ \equiv \ \exists i \ (\ i > x + y \ \wedge \ bikoiti(i) \ )$

Ordezpenaz gain, arrazonatzean garrantzi handikoa da formulen arteko *ahulago-gogorrago* erlazioa. Formalki, ϕ formula ψ baino gogorragoa da (edo ψ ϕ baino ahulagoa) baldin eta soilik baldin

$$\{ \ s \mid s(\phi) = true \ \} \ \subseteq \{ \ s \mid s(\psi) = true \ \}$$

non $\{ \ s \mid s(\phi) = true \ \}$ multzoa den ϕ formulak egiazko egiten dituen egoeren multzoa, eta era berean $\{ \ s \mid s(\psi) = true \ \}$ multzoa ψ formulak errepresentatzen duen egoeren multzoa.

Adibidez, $\phi \equiv lehena(x) \ \wedge \ x > 3$ formula $\psi \equiv \neg(x \ mod \ 3 = 0)$ baino gogorragoa da:

$$\begin{array}{rcl} \{ \ s \mid s(\phi) = true \ \} & = & \{ \ \{x = 5\}, \ \{x = 7\}, \ \ldots \ \} \\ & & \ \ \ \ \ \ |\bigcap \\ \{ \ s \mid s(\psi) = true \ \} & = & \{ \ \{x = 1\}, \ \{x = 2\}, \ \{x = 4\}, \ \{x = 5\}, \ \{x = 7\}, \ \ldots \ \} \end{array}$$

Definizio horretatik zenbait propietate interesgarri ondoriozta daitezke:

- $\phi \ \rightarrow \ \psi$ betetzen da baldin eta soilik baldin ϕ gogorragoa bada ψ baino (edo, beste era batean esanda, ψ ahulagoa bada ϕ baino).
- $true$ da formularik ahulena (egiazkoa baita edozein s egoeratan) eta $false$ da gogorrena (faltsua da-eta edozein s egoeratan).

  Hortaz, betetzen da $\phi \rightarrow true$ eta $false \ \rightarrow \phi$ edozein ϕ formulatarako.
- $\phi \ \rightarrow \ (\ \phi \vee \psi \ )$
- $(\ \phi \wedge \psi \ ) \ \rightarrow \ \phi$
- Edozein ϕ-tarako, ϕ_x^t formula gogorragoa da $\exists x \ \phi$ baino, eta ahulagoa $\forall x \ \phi$ baino.

Adibide batzuen bidez ilustratuko dugu formula ahulago eta gogorragoen kontzeptua:

- $(\ x > 0 \ )$ gogorragoa da $(\ x \geq 0 \ )$ baino, zeren $(\ x \geq 0 \ ) \ \equiv \ (\ x > 0 \vee x = 0 \ )$ (gogoratu $\phi \ \rightarrow \ (\ \phi \vee \psi \ )$ betetzen dela).
- $(\ x = 0 \wedge y = 1 \ )$ gogorragoa da $y = z^x$ baino.

- ($k = 0$) eta ($k = 1 \wedge A(1) = 0$) formulak gogorragoak dira

  $$\forall j \; (\; 1 \leq j \leq k \; \rightarrow \; A(j) = 0 \;)$$

  baino.

  $k = 0$ denean, kuantifikatzaile unibertsalaren domeinua hutsa da eta, ondorioz, kuantifikazioaren balioa *true* da. Gogoratu: $(k = 0) \rightarrow true$

- $lehena(2)$ formula gogorragoa da $\exists x \; (\; lehena(x) \;)$ baino.

- ($5 \leq n \wedge A(5) > 0$) eta ($1 \leq i \leq n \wedge A(i) > 0$) formulak gogorragoak dira $\exists j \; (\; 1 \leq j \leq n \; \wedge \; A(j) > 0 \;)$ baino.

- $\forall i \; (\; 1 \leq i \leq n \; \rightarrow \; bikoiti(A(i)) \;)$ formula gogorragoa da

  $$(\; 7 \leq n \; \rightarrow \; bikoiti(A(7)) \;)$$

  baino.

- $\forall j \; (\; 1 \leq j \leq n \; \rightarrow \; A(j) > 0 \;)$ formula gogorragoa da $(\; 1 \leq i \leq n \; \rightarrow \; A(i) > 0 \;)$ baino.

2.3 Aurre-ondoetako espezifikazio formala

Aurreko atalean azaldu dugu zein lengoaia erabiliko dugun espezifikazioak modu formalean adierazteko. Honako honetan, espezifikazioaren formatua deskribatuko dugu.

Irakasgai honetan, *aurre-ondoetako* formatua erabiliko dugu espezifikazioetarako. Formatu hau erabiliz, programak horrela espezifikatzen dira

$$\{ \; \Phi \; \} \; \mathtt{S} \; \{ \; \Psi \; \}$$

non S programa den, eta Φ eta Ψ asertzioak diren. Φ asertzioari *aurreko baldintza* (edo *aurre-baldintza*) deitzen zaio, eta Ψ-ri *ondoko baldintza* (edo *post-baldintza*). Kontuan hartu behar da programa programazio-lengoaian idatzita dagoela, eta asertzioak, aldiz, logika-lengoaian. Gure kasuan, programazio-lengoaia ADA da eta asertzioen lengoaia, lehen mailako logikaren lengoaia.

Esanahiaren aldetik, aurreko espezifikazioak adierazten du Φ betetzen den egoera batean hasten bada S programa, orduan Ψ betetzen den egoera batean bukatzen dela.

Aurreko baldintzak adierazten du zein egoeratan abia daitekeen S programa. Horregatik, komeni da ahalik eta formula *ahulena* definitzea, nolabait, datuen *baldintza minimoak* programaren exekuzioa zuzena izan dadin.

Aitzitik, post-baldintzak adierazten du datuen eta emaitzen arteko erlazioa eta, horregatik, formularik *gogorrena* definitu behar da, ahalik eta modurik zorrotzenean esateko zein diren emaitza onargarriak sarreren arabera.

Adibidez, zatidura eta hondar osoa kalkulatzen duen programaren aurre-ondoetako espezifikazioa honako hau izan daiteke:

Aurre $\equiv \{\ y > 0\ \wedge\ x \geq 0\ \}$

```
begin
   z := 0;
   h := x;
   while h >= y loop
      z := z+1;
      h := h-y;
   end loop;
end ;
```

Post $\equiv \{\ x = z \times y + h\ \wedge\ y > h \geq 0\ \wedge\ z \geq 0\ \}$

Aurre-ondoetako asertzio horiek espezifikazio egokia osatzen dute. Aurreko baldintza aski ahula da datu onargarri guztiak onartzeko. Post-baldintza aski gogorra da onartezinak liratekeen emaitzak bazter uzteko.

Aurre-baldintza adierazteko ($y > 1 \wedge x \geq 0$) erabili izan bagenu, gogorregia izango litzateke, programak 1 zenbakiaren zatiketa osoa ere kalkulatzen duelako. Bestetik, ($\neg y = 0\ \wedge\ x \geq 0$) erabili izan bagenu, ahulegia izango litzateke, aurre-baldintzak onartu arren programak ez baitu ondo kalkulatzen zenbaki negatiboen zatidura.

Era berean, post-baldintzari dagokionez, ($x = z \times y + h$) formula ahulegia izango litzateke. Erraz samar topa dezakegu egoera bat post-baldintza beteko lukeena baina zatidura onargarritzat har ezin daitekeena. Esate baterako:

$$\{\ x = 31\ \wedge\ y = 7\ \wedge\ z = 3\ \wedge\ h = 10\ \}$$

Bestetik, ($x = z \times y + h\ \wedge\ h = y - 1$) gogorregia izango litzateke; kasu honetan badaude egoera batzuk ($\{\ x = 31 \wedge y = 7 \wedge z = 3 \wedge h = 10\ \}$, adibidez) post-baldintza beteko ez luketenak, baina emaitza egokiak direnak.