3

Programen egiaztapena

Egindako programa bat zuzena den ala ez jakitea programatzaileon zeregin garrantzitsuenetakoa da. Programa baten garapena ezin da bukatutzat eman zuzena dela egiaztatu aurretik. Hori horrela, programazioaren bilakaeran era askotako teknikak baliatu izan dira zuzentasunaren egiaztapena helburu hartuta.

Bi sail nagusitan banatu daitezke teknika horiek: batzuk probatan oinarritzen dira, eta beste batzuk metodo formal-matematikoetan.

Probatan oinarritutako teknikek datuen kasuistika zabal bat definitu eta horien arabera aztertzen dute programen portaera. Egiaztapenaren kalitatea diseinatutako proba-bankuaren estalduraren araberakoa da. Hala ere, kontuan hartu behar da E. W. Dijkstra-k esandakoa, hots, probek programak erroreak badituela froga dezakete, ez ordea errore gabea denik.

Eredu formal-matematikoetan oinarritutako metodoek konplexutasun formala dute arazo nagusi. Sarri gertatzen da programa egitea bezain konplexua dela programa horren zuzentasuna frogatzea.

Kapitulu honetan egiaztapen formalerako teknika batean barneratuko gara. Jakitun gara metodo formal honen aplikazioak konplexutasuna erants diezaiokeela programan garapenari, baina, hala eta guztiz ere, uste dugu programei buruzko arrazonamendu formalean trebatzea oso ekarpen aberatsa dela.

3.1 Programen zuzentasuna eta sistema formalak: Hoare-ren sistema formala

Programen zuzentasuna egiaztatu ahal izateko, lehenik eta behin, programen *zuzentasuna* zer den definitu behar dugu. Horretarako, aurreko kapituluan ikusitako *aurre-ondoetako espezifikazioa*ren kontzeptuan oinarrituko gara.

Zuzentasuna espezifikazioari lotuta dago. Programa bat zuzena da, ala ez, beti espezifikazio batekiko. Bi plano bereizten dira: zer espero den programaren exekuzioaz (espezifikazioa), eta benetan zer egiten duen programa horrek. Biak bat badatoz, programa zuzena da; espero zena eta benetan gertatzen dena berdinak ez badira, programa ez da zuzena.

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

Izan bitez S programa eta (Φ, Ψ) espezifikazioa, non Φ aurre-baldintza den eta Ψ post-baldintza.

Esango dugu S programa *partzialki zuzena* dela (Φ, Ψ) espezifikazioarekiko

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

baldin eta soilik baldin Φ betetzen den egoera batean hasten den Sren edozein konputazio, bukatzen bada, Ψ betetzen den egoera batean bukatzen bada.

Zuzentasun partzialak ez du aintzat hartzen programa bukatzen den ala ez. Bukaeraren alderdia ere tratatzeko, *Buka* predikatua definituko dugu:

$$Buka(S, \Phi)$$

espresioak adierazten du Φ betetzen den egoera batean hasten den Sren edozein konputazio pauso kopuru finituan *bukatzen* dela. Ohartzekoa da *Buka* predikatuak bi argumentu dituela: programa bera eta aurreko baldintza. Aurreko baldintza erabakigarria da, izan ere, Φ finkatu ezean ezin baita baieztatu programa bat bukatuko den ala ez.

Eta, azkenik, S programa *osoki zuzena* da (Φ, Ψ) espezifikazioarekiko

$$\{\ \Phi\ \}\ [\ \mathtt{S}\ ]\ \{\ \Psi\ \}$$

baldin eta soilik baldin Φ betetzen den egoera batean hasten den Sren edozein konputazio Ψ betetzen den egoera batean bukatzen bada. Hau da, programa bat osoki zuzena da espezifikazio batekiko partzialki zuzena izateaz gain beti bukatzen bada:

$$\{\ \Phi\ \}\ [\ \mathtt{S}\ ]\ \{\ \Psi\ \} \equiv \{\ \Phi\ \}\ \mathtt{S}\ \{\ \Psi\ \} + Buka(S, \Phi)$$

Zuzentasunaren definizio formal honetatik abiatuta, programa baten zuzentasuna frogatzeko sistema formalak erabiltzen dira. Orokorrean, sistema formal batean honako osagaiak definitzen dira:

- Axiomak: egiazkoak diren oinarrizko propietateak.
- Inferentzi erregelak: nola deduzitu propietate batzuk besteetatik. Esate baterako, P propietatea $P_1,\ P_2, \ldots, P_n$ propietateetatik deduzitzen dela honela adierazten da:

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

  Marraren gainean dauden $P_1,\ P_2, \ldots, P_n$ propietateak premisak dira, eta P da ondorioztatzen den propietatea. Alegia, inferentzi erregelak dio $P_1,\ P_2, \ldots, P_n$ propietateak betetzen direnean, sistema formalean P propietatea ere betetzen dela.

Sistema formal batean, *frogapenak* propietateen segidak dira

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

non P_k frogatu nahi dugun propietatea den eta P_i ($1 \leq i \leq k$) bakoitza den

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

Programen egiaztapenerako sistema formal bat eratu zuen Hoare-k, eta sistema hori landuko dugu irakasgai honetan. Sistema formal horretan $\{\ \Phi\ \}\ \mathtt{S}\ \{\ \Psi\ \}$ motako propietateak edo baieztapenak frogatzen dira, hau da, zuzentasun partzialari buruzko baieztapenak. *Hoare-ren hirukote* ere deitzen zaie baieztapen horiei.

Zuzentasun partziala erabaki-problema ez da osoa. Hau da, sistema formala bat hartuta, zuzentasun partzialari buruzko baieztapen zuzen guztiak ez dira frogagarriak. Hala ere, gerta daiteke sistema formala jakin bat erabiliz frogagarria ez den baieztapen zuzen bat frogagarria izatea beste sistema formala batean. Adibidez, honako baieztapen hau froga daiteke Hoare-ren sistema formala erabiliz

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

baina beste hau

$\{\ true\ \}$
```
   lag := x;
   x := y;
   y := lag;
```
$\{\ x = y\ \}$

ez da frogagarria Hoare-ren sistema formalean.

Kapitulu honetako hurrengo ataletan Hoare-ren sistema formala deskribatuko dugu. Horretarako sistema osatzen duten oinarrizko axiomak eta inferentzi erregelak definituko ditugu, eta beren erabilera landuko dugu programen egiaztapenean.

3.2 Asignazioa

Asignazioak dira edozein programazio-lengoaia agintzaileko agindu mota oinarrizkoena. Paradigma agintzailean, aldagaien balio-aldaketa asignazioen bidez gauzatzen da. Oro har, asignazio-agindu batek termino baten balioa aldagai bati esleitzen dio, aurretik aldagaiak zuen balioa ezabatuz. ADA programazio-lengoaian horrela adierazten ditugu asignazioak:

```
x := t;
```

non x aldagaia den eta t, terminoa. Gogora dezagun terminoak sinpleak edo konposatuak izan daitezkeela. Zehazki, t izan daiteke aldagai bat, konstante bat, edo eragiketak eta eragigaiak konposatuz osatutako espresio bat.

Hoare-ren sistema formalean asignazioekin lotutako propietateak edo baieztapenak frogatzeko *Asignazioaren Axioma* erabili behar dugu. Formalki, asignazioaren axioma horrela definitzen da:

(AA) $\boxed{\{\ def(t)\ \wedge\ \varphi_x^t\ \}\ \texttt{x := t;}\ \{\ \varphi\ \}}$

Izan bedi `x := t;` agindua, asignazioaren axiomak adierazten du φ_x^t betetzen den egoera batean hasten den edozein konputazio φ betetzen den egoera batean bukatzen dela baldin eta t terminoa *definituta* badago. Beste hitz batzuekin esanda: aurreko baldintzan t definituta egonez gero, t terminoak aurreko baldintzan betetzen dituen propietateak betetzen ditu x aldagaiak ondoko baldintzan.

(AA)ren formulazioa eskema orokor bat da. Hau da, edozein x aldagai, t termino eta φ formula hartuz, eskema horren instantziak edo formulazio konkretuak egin daitezke, eta horrela osatutako adierazpen konkretuak dira, izatez, asignazioaren axiomak. Adibidez,

$\{\ z \neq 0\ \wedge\ w/z = 10\ \}$
```
   y := 10;
```
$\{\ w/z = y\ \}$

hirukotea asignazioaren axioma bat da.

t terminoa definituta egoteak esan nahi du haren ebaluazioak ez duela errorerik sortuko. Kontzeptu hau 2. kapituluan definitu da. Orokorrean, gure testuinguru honetan suposatuko dugu edozein terminoren ebaluazioak ez duela errorerik sortzen, bi salbuespenezko kasuetan izan ezik. Bata, $s/0$ (zati zero) motako adierazpenek errorea sortzen dute. Bestea, $A(1..n)$ bektorea emanda, $A(i)$ motako adierazpenek errorea sortzen dute baldin eta $i < 1$ edo $i > n$ bada. Gainera, erroreak barreiatu egiten dira. Hau da, t termino konposatuaren azpi-termino batek errorea sortzen badu, orduan t terminoak ere errorea sortzen du.

$def(t)$ formulak t terminoa definituta egoteko baldintza minimoa adierazten du; hau da, zein baldintzatan ez duen errorerik sortuko t terminoaren ebaluazioak.

Adibidez, aurreko bi salbuespenak kontuan hartuta, 10 terminoa beti definituta dago

$$def(10) \equiv true$$

eta $10/x$ definituta dago baldin eta x-ren balioa 0 ez bada:

$$def(10/x) \equiv (x \neq 0)$$

Oro har, termino sinple guztiak beti definituta daude, bai aldagaiak bai konstanteak.

Esandakoen arabera, hurrengo baieztapena froga daiteke, **(AA)** erabiliz:

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

non $\varphi \equiv (x > 0)$, $\varphi_x^{A(i)} \equiv (A(i) > 0)$ eta $def(A(i)) \equiv (1 \leq i \leq n)$.

Asignazioaren axiomak aukera ematen digu asignazioen portaera modelizatzeko eta asignazioen eraginari buruz arrazonatzeko. Hala ere, asignazioei buruz egiten diren baieztapen guztiak ezin dira **(AA)** erabiliz frogatu, nahiz eta zuzenak izan. Esate baterako, honako baieztapena

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

frogagarria da **(AA)** baizik erabili gabe. Baina beste hau

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

frogatzeko ez da nahikoa **(AA)** erabiltzea, nahiz eta baieztapen zuzena den. Kontua da definitutako aurre-baldintza gogorregia dela **(AA)** erabiltzeko, eta beste aurre-baldintza ahulago bat beharko genukeela. Gerta daiteke baita ere post-baldintza ahulegia izatea eta beste post-baldintza gogorrago bat behar izatea. Arrazonatzeko gabezia eta ezintasun honi aurre egiteko, Hoare-ren sistema formaleko *Ondorioaren Erregela* erabil dezakegu. Formalki, *Ondorioaren Erregela* horrela definitzen da:

(ODE) $$\frac{\phi \ \rightarrow \ \phi_1, \{ \phi_1 \} \ \mathtt{S} \ \{ \psi_1 \}, \psi_1 \ \rightarrow \ \psi}{\{ \phi \} \ \mathtt{S} \ \{ \psi \}}$$

Erregela honek baditu ohikoak diren bi aldaera edo kasu partikular:

$$\frac{\phi \;\rightarrow\; \phi_1,\ \{\ \phi_1\ \}\ \mathtt{S}\ \{\ \psi\ \}}{\{\ \phi\ \}\ \mathtt{S}\ \{\ \psi\ \}} \qquad \frac{\{\ \phi\ \}\ \mathtt{S}\ \{\ \psi_1\ \},\ \psi_1 \;\rightarrow\; \psi}{\{\ \phi\ \}\ \mathtt{S}\ \{\ \psi\ \}}$$

Inferentzi erregela honek adierazten du beharrezkoa denean baieztapen baten aurre-baldintza ahuldu daitekeela, eta post-baldintza, gogortu.

(AA) axioma eta **(ODE)** inferentzi erregela erabiliz, gure lehenengo baieztapena frogatu dezakegu. Horretarako, pauso bat baino gehiagoko frogapena egin beharko dugu. Orokorrean, frogapen bat axiomen eta inferentzi erregelen aplikaziotik eratortzen den segida bat da, tartean ondorio logikoak ere erabil daitezkeelarik. Segidaren azken baieztapena izango da frogatu nahi dugun zuzentasun partzialari buruzko propietatea.

Frogapenak *goitik beherakoak* edo *behetik gorakoak* izan daitezke. Goitik beherako frogapenak baieztapenaren aurre-baldintzatik hasten dira, eta pixkanaka baieztapen orokorragoak frogatzen dira, axiomak, inferentzi erregelak eta ondorio logikoak erabiliz *aurreranzko* arrazonamenduan. Azken urratsean, frogatu nahi zen hasierako baieztapena frogatuko da. Aldiz, behetik gorako frogapenak post-baldintzatik hasten dira. Honakoan ere, segidaren azken urratsean hasierako propietatea frogatuko da, baina *atzeranzko* bidea egingo da.

Adibidez, honako hau da aurreko baieztapenaren goitik beherako frogapen bat:

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

Frogapen horretan, lehenengo urratsean aurre-baldintza ahultzen da, ondorio logiko baten bidez. Ondoren, **(AA)** aplikatzen da bigarren urratsean. Eta, azkenik, hirugarren urratsean, hasierako propietatea frogatzen da, **(ODE)** aplikatuz 1. eta 2. urratseko propietateei. Kontuan hartu **(ODE)** erregelaren lehenengo aldaera erabili dugula.

Behetik gorako frogapena honelakoa izan daiteke:

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

Kasu honetan, post-baldintzatik hasita **(AA)** erabiltzen dugu. Ondoren, ateratako aurre-baldintza gogortzen dugu ondorio logiko baten bidez. Eta, azkenik, frogapena bukatzen dugu **(ODE)** erabiliz 1. eta 2. propietateen gainean.

Ikus ditzagun frogapen-adibide gehiago.

Izan bitez $A(1..n)$ osokoen bektore bat eta *positibo* predikatua:

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

Honako baieztapen hau

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

zuzena dela frogatuko dugu, behetik gorako hurbilpena erabilita:

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

Frogapen horretan, **(AA)** axiomaren aplikaziotik eratorritako aurre-baldintza hasierako aurre-baldintza baino ahulagoa da. Horregatik, **(ODE)** erregela erabiltzeko beharra dago. Saia zaitez goitik beherako frogapena egiten.

Beste adibide bat:

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

Propietate hori frogatzeko ere behetik gorako hurbilpena erabiliko dugu:

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

Frogapen horretako giltzarria bigarren propietatean dago, non ondorio logikoaren bidez $(\ 1 \leq i < n\ )$ formulatik $(\ 1 \leq i+1 \leq n\ )$ formula lortzen den. Kontuan izan behar da bi formula horiek ez direla baliokideak.

Beste adibide bat:

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

Berriz ere behetik gora frogatzen da hemen:

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

Beste adibide bat

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

non b aldagai boolearra den. Hurrengo frogapenean goitik beherako bidea egiten da, eta bi ondorio logiko erabiltzen dira **(ODE)** aplikatu ahal izateko:

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

Hala ere, ez da guztiz beharrezkoa bi ondorio logiko erabiltzea, eta bakar batekin ere frogapena osa daiteke:

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

Asignazio bakarreko programa-zatiak egiaztatzeko gai gara asignazioaren axioman eta ondorioaren erregelan oinarrituta. Hala ere, programazio agintzailearen izaerak, non aldagaiek beren balioa aldatu eta eguneratu egiten duten, notazio-arazoak sor ditzake. Ikus dezagun adibide bat, esaten ari garena hobeto azaltzeko. Demagun honako hirukotea egiaztatzeko eskatzen zaigula:

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

Lehenengo begiratuan, tupustean, baieztapena egiazkoa dela esateko joera da ohikoa. Baina istant batez pentsatuz gero, berehala konturatzen gara $x = x + 1$ berdintza ezinezkoa dela, faltsua dela definizioz. Zer gertatzen da? Kontua da asignazio berean bi

x aldagai erabiltzen ari garela, bata asignazioa gauzatu aurrekoa, eta bestea ondorengoa. x aldagaiak ez du izena aldatzen, baina x erreferentziatzen dugunean asertzioan bi x desberdinez ari gara. Beraz, asertzio egokiak idatz ditzagun bi x horien balioak desberdindu beharra daukagu. Notazio-arazo honi irtenbide desberdinak eman izan zaizkio. Guk honako hau erabiliko dugu: *aldagai batek datuaren eta emaitzaren rola jokatzen badu programa batean, aldagai horren hasierako eta bukaerako balioak bereizteko, aurre-baldintzan izendatu egingo dugu zein den bere hasierako balioa. Hortik aurrera hasierako balioaren erreferentzia egin nahi dugunean, hasierako balio hori aipatuko dugu.* Modu honetan, garbi bereizten dira aldaketaren aurreko balioa eta aldaketaren ondorengoa. Gure adibidera etorriz, honela formulatuko genuke hirukotea:

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

Kontuan izan behar da b balioa generikoa dela, orokorra, eta ez duela inolako murriztapenik ezartzen x-ren gainean. Jakina, b izenaren hautua ez da garrantzitsua. Beste edozein izen erabil zitekeen, programaren testuinguruan aldagai-izenekin talkarik egin gabe, noski. Aldagaiak aldi berean datu eta emaitza direnerako konbentzio hau sistematikoki erabiliko da irakasgai honetan.

3.3 Konposaketa sekuentziala

Orain arte frogatu ahal izan ditugun baieztapenak agindu bakarrekoak ziren. Atal honetan ikusiko dugu nola frogatu agindu bat baino gehiagoko baieztapenak.

Horretarako, beste erregela bat erabiliko dugu: *Konposaketaren Erregela.* Formalki, *Konposaketaren Erregela* horrela definitzen da, formulaziorik orokorrenean

(KPE)
$$\frac{\{\ \phi\ \}\ \mathtt{S_1}\ \{\ \theta_1\ \},\ \{\ \theta_1\ \}\ \mathtt{S_2}\ \{\ \theta_2\ \},\ \ldots\ ,\ \{\ \theta_{n-1}\ \}\ \mathtt{S_n}\ \{\ \psi\ \}}{\{\ \phi\ \}\ \mathtt{S_1};\ \mathtt{S_2};\ \ldots\ ;\ \mathtt{S_n}\ \{\ \psi\ \}}$$

non $S_1,\ S_2,\ \ldots, S_n$ bata bestearen segidan dauden aginduak (edo agindu-blokeak) diren.

Erregela honek badu formulazio sinpleagoa bi aginduen segidarako:

$$\frac{\{\ \phi\ \}\ \mathtt{S_1}\ \{\ \theta\ \},\ \{\ \theta\ \}\ \mathtt{S_2}\ \{\ \psi\ \}}{\{\ \phi\ \}\ \mathtt{S_1};\ \mathtt{S_2}\ \{\ \psi\ \}}$$

*Konposaketaren erregela*k adieraz ten du $\{\ \phi\ \}\ \mathtt{S_1};\ \mathtt{S_2}\ \{\ \psi\ \}$ moduko baieztapenak frogatu ahal izateko tarteko asertzio bat (θ) erabili behar dela. Tarteko asertzio horrekin bi baieztapen osatzen dira. Lehenengo baieztapena $\{\ \phi\ \}\ \mathtt{S_1}\ \{\ \theta\ \}$ da, non θ baieztapenaren post-baldintza den, eta bigarren baieztapena, $\{\ \theta\ \}\ \mathtt{S_2}\ \{\ \psi\ \}$ da, non θ aurre-baldintza den. Hau da, lehenengo baieztapenaren post-baldintza bigarren baieztapenaren aurre-baldintza da. Bi aginduen arteko zubi-lana egiten du tarteko asertzioak. Horrela, bi baieztapen horiek konposatuz, $\{\ \phi\ \}\ \mathtt{S_1};\ \mathtt{S_2}\ \{\ \psi\ \}$ froga daiteke. Orokorrean, n aginduko segidak baditugu, $n-1$ tarteko asertzio erabili behar dira, eta tarteko baieztapenak konposatuta baieztapen orokorra frogatuko dugu.

Argitu dezagun erregelaren aplikazioa adibide baten bidez. Demagun zuzentasun partzialari buruzko honako baieztapena frogatu nahi dugula:

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

Hiru agindu daudenez, bi tarteko asertzio erabili behar dira frogapena egiteko. Orain ere frogapenak behetik gorakoak edo goitik beherakoak izan daitezke. Nola jokatzen dugun, behetik gora ala goitik behera, tarteko asertzio desberdinak aterako zaizkigu:

	Tarteko asertzioak	
	Behetik gorakoak	Goitik beherakoak
$\{\ x = a\ \wedge\ y = b\ \}$		
`x := x+y;`	$\theta_1^1\ \equiv \{\ x = a + b\ \wedge\ y = b\ \}$	$\theta_1^2\ \equiv \{\ x - y = a\ \wedge\ y = b\ \}$
`y := x-y;`	$\theta_2^1\ \equiv \{\ x = a + b\ \wedge\ y = a\ \}$	$\theta_2^2\ \equiv \{\ x - y = b\ \wedge\ y = a\ \}$
`x := x-y;`		
$\{\ x = b\ \wedge\ y = a\ \}$		

Tarteko asertzio horiek erabiliz, behetik gorako frogapena horrelakoa izan daiteke:

1. $\{\ x = a + b\ \wedge\ y = a\ \}$
   `x := x-y;`
   $\{\ x = b\ \wedge\ y = a\ \}$ **(AA)**
2. $\{\ x = a + b\ \wedge\ y = b\ \}$
   `y := x-y;`
   $\{\ x = a + b\ \wedge\ y = a\ \}$ **(AA)**
3. $\{\ x = a\ \wedge\ y = b\ \}$
   `x := x+y;`
   $\{\ x = a + b\ \wedge\ y = b\ \}$ **(AA)**
4. $\{\ x = a\ \wedge\ y = b\ \}$
   ```
   x := x+y;
   y := x-y;
   x := x-y;
   ```
   $\{\ x = b\ \wedge\ y = a\ \}$ 1, 2, 3 eta **(KPE)**

eta, goitik beherako frogapena:

1. $\{\ x = a\ \wedge\ y = b\ \}$
   `x := x+y;`
   $\{\ x - y = a\ \wedge\ y = b\ \}$ **(AA)**
2. $\{\ x - y = a\ \wedge\ y = b\ \}$
   `y := x-y;`
   $\{\ x - y = b\ \wedge\ y = a\ \}$ **(AA)**
3. $\{\ x - y = b\ \wedge\ y = a\ \}$
   `x := x-y;`
   $\{\ x = b\ \wedge\ y = a\ \}$ **(AA)**

4. $\{\ x = a \ \wedge \ y = b\ \}$
```
   x := x+y;
   y := x-y;
   x := x-y;
```
$\{\ x = b \ \wedge \ y = a\ \}$ 1, 2, 3 eta **(KPE)**

3.4 Baldintzazko sententziak

ADAren kontrol-egituretan aurrera eginez, *baldintzazko sententziak* jorratuko ditugu atal honetan.

Baldintzazko aginduen egitura ohiko bat honako hau da:

```
if B then
   S1;
else
   S2;
end if;
```

if-then-else patroiko agindu hori exekutatzen denean, B baldintza ebaluatu eta ebaluazio horren arabera S_1 ala S_2 agindu-blokea exekutatuko da. Hau da, B baldintza egiazkoa bada, orduan S_1 exekutatzen da, bestela S_2. Bi kasuetan, ϕ betetzen den egoera batean hasten den edozein konputazio ψ betetzen den egoera batean bukatuko da, bidean S_1 edo S_2 agindu-segida exekutatu delarik. Horrenbestez, hurrengo baieztapena frogatzeko

$\{\ \phi\ \}$
```
   if B then
      S1;
   else
      S2;
   end if;
```
$\{\ \psi\ \}$

bi aukerak hartu behar ditugu kontuan, B betetzen denekoa eta betetzen ez denekoa. Zehazki, B baldintza betetzen bada, orduan S_1 agindu-segida exekutatzen denez,

$$\{\ \phi \ \wedge \ B\ \}\ \texttt{S1;}\ \{\ \psi\ \}$$

baieztapena frogatu behar da. Bestela, B baldintza betetzen ez bada, orduan S_2 exekutatzen denez,

$$\{\ \phi \ \wedge \ \neg B\ \}\ \texttt{S2;}\ \{\ \psi\ \}$$

baieztapena frogatu behar da. Gainera, B baldintzaren ebaluazio egokia bermatzeko, B-k definituta egon behar du. Hau da, $def(B)$ ere bete behar da, konputazioa errore batez buka ez dadin. Kontuan izan def predikatua terminoekin erabili dugula **(AA)** axioman, eta orain formulekin erabiltzen ari garela. B baldintza (edo formula) definituta egongo da, baldin eta soilik baldin B-n agertzen diren termino guztiak definituta badaude. Edonola ere, B-ren egia-balioa B-n agertzen diren terminoen araberakoa da. Ez dira, hortaz, bi kontzeptuak nahastu behar: gauza bat da baldintza ondo definituta egotea, eta beste bat bere ebaluazioak balio egiazkoa ($true$) ala faltsua ($false$) ematea.

Aztertzen ari garen baldintzazko aginduan ez dugu eskatzen B baldintza edozein konputazio-egoeratan definituta egotea, baizik eta ϕ betetzen den edozein egoeratan hala egotea. Horregatik, $(\phi \rightarrow def(B))$ betetzen dela frogatzearekin nahikoa da.

Honako hau da *baldintzaren erregela*ren formulazioa *if-then-else* patroiko aginduentzat:

(BDE)

$$\frac{\{ \phi \wedge B \}\ \mathtt{S_1}\ \{ \psi \},\ \{ \phi \wedge \neg B \}\ \mathtt{S_2}\ \{ \psi \},\ (\phi \rightarrow def(B))}{\{ \phi \}\ \underline{\mathtt{if}}\ \mathtt{B}\ \underline{\mathtt{then}}\ \mathtt{S_1;}\ \underline{\mathtt{else}}\ \mathtt{S_2;}\ \underline{\mathtt{end\ if}}\mathtt{;}\ \{ \psi \}}$$

(BDE) inferentzi erregelak aukera ematen du hau bezalako baieztapenak frogatu ahal izateko:

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

Hona frogapena:

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

Adibide honetan egin dugun bezala, baldintzazko aginduen frogapenak, normalean, goitik beherakoak izango dira.

Hala ere, baldintzazko aginduak era askotakoak dira. Adibidez, agindu batzuek ez dute *else* kasurik:

```
if B then
   S;
end if;
```

Horrelakoetan, B baldintza betetzen bada, orduan S agindu-segida exekutatzen da. Baina B betetzen ez bada, ez da agindurik exekutatzen. Horrek esan nahi du ϕ bete eta B betetzen ez den edozein egoeratan ψ bete beharrekoa dela:

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

Propietate horrek *else* bidearen exekuzioa islatzen du. Hortaz, horrela geratzen da *baldintzaren erregela else* espliziturik gabeko kasuetan:

(BDE)

$$\frac{\{ \phi \wedge B \}\ \texttt{S}\ \{ \psi \},\ (\phi \wedge \neg B) \rightarrow \psi,\ (\phi \rightarrow def(B))}{\{ \phi \}\ \texttt{if B then S; end if;}\ \{ \psi \}}$$

Baldintzaren erregela egokitu horren bidez froga daiteke honako baieztapen hau:

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

Frogapena:

1. $(x = a \wedge x < 0) \rightarrow (-x = |a|)$
2. $\{ -x = |a| \}$
   ```
   x := -x;
   ```
   $\{ x = |a| \}$ **(AA)**
3. $\{ x = a \wedge x < 0 \}$
   ```
   x := -x;
   ```
   $\{ x = |a| \}$ 1, 2 eta **(ODE)**
4. $(x = a \wedge x \geq 0) \rightarrow (x = |a|)$
5. $(x = a) \rightarrow def(x < 0)$
6. $\{ x = a \}$
   ```
   if x < 0 then
      x := -x;
   end if;
   ```
   $\{ x = |a| \}$ 3, 4, 5 eta **(BDE)**

Ikusitakoez gain, baldintzazko aginduek beste formatu batzuk ere har ditzakete, zenbaitetan konplexuagoak direnak. Adibidez, adarkatze anizkoitza posible da, agindu honetan bezala:

```
if B1 then
   S1;
elsif B2 then
   S2;
else
   S3;
end if;
```

Kasu honetan, B_1 betetzen bada, S_1 exekutatzen da. Baina B_1 betetzen ez bada, orduan B_2 baldintza ebaluatu eta, egiazkoa bada, S_2 exekutatuko da. Aldiz, B_1 eta B_2 betetzen ez badira, S_3 exekutatzen da. Kontuan izan B_2 baldintza ez dela ebaluatzen B_1 betetzen bada. Portaera hori jasotzeko, honako inferentzi erregela daukagu:

$$\frac{\begin{array}{c}\{\ \phi \wedge B_1\ \}\ \mathtt{S_1}\ \{\ \psi\ \},\ \{\ \phi \wedge \neg B_1 \wedge B_2\ \}\ \mathtt{S_2}\ \{\ \psi\ \},\\ \{\ \phi \wedge \neg B_1 \wedge \neg B_2\ \}\ \mathtt{S_3}\ \{\ \psi\ \},\ (\ \phi \rightarrow def(B_1)\ ),\\ (\ \phi \wedge \neg B_1\ ) \rightarrow def(B_2)\end{array}}{\{\ \phi\ \}\ \underline{\mathtt{if}}\ \mathtt{B_1}\ \underline{\mathtt{then}}\ \mathtt{S_1};\ \underline{\mathtt{elsif}}\ \mathtt{B_2}\ \underline{\mathtt{then}}\ \mathtt{S_2};\ \underline{\mathtt{else}}\ \mathtt{S_3};\ \underline{\mathtt{end\ if}};\ \{\ \psi\ \}}$$

Orain artean aztertuko aginduek erakutsi digutenez, baldintzazko aginduen egiturei egokitzen zaizkie inferentzi erregelak. Hau da, aginduen egitura aztertuz, erabaki behar dugu zein propietate frogatu behar diren baldintzazko agindua osotasunean frogatzeko. Esan daiteke, hortaz, ez dagoela baldintzazko erregela bakarra, baizik eta baldintzazko aginduaren egiturak agintzen duela nolakoa izan behar duen inferentzi erregelak (*Asignazioaren Axioma* eta esleipenekin gertatzen den bezalaxe). Baina, bestalde, bada beti betetzen den patroi bat: baldintzazko agindu bat frogatzeko beharrezkoa da i) edozein adar edo bidetatik joanda ondoko baldintza beteko dela egiaztatzea, eta ii) baldintza guztiak ondo definituta egotea. Printzipio orokor horiek jaso behar dituzte baldintzazko aginduen frogapenek, edozein egiturakoak direlarik ere.

3.5 Frogapenaren eskema: programa dokumentatzeko modua

Prometan askotariko aginduak erabiltzen dira. Agindu horiek zein motakoak diren, horien araberako axioma edo inferentzi erregela egokiak erabili behar dira frogapenean. Programak sinpleak edo agindu gutxikoak diren bitartean, sistema formalaren gaineko propietate-frogapenak zorrotz egin daitezke, pausoz pauso eta axiomak nahiz inferentzi erregelak banan-banan kateatuta. Baina, dimentsioz eta konplexutasunez programa handiagoak landu behar direnean, frogapen-teknika horren erabilera laburragoa eta eskematikoagoa komeni izaten da.

Laburtze horri begira, komenigarria da frogapenari ekiterako zein propietate frogatu behar diren aztertzea. Propietate horiek prometan sartzen dira asertzio gisa, horrela programa *dokumentatuz*.

Adibidez, har dezagun honako baieztapen hau:

$$\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i < n\ \}$$

```
   i := i+1;
   if A(i) = 0 then
      zenbatzero := zenbatzero+1;
   end if;
```

$$\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \}$$

Programa-zati horretan bi agindu daude: asignazio bat eta baldintzazko agindu bat. Hortaz, batetik **(AA)** (eta beharrezkoa bada, **ODE**) erabiliko ditugu asignazioaren zuzentasuna frogatzeko, eta bestetik **(BDE)**, baldintzazko aginduaren zuzentasuna frogatzeko. Asignazioaren post-baldintza izango da baldintzazko aginduaren aurre-baldintza, hori izango da, hortaz, tarteko asertzioa. Adar bakarreko baldintzazko agindua denez, edo, nahi bada, *else* espliziturik ez dagoenez, $A(i) = 0$ baldintza betetzen ez denean ondorio logikoak eraman behar du post-baldintza betetzera. Azkenik,

(KPE) erabiliko dugu baieztapen osoaren zuzentasuna frogatzeko. Horretarako, tarteko asertzioa erabiliko dugu, bi aginduen arteko lotura markatzen duelako. Beraz, hiru propietate nagusi daude frogapenaren nondik norakoak markatzen dituztenak: i) tarteko asertzioa, ii) *then* adarraren bidetik joanda gertatzen den asertzioa, iii) baldintza betetzen ez den kasurako, ondorio logikoa. Propietate horiek adierazteak frogapenaren eskema definitzen du, eta, bide batez, *programaren dokumentazioa* osatzen du:

$\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i < n\ \}$
`i := i+1;`
$\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i-1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \} \equiv \theta$
`if A(i) = 0 then`
$\{\ zenbatzero + 1 = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \}$
`zenbatzero := zenbatzero+1;`
$(\ \theta\ \wedge\ A(i) \neq 0\ )\ \rightarrow\ (\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ )$
`end if;`
$\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \}$

Frogapenaren eskema horrek ondo nabarmentzen ditu egiaztapenaren alderdi esanguratsuenak. Hala ere, eskeman jarri diren asertzioak ez dira hutsetik sortzen. Horien atzean eta horiek justifikatzeko, beti da beharrezkoa sistema formala ondo aplikatzea (agerian ez bada ere). Horregatik, gai izan behar dugu pausoz pausoko frogapena egiteko:

1. $(\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i < n\ )$
   $\rightarrow\ (\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i + 1 - 1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i + 1 \leq n\ )$
2. $\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i + 1 - 1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i + 1 \leq n\ \}$
   `i := i+1;`
   $\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i-1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \} \equiv \theta$ **(AA)**
3. $\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i < n\ \}$
   `i := i+1;` 1, 2 eta **(ODE)**
   $\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i-1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \} \equiv \theta$
4. $(\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i-1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \wedge\ A(i) = 0\ )$
   $\rightarrow\ (\ zenbatzero + 1 = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ )$
5. $\{\ zenbatzero + 1 = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \}$
   `zenbatzero := zenbatzero+1;`
   $\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \}$ **(AA)**
6. $\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i-1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \wedge\ A(i) = 0\ \}$
   `zenbatzero := zenbatzero+1;` 4, 5 eta **(ODE)**
   $\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \}$
7. $(\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i-1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \wedge\ A(i) \neq 0\ )$
   $\rightarrow\ (\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ )$
8. $(\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i-1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ )$
   $\rightarrow\ (\ 1 \leq i \leq n\ )\ \rightarrow\ def(A(i))$

9. $\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i-1\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \}$

```
if A(i) = 0 then
   zenbatzero := zenbatzero+1;
end if;
```

6, 7, 8 eta **(BDE)**

$\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \}$

10. $\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i < n\ \}$

```
i := i+1 then
if A(i) = 0 then
   zenbatzero := zenbatzero+1;
end if;
```

3, 9 eta **(KPE)**

$\{\ zenbatzero = \mathcal{N}k\ (\ 1 \leq k \leq i\ \wedge\ A(k) = 0\ )\ \wedge\ 1 \leq i \leq n\ \}$

3.6 Iterazioak: inbarianteak

Orain arte ikusitako aginduak behin baino ez dira exekutatzen, baina prometan oso ohikoa da agindu-segida batzuk behin baino gehiagotan exekutatzea. Prozesu errepikakor horiek inplementatzeko modu bat iterazioen kontrol-egiturak erabiltzea da. Atal honetan, ADA programazio-lengoaian definituta dauden kontrol-egitura iteratibo batzuk landuko ditugu. Adibidez, *while* aginduak:

```
while B loop
   S;
end loop;
```

Kontrol-egitura hau exekutatzen denean, B baldintza betetzen bada orduan S aginduen segida exekutatzen da. S-ren exekuzioa bukatu ondoren, berriro B baldintza aztertzen da. Eta horrela, behin eta berriro, harik eta B baldintza betetzen ez den arte.

Era honetan, agindu-segida errepikatuz, gauzatzen du iterazioak bere egitekoa. Iterazioa zentzuzkoa bada, aldaketak aldaketa, bada egindakoa islatzen duen eta prozesuan zehar kontserbatzen den propietate bat. Adibidez, honako iterazioa hartuz gero

$\{\ x = a\ \wedge\ y = b \geq 0\ \}$

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

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

espezifikazioak dioenez, x aldagaian kalkulatzen da x eta y aldagaien batura. Batuketa pausoz pauso kalkulatzen da: iterazio bakoitzean bat gehitzen zaio x-ri eta bat kentzen zaio y-ri. Hori horrela, begizta bakoitzean exekutatzen den agindu-segidak kontserbatu egiten du honako propietate hau:

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

Beste baieztapen honetan

$\{\ x \geq 1\ \wedge\ y = 1\ \}$

```
while 2*y <= x loop
   y := 2*y;
end loop;
```

$\{\ y = max\{k \mid ber2(k)\ \wedge\ k \leq x\}\ \}$

x baino txikiagoa den 2-ren berretura handiena bueltatzen da y aldagaian, non $ber2(k)$ atomoa egiazkoa den baldin eta soilik baldin k 2ren berretura bada. Kasu honetan, y aldagaia 1 balioarekin dator, eta iterazio bakoitzean bere balioa bikoiztu egiten da. Iterazioak kontserbatzen duen propietatea hau da:

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

Hirugarren adibide honetan

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

osokoen bektore bateko elementuen batura kalkulatzen da s aldagaian. Horretarako, iterazio bakoitzean bektoreko elementu bat batzen zaio s aldagaiari. Portaera hori adierazten duen eta iterazioak kontserbatzen duen propietatea honako hau da:

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

Ikusi ditugun adibide hauetan S agindu-segidak aldagaiak aldatzen ditu, baina balio berriek kontserbatu egiten dute delako propietatea. Iterazioek kontserbatzen duten propietate horri *inbariante* deitzen zaio. Garbi utzi behar da inbariantea ez dela S-k kontserbatzen duen edozein propietate, baizik eta iterazioaren semantika adierazten duena, hain juxtu. Azkeneko adibidera joz, A taulako elementuen batuketa egiten duen iterazioan, esan dezakegu $(0 \leq i \leq n)$ propietatea ere kontserbatzen dela, baina ez du ezer askorik adierazten iterazioaren eginkizunari buruz, eta, hortaz, ezin dezakegu inbariantetzat hartu.

Inbariantea (**INB**) asertzio bat da, *while* aginduetan baldintza ebaluatu aurreko puntuan kokatzen dena:

$\{ \ \phi \ \}$
```
   while { INB } B loop
      S;
   end loop;
```
$\{ \ \psi \ \}$

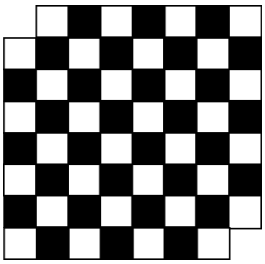
Konputazioa B ebaluatu aurreko puntu horretatik pasatzen den aldiro beteko da inbariantea. Beti: bai lehen aldiz pasatzen denean (artean iterazioaren barruko agindu-segida exekutatu ez denean) bai azkeneko aldiz pasatzen denean ere (B baldintzaren ebaluazioak $faltsu$ emango duenean). Gainera, kontserbatzen denez, B baldintza betetzen denean, S agindu-segidaren exekuzioak inbariantea berriz betetzea ekartzen du. Eta, horrela, *while* aginduaren exekuzioa bukatu arte. Hau da, B baldintza betetzen ez den arte.

Laburbilduz, inbarianteak honako propietate hauek beteko ditu:

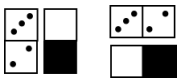
- S aginduen segidak erabiltzen eta aldatzen dituen aldagaiei dagokie,
- S aginduen segidak kontserbatu egiten du, eta

- iterazioaren semantika zehazten du.

Inbariantearen kontzeptua hobeto ulertu eta erabilgarritasuna erakusteko, programen egiaztapen formalaren mundu abstraktutik testuinguru errealago batera pasako gara. Jo dezagun xakeko taulatu bati ezkerreko goi-laukia eta eskuineko behe-laukia kendu egiten dizkiogula:



Hona erantzun beharreko galdera: *litekeena al da taulatua erabat estaltzea dominoko fitxekin?*



Dominoko fitxek bi laukitxo estaltzen dituzte eta modu horizontalean nahiz bertikalean jar daitezke taulatuan.

Galdera honi erantzuteko problemak soluzioren bat duen ala ez pentsatu behar dugu. Azter dezagun, bada, egoera: xakeko taula estandarretan 32 laukitxo beltz eta 32 laukitxo txuri daude. Honako honetan, ordea, 30 laukitxo beltz baino ez daude:

$$ZenbatTxuri = 32$$
$$ZenbatBeltz = 30$$

Hau da, bi laukitxo txuri gehiago daude laukitxo beltzak baino. Guztira, 62 laukitxo daude eta, beraz, 31 dominoko fitxa beharrezkoak izango lirateke xakeko taula guztia estaltzeko. Gainera, dominoko fitxa bat jartzen dugunean, laukitxo txuri bat eta laukitxo beltz bat estaltzen da, fitxak diagonalean jarri ezin direlako. Hortaz, lehenengo fitxa jarri ondoren, estali gabeko 31 laukitxo txuri eta 29 laukitxo beltz gelditzen dira:

$$ZenbatTxuri = 32 - 1 = 31$$
$$ZenbatBeltz = 30 - 1 = 29$$

Honek esan nahi du oraindik bi laukitxo txuri gehiago daudela laukitxo beltzak baino. Hurrengo fitxa jartzen dugunean ere erlazio hori mantentu egiten da. Eta berdin hurrengoan eta hurrengoan... Argi dago: fitxa beltzen eta txurien kopuruen arteko erlazio hori mantendu egiten da fitxak jarri ahala. Propietate horri eutsi egiten zaio prozesu errepikakorrean zehar. Horrenbestez, gure problemaren inbariantetzat har daiteke:

$$ZenbatTxuri - 2 = ZenbatBeltz$$

30 dominoko fitxa jarri ondoren, laukitxo beltz guztiak estalita daude eta, beraz, bi laukitxo txuri estali gabe gelditzen dira:

$$ZenbatTxuri = 2$$
$$ZenbatBeltz = 0$$

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

Xakeko taularen adibidea jarrita, ikusi dugu nola prozesu errepikakorraren inbariantea zehazteak erraztu egin duen galdera baten erantzun justifikatua aurkitzea. Fitxa beltzen eta txurien kopuruen arteko erlazioak ondo deskribatu du prozesu horretan zer gertatzen zen, eta bidea eman dio erantzun argi bati.

Antzera egiten da kontrol-egitura iteratiboen zuzentasun-egiaztapenean ere. Inbarianteak erabiliz, *while* aginduak erabiltzen dituen programen zuzentasun partziala egiazta daiteke. Horretarako, *While-ren Erregela* definituko dugu:

(WHE)

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

Erregela honek adierazten du *while* agindu baten zuzentasun partziala frogatzeko 4 propietate frogatu behar direla:

- Aurre-baldintzak inbariantea ondorioztatu behar du. Beste era batera esanda, konputazioa lehenbizikoz iteraziora iristen denean, inbariantea bete egin behar da.
- Inbariantea betetzen denean B baldintza ondo definituta dago.
- S agindu-segidak inbariantea kontserbatzen du.
- Inbariantea betetzen bada eta B baldintza betetzen ez bada, orduan ondoko baldintza betetzen da.

Orain arteko arrazonamenduan ez diogu heldu iterazioen bukaeraren arazoari. Esan gabe, baina suposatzen ari gara S agindu-segida kopuru finituan exekutatuko dela. Hau da, $\neg B$ noizbait beteko dela. Hurrengo atalean bukaera frogatzen ikasiko dugu, eta horrela zuzentasun osoa egiaztatu ahal izango dugu.

Goazen orain **(WHE)** aplikatzera *while* programa baten zuzentasuna partzialki frogatzeko. Har dezagun lehen aipatutako adibidea. Egiaztapena bideratzeko lehen urratsa inbariante bat definitzea da (xakeko taularen probleman egin dugun bezala):

$\{\ x = a \wedge y = b \geq 0\ \} \equiv\ \phi$

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

$\{\ x = a + b\ \} \equiv\ \psi$

INB $\equiv \{\ x + y = a + b\ \wedge\ 0 \leq y \leq b\ \}$

Inbariante horretan oinarrituta, eta **(WHE)** inferentzi erregela ardatz hartuta, hona frogapena:

1. $(\ \underbrace{x = a\ \wedge\ y = b \geq 0}_{\phi}\ )\ \rightarrow\ (\ \underbrace{x + y = a + b\ \wedge\ 0 \leq y \leq b}_{INB}\ )$
2. $(\ \underbrace{\underbrace{x + y = a + b\ \wedge\ 0 \leq y \leq b}_{INB}\ \wedge\ \underbrace{y \neq 0}_{B}}\ )$
   $\rightarrow\ (\ (x + 1) + (y - 1) = a + b\ \wedge\ 0 \leq y - 1 \leq b\ )$

3. $\{\ (x+1)+(y-1) = a+b\ \wedge\ 0 \leq y-1 \leq b\ \}$
   `x := x+1;`
   `y := y-1;`
   $\{\ x+y = a+b\ \wedge\ 0 \leq y \leq b\ \}$ **(AA)** eta **(KPE)**

4. $\{\ \underbrace{x+y = a+b\ \wedge\ 0 \leq y \leq b}_{INB}\ \wedge\ \underbrace{y \neq 0}_{B}\ \}$
   `x := x+1;`
   `y := y-1;`
   $\{\ \underbrace{x+y = a+b\ \wedge\ 0 \leq y \leq b}_{INB}\ \}$ 2, 3 eta **(ODE)**

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

6. $(\ \underbrace{x+y = a+b\ \wedge\ 0 \leq y \leq b}_{INB}\ )\ \rightarrow\ def(\underbrace{y \neq 0}_{B})$

7. $\{\ x = a \wedge y = b \geq 0\ \}$
   `while` $y \neq 0$ `loop`
   `x := x+1;`
   `y := y-1;`
   `end loop;`
   $\{\ x = a+b\ \}$ 1, 4, 5, 6 eta **(WHE)**

Ikus daitekeenez, lehenengo urratsean aurre-baldintzak inbariantea ondorioztatzen duela ($\phi \rightarrow INB$) frogatzen da, laugarren urratsean S agindu-segidak inbariantea kontserbatzen duela (bigarren eta hirugarren urratsetan oinarrituz), bosgarren urratsean inbarianteak eta B baldintza ezeztatuak post-baldintza ondorioztatzen dutela, eta seigarren urratsean baldintza definituta dagoela inbariantea betetzen bada.

Era berean froga genezake lehen erabili dugun bigarren adibidearen zuzentasun partziala. Hona hemen programa espezifikatua eta inbariantea:

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

Frogapen formala:

1. $(\ \underbrace{x \geq 1\ \wedge\ y = 1}_{\phi})\ \rightarrow\ (\ \underbrace{ber2(y)\ \wedge\ 1 \leq y \leq x}_{INB}\ )$

2. $(\ \underbrace{ber2(y)\ \wedge\ 1 \leq y \leq x}_{INB}\ \wedge\ \underbrace{2 \times y \leq x}_{B}\ )\ \rightarrow\ (\ ber2(2 \times y)\ \wedge\ 1 \leq 2 \times y \leq x\ )$

3. $\{\ ber2(2 \times y)\ \wedge\ 1 \leq 2 \times y \leq x\ \}$
   `y := 2*y;`
   $\{\ \underbrace{ber2(y)\ \wedge\ 1 \leq y \leq x}_{INB}\ \}$ **(AA)**

4. $\{\ \underbrace{ber2(y)\ \wedge\ 1 \leq y \leq x}_{INB}\ \wedge\ \underbrace{2 \times y \leq x}_{B}\ \}$

   `y := 2*y;`

   $\{\ \underbrace{ber2(y)\ \wedge\ 1 \leq y \leq x}_{INB}\ \}$ 2, 3 eta **(ODE)**

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

6. $(\ \underbrace{ber2(y)\ \wedge\ 1 \leq y \leq x}_{INB}\ )\ \rightarrow\ def(\underbrace{2 \times y \leq x}_{B})$

7. $\{\ x \geq 1\ \wedge\ y = 1\ \}$

   `while 2*y <= x loop`

   `   y := 2*y;`

   `end loop;`

   $\{\ y = max\{k \mid ber2(k)\ \wedge\ k \leq x\}\ \}$ 1, 4, 5, 6 eta **(WHE)**

3.7 Iterazioak: bukaera

Aurreko ataletan iterazioen zuzentasun partziala frogatzeko metodoa aztertu dugu. Baina, 3.1 atalean ikusi dugunez, zuzentasun partzialetik zuzentasun osora pasatzeak programen bukaera ere egiaztatzea eskatzen du. Iterazioetan, nola prozesu errepikakorrak diren, badago ez bukatzeko arriskua. Beraz, garrantzitsua da iterazioak bukatzen direla frogatzea.

Bukaera frogatzeko gure hurbilpena iterazio-pausoen kopuruan oinarritzen da. Alegia, iterazio batek beteko du bukaeraren propietatea baldin eta bere edozein konputazio amaitzen bada pauso edo begizta kopuru finitu batean. Iterazio-pausoen kopurua aztertuko dugu, eta ikusi ea beti zenbaki finitua den.

Kontuan izatekoa da, bestetik, bukaeraren propietatea hasierako egoerei (aurreko baldintzari) estuki lotuta dagoela. Iterazio baten exekuzioa nola abiatzen den aztertzea funtsezkoa da erabakitzeko bukatuko den ala ez.

Horiek horrela, hartu dugun hurbilpenak eskatzen du kontuan hartzea bai aurrebaldintza bai inbariantea bukaera frogatzeko. Horregatik, S programak bukaeraren propietatea betetzen duela adierazteko *Buka* predikatua erabiliko dugu bi argumenturekin: $Buka(S, \Phi)$, non bigarren argumentuak adierazten duen programaren aurrebaldintza. Gogora dezagun 3.1 atalean honela formulatu dugula zuzentasun osoa: S programa *osoki zuzena* da $(\ \Phi,\ \Psi\ )$ espezifikazioarekiko

$$\{\ \Phi\ \}\ [\ \mathtt{S}\ ]\ \{\ \Psi\ \}$$

baldin eta soilik baldin Φ betetzen den egoera batean hasten den Sren edozein konputazio Ψ betetzen den egoera batean bukatzen bada. Hau da, programa bat osoki zuzena da espezifikazio batekiko partzialki zuzena izateaz gain beti bukatzen bada:

$$\{\ \Phi\ \}\ [\ \mathtt{S}\ ]\ \{\ \Psi\ \} \equiv \{\ \Phi\ \}\ \mathtt{S}\ \{\ \Psi\ \} + Buka(S, \Phi)$$

Hurrengo adibideak erabiliko ditugu argiago azaltzeko nola eragiten duten aurrebaldintzak eta inbarianteak programa baten bukaeraren frogapenean.

Lehenengo adibidea:

$S_1 =$ `while i` $\neq$ `0 loop`
`i := i-1;`
`end loop;`

Nabaria da { *true* } S_1 { $i = 0$ } baieztapena betetzen dela. $Buka(S_1, true)$ baieztapena, ordea, ez da betetzen, izan ere, *true* aurre-baldintza ahulegia baita (i-ren balio negatiboetarako ez da iterazioa bukatuko). Hau da, { *true* } [S_1] { $i = 0$ } ezin dugu frogatu. Aldiz, $i \geq 0$ aurre-baldintza hartuz gero, betetzen dira { $i \geq 0$ } S_1 { $i = 0$ } eta $Buka(S_1, i \geq 0)$ baieztapenak. Ondorioz, { $i \geq 0$ } [S_1] { $i = 0$ } egiazta daiteke.

Bigarren adibidea:

$S_2 =$ `while i` $\neq$ `0 loop`
`i := i-2;`
`end loop;`

{ *true* } S_2 { $i = 0$ } baieztapenaren zuzentasun partziala ere froga daiteke: bistan da iterazioa bukatzen bada $i = 0$ egoeran bukatuko dela. $Buka(S_2, true)$ propietateari dagokionez, orain ere ez da beteko, *true* aurre-baldintza ahulegia baita berriz ere: i-ren hasierako balioa negatiboa denean ez da 0 izatera ailegatuko inoiz. $i \geq 0$ aurre-baldintza hartzen badugu, baiezta al dezakegu $Buka(S_2, i \geq 0)$ propietatea betetzen dela? Kasu honetan ere $i \geq 0$ aurre-baldintza ahulegia dela esan behar dugu, i-ren hasierako balioa bakoitia denean i ez delako inoiz 0 izatera iritsiko. Hirugarren saiakeran, $i \geq 0 \wedge bikoiti(i)$ aurre-baldintza hartuko dugu. Testuinguru honetan esan dezakegu $bikoiti(i)$ propietatea mantendu egiten dela iterazioan eta i-ren balioa gero eta txikiagoa izango dela. Beraz, orain bai, $Buka(S_2, i \geq 0 \wedge bikoiti(i))$ betetzen da. Ondorioz, { $i \geq 0 \wedge bikoiti(i)$ } [S_2] { $i = 0$ } froga daiteke.

Bi adibideok argi erakutsi digute iterazioen bukaera aurre-baldintzaren araberakoa dela. Modu intuitiboan egin dugun arrazonamendua formalizatzen saiatuko gara hemendik aurrera.

Ikus dezagun, bada. *while* iterazio bat emanda

`while B loop` { **INB** }
`S;`
`end loop;`

iterazio hori noizbait bukatuko da, edo, baliokidea dena, pauso kopuru finituan bukatuko da baldin

> *$INB \wedge B$ betetzen den egoera batean hasita, S agindu-segidak B faltsu egiten badu denbora finituan.*

Horretarako, derrigorrez S agindu-segidak B baldintzako aldagairen bat aldatu behar du.

Irakasgai honetan erabiliko dugun metodoa honako propietatean oinarrituko da:

> *Iterazioan zehar, E espresio batek balio arruntak baditu eta pauso bakoitzean balio hertsiki txikiagoa hartzen badu, pausoen kopuruak finitua izan behar du derrigor.*

Esan dezakegu $(\mathbb{N}, \leq)$ ondo oinarritutako ordena dela, ez daukalako beheranzko kate infiniturik. Adibidez:

$$80 > \ldots > 23 > \ldots > 14 > \ldots > 3 > \ldots > 0 \text{ (ez dago besterik)}$$

Eredu formal horretan oinarrituta, bukaera frogatzeko metodoak honako urrats hauek ditu:

1) E zenbakizko espresio bat (S-ko aldagaiez osatua) definitu behar da lehenbizi. E espresio horri *borne-adierazpen* deituko diogu. E espresioak mugatu (*bornatu*) egiten du iterazio bati bukatzeko falta zaizkion pausoen kopurua.

2) E espresioaren gainean honako bi propietate hauek frogatu behar dira:

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

   b) $\{ \ INB \ \wedge \ B \ \wedge \ E = z \ \}$ `S` $\{ \ E < z \ \}$, non $z \notin \mathit{Aldagaiak}(INB, B, S)$

Bigarren propietate horretan erabilitako z balioa generikoa da. E espresioak duen balioa adierazten du, baina ez du inongo murriztapenik ezartzen. Balioa izendatzeko erabiltzen den identifikadorea (adibide honetan z) edozein izan daiteke, baina ez du talkarik eragin behar *INB*-en, B-n edo S-n erabilitako identifikadoreekin.

Metodoa horrela justifikatzen da: izan bedi z_i balioa E espresioak i-garren pausoan hartzen duena ($E = z_i \in \mathbb{N}$). Honako bi propietate hauek betetzen badira

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

orduan $INB \ \wedge \ B$ ezin daiteke mugagabeki bete, eta halako batean $\neg B$ beteko da.

Horrenbestez, metodoko bi propietate horiek frogatuz gero (E espresioak balio arruntak hartzen dituela eta pausoero eguneratzen dela balio hertsiki txikiagoekin), esango dugu *while* moldeko agindu hau

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

pauso kopuru finituan amaitzen dela.

Azter dezagun adibide konkretu ezagun hau:

$\{ \ x = a \wedge y = b \geq 0 \ \}$

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

INB $\equiv \{ \ x + y = a + b \ \wedge \ 0 \leq y \leq b \ \}$

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

Orain baieztapenaren zuzentasun osoa frogatzerik badugu, bukaeraren azterketa ere egin dezakegulako. Lehenengo urratsa, metodoan zehaztu dugunaren arabera, E borne-adierazpena zehaztea da. Horretarako S agindu-segidak erabiltzen dituen aldagaiei erreparatuko diegu: bi aldagai agertzen dira, x eta y. Biak dira erabilgarri E borne-adierazpenean. 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. Kontuan izan, gainera, y espresio zenbakizkoak mugatu egiten duela iterazioari bukatzeko falta zaizkion pausoen kopurua. Kasu honetan, mugatu ez ezik, zehazki adierazten du zenbat pauso falta diren. Horrenbestez:

$$E \ \equiv \ y$$

Ondoren, aurretik aurkeztutako a eta b propietateak frogatu behar dira.

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

Era berean frogatuko dugu beste adibide honen bukaera:

$\{ x \geq 1 \ \wedge \ y = 1 \}$

```
while 2*y <= x loop
   y := 2*y;
end loop;
```

INB $\equiv \{ ber2(y) \ \wedge \ 1 \leq y \leq x \}$

$\{ y = max\{k \mid ber2(k) \ \wedge \ k \leq x\} \}$

E borne-adierazpena definitu behar dugu, hasteko. Kasu honetan, y aldagaia baino ez da agertzen S aginduen segidan, baina bere balioa gero eta handiagoa da iterazioak aurrera egin ahala. $E \ \not\equiv \ y$, espresioak balio gero eta handiagoak hartuko lituzke eta ez du mugatzen bukaerarako falta zaizkigun pausoen kopurua. $E \ \not\equiv \ -y$ espresioak ere ez du balio, E-k gero eta balio txikiagoak hartu arren, balio negatiboak hartuko lituzkeelako. Iterazioan erabiltzen da bigarren aldagai bat: x. Iterazioak ez du x-ren balioa aldatzen, baina y-k hartuko dituen balioen goi-muga bat markatzen du. Hortaz, honako borne-adierazpena finkatuko genuke:

$$E \ \equiv \ x - y$$

Metodoak ezarritako bi propietateak frogatzea dagokigu orain.

a)

$$(\underbrace{ber2(y) \ \wedge \ 1 \leq y \leq x}_{INB} \wedge \underbrace{2 \times y \leq x}_{B}) \ \rightarrow \ \underbrace{x - y}_{E} \in \mathbb{N}$$

b)
$$\{\ \underbrace{ber2(y)\ \wedge\ 1 \leq y \leq x}_{INB}\ \wedge\ \underbrace{2 \times y \leq x}_{B}\ \wedge\ \underbrace{x - y}_{E} = z\ \}$$
```
   y := 2*y;
```
$$\{\ \underbrace{x - y}_{E} < z\ \}$$

Frogapena:

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

Atal honetan adibidetzat hartu dugun hirugarren iterazioaren bukaera aztertuko dugu orain:

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

Programan i eta neg aldagaiak 0 balioarekin hasieratu eta beren balioa gero eta handiagoa da. n konstantea i eta neg baino handiagoa da. Horren arabera, bi aukera plantea zitezkeen E espresiorako: $n - i$ ala $n - neg$. Kontuan izanda borne-adierazpenak balio txikiagoa hartu behar duela iterazio bakoitzean, ez da egokia $n - neg$ espresioa, neg aldagaia ez delako iterazio guztietan handitzen (soilik $A(i) < 0$ gertatzen denean). $n - i$ espresioak, ordea, betetzen du pauso guztietan txikiagotzeko baldintza. $n - i$ zenbakizko espresioak ondo adierazten du zenbat falta den bukatzeko. Taulen tratamendu sekuentziala egiten denean, ohikoa da eredu horretako borne-adierazpenak definitzea: taulako elementuak tratatu ahala, oraindik tratatu gabeko elementuen kopurua mugatzen du borne-adierazpenak.

$E\ \equiv n - i$ hartuta, hona hemen bukaeraren frogapena:

a)
$$(\ \underbrace{0 \leq i \leq n\ \wedge\ neg = \mathcal{N}j\ (\ 1 \leq j \leq i\ \wedge\ A(j) < 0\ )}_{INB}\ \wedge\ \underbrace{i < n}_{B}\ )\ \rightarrow\ \underbrace{n - i}_{E} \in \mathbb{N}$$

b) Frogapena:

1. $(0 \leq i \leq n \ \wedge \ neg = \mathcal{N}j \ (\ 1 \leq j \leq i \ \wedge \ A(j) < 0 \ ) \ \wedge \ i < n \ \wedge \ n - i = z > 0 \ )$
   $\rightarrow \ (\ n - i = z > 0 \ )$
   $\rightarrow \ (\ n - (i+1) < z \ )$
2. $\{ \ n - (i+1) < z \ \}$
   `i := i+1;`
   $\{ \ n - i < z \ \}$ **(AA)**
3. $\{ \ 0 \leq i \leq n \ \wedge \ neg = \mathcal{N}j \ (\ 1 \leq j \leq i \ \wedge \ A(j) < 0 \ ) \ \wedge \ i < n \ \wedge \ n - i = z \ \}$
   `i := i+1;`
   $\{ \ n - i < z \ \}$ 1, 2 eta **(ODE)**
4. $\{ \ n - i < z \ \}$
   ```
   if A(i) < 0 then
      neg := neg+1;
   end if;
   ```
   $\{ \ n - i < z \ \}$ **(BDE)**
5. $\{ \ 0 \leq i \leq n \ \wedge \ neg = \mathcal{N}j \ (\ 1 \leq j \leq i \ \wedge \ A(j) < 0 \ ) \ \wedge \ i < n \ \wedge \ n - i = z \ \}$
   ```
   i := i+1;
   if A(i) < 0 then
      neg := neg+1;
   end if;
   ```
   $\{ \ n - i < z \ \}$ 3, 4 eta **(KPE)**

3.8 Inbarianteak programazio-ideien formalizaziorako

Aurreko ataletan inbarianteak eta borne-adierazpenak erabili ditugu *while* aginduen zuzentasun osoa frogatzeko. Baieztapenen formatuan

$$\{ \ \phi \ \} \ \texttt{[S]} \ \{ \ \psi \ \}$$

hiru osagai agertzen dira: ϕ aurre-baldintza, S agindu-segida eta ψ post-baldintza. Inbariantea eta borne-adierazpena ez zaizkigu hirukote horretan ematen, baizik eta asmatu egin behar ditugu frogapena egiteko.

Inbariante egokia asmatzea beharrezkoa da frogapena zuzen bideratzeko. Baina, nola asmatzen da inbariantea? Inbarianteak iterazioaren portaera adierazten duenez, *while* aginduak egiten duena aztertzea da inbariantea asmatzeko oinarririk sendoena. Iterazioak zer eta nola egiten duen aztertu, eta horren arabera asma daiteke inbariantea, beste ezer gabe. Alabaina, badira programen portaeratik eta espezifikaziotik modu erdi-automatikoan atera daitezkeen propietateak, inbariantea osatzeko lagungarriak direnak. Besteak beste, zenbaitetan laguntza handikoa izan ohi da post-baldintza aztertzea, eta hura oinarritzat hartzea.

Adibidez, demagun $A(1..n)$ bektoreko osagaien batura kalkulatzen duen S agindu-segida bat daukagula. S segida horren post-baldintza honako hau izan daiteke:

$$\psi \ \equiv \ x = \sum_{j=1}^{n} A(j)$$

Post-baldintza hori betetzen duten hainbat programa desberdin eduki ditzakegu. Adibidez, honako hiru baieztapen hauek zuzenak lirateke:

1. bertsioa:

$\{\ true\ \}$
```
   i := 0;
   x := 0;
   while i ≠ n loop      { INB1 }
      i := i+1;
      x := x+A(i);
   end loop;
```
$\{\ x = \sum_{j=1}^{n} A(j)\ \}$

2. bertsioa:

$\{\ true\ \}$
```
   i := 1;
   x := 0;
   while i ≠ n+1 loop      { INB2 }
      x := x+A(i);
      i := i+1;
   end loop;
```
$\{\ x = \sum_{j=1}^{n} A(j)\ \}$

3. bertsioa:

$\{\ true\ \}$
```
   i := 1;
   x := A(1);
   while i ≠ n loop      { INB3 }
      i := i+1;
      x := x+A(i);
   end loop;
```
$\{\ x = \sum_{j=1}^{n} A(j)\ \}$

Hiru programa iteratibo desberdin ditugu espezifikazio bera betetzen dutenak. Baina, zein dira inbarianteak? Zein desberdintasun dago INB_1, INB_2 eta INB_3-ren artean? Nola asma dezakegu programa bakoitzari dagokion inbariantea? Esan dugunez, *while* aginduak egiten duena aztertzea da inbariantea asmatzeko oinarririk sendoena. Baina, post-baldintza aztertzea, eta hura oinarritzat hartzea ere laguntza handikoa izan daiteke. Adibide honetan, hiru programei dagozkien inbarianteak post-baldintzatik atera ditzakegu:

$$INB_1 \equiv x = \sum_{j=1}^{i} A(j) \ \wedge\ 0 \leq i \leq n$$

$$INB_2 \equiv x = \sum_{j=1}^{i-1} A(j) \ \wedge\ 1 \leq i \leq n+1$$

$$INB_3 \equiv x = \sum_{j=1}^{i} A(j) \ \wedge\ 1 \leq i \leq n$$

INB_1-en, ψ oinarritzat hartu, eta ψ-ko n konstantea i aldagaiarekin ordezkatu dugu batukarian, eta ondoren i aldagaiaren balioa mugatzen dugu 0 eta n tartean. i aldagai berri hori iterazioaren indizea izango da bektorea korritzeko, eta x aldagaia zerorekin hasieratu behar dugu lehenengo iterazioa exekutatu baino lehenago inbariantea betetzeko. i aldagaiak propietate oso zehatza betetzen du inbariantean: *i da batu den azken elementuaren indizea.* Gainera, borne-adierazpena $n - i$ izan daiteke, i aldagaiaren balioa gero eta handiagoa delako, eta espresio horrek bukatzeko falta dena ondo adierazten duelako.

INB_2 oso antzekoa da, baina n konstantea $i - 1$ adierazpenarekin ordekatzen da. Horregatik, i aldagaiak har ditzaken balioak aldatu behar dira, 1etik $n-1$era. Inbariante honetan i-k adierazten du *zein den batzera goazen hurrengo elementuaren indizea.* Borne-adierazpena ere egokitu egingo genuke ideia hori hobeto islatzeko: $(n + 1) - i$.

INB_3 ere antzekoa da, eta INB_1-ekin duen alde bakarra da i aldagaiak hartzen duen balio txikiena 1 dela. Inbariantea lehenengo aldiz beteko bada aldagaien hasieraketa aldatu beharra dago: x aldagaia ez da 0 balioarekin hasieratuko, baizik eta $A(1)$ elementuarekin. Kontuan izan orain batu den azken elementuaren indizea dela i. Borne-adierazpena $n - i$ izango da.

Laburbilduz, aurreko hiru kasuetan n konstantea aldagai batekin ordezkatzen da inbariantea lortzeko. Post-baldintzatik inbariantea asmatzeko teknika emankorra da konstante bat aldagai batez ordezkatzea.

Badira beste teknika orokor batzuk, post-baldintza ahulduz beti ere, inbariantea asmatzeko aukera ematen dutenak. Hona hemen horietako batzuk:

- *ψ-ko espresio baten ordez (konstantea izan daiteke) beste espresio orokorrago bat (askotan aldagaia izaten da) jartzea inbariantean.* Inbariantera ekartzen den aldagai berriaren barrutia ondo definitu behar da. Bektoreko elementuen batura kalkulatzen duen programari buruz lortu ditugun hiru inbarianteak teknika honetan oinarritu dira.
- *Konjuntzio bat kentzea.*
- *Bien konbinazioa.* Konjuntzio bat kendu eta, horrez gain, espresio baten ordez beste orokorrago bat jartzea.
- *Post-baldintzaren eta aurre-baldintzaren konbinazioa ahultzea.*

Adibide gehiago ikusiko ditugu teknika horien erabilera hobeto azaltzeko.

- **Post-baldintzako espresiobaten ordez, inbariantean beste espresio orokorrago bat jartzea**
  - $A(1..n)$ bektoreko elementu maximoa bilatzen duen programa. Ondoko baldintza hau izan daiteke:

    $$\psi \equiv m = max(A(1..n))$$

    Inbariantea asmatzeko, n konstantea i aldagaiarekin ordezka daiteke, eta i aldagaiarekin barrutia ondo definitu:

    $$INB \equiv m = max(A(1..i)) \wedge 1 \leq i \leq n$$

    Kasu honetan, i aldagaia aztertu den azkeneko elementuaren indizea da inbariantean.

– $A(1..n)$ bektoreko zenbaki negatiboen kopurua bueltatzen duen programa:

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

Inbariantea lortzeko n konstantearen ordez i adierazpena jar daiteke inbariantean:

$$\mathbf{INB} \equiv neg = \mathcal{N}j\ (\ 1 \leq j \leq i\ \wedge\ A(j) < 0\ )\ \wedge\ 0 \leq i \leq n$$

Inbariante honetan, aztertutako azken elementuaren indizea da i.

- **Konjuntzio bat kentzea**

  – x zenbakira behetik gehien hurbiltzen den 2-ren berretura itzultzen duen programa:

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

Inbariantea ateratzeko azken konjuntzioa kentzearekin nahikoa da:

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

– $A(1..n)$ bektorean x-ren lehenengo agerpenaren indizea bueltatzen duen programa. Post-baldintza:

$$\psi\ \equiv\ 1 \leq i \leq n+1\ \wedge\ x \notin A(1..i-1)\ \wedge\ (\ i = n+1\ \vee\ x = A(i)\ )$$

Kasu honetan, inbariantea horrela gelditzen da azken konjuntziorik gabe:

$$INB\ \equiv\ 1 \leq i \leq n+1\ \wedge\ x \notin A(1..i-1)$$

- **Konjuntzio bat kentzea eta espresio baten ordez beste orokorrago bat jartzea**

  – $A(1..n)$ bektoreko elementurik handiena bikoitia ote den erabakitzen duen programa. Post-baldintza honako hau izan daiteke:

$$\psi\ \equiv\ max\_bikoitia\ \leftrightarrow\ max(A(1..n))\ mod\ 2 = 0$$

Asertzio hori bi zatitan bana daiteke, suposatuz m aldagai batek bektoreko elementu maximoa gordetzen duela:

$$\psi \equiv (max\_bikoitia \leftrightarrow m \; mod \; 2 = 0) \wedge m = max(A(1..n))$$

Inbariantea ateratzeko lehenengo zatia ezabatzen da eta bigarrenean n konstantea i aldagaiarekin ordezkatzen da. Aldagai berriaren barrutia definitu egin behar da:

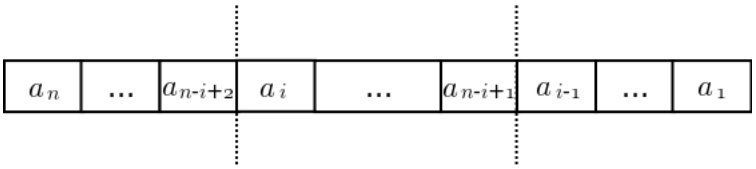
$$INB \equiv m = max(A(1..i)) \wedge 1 \leq i \leq n$$

- **Post-baldintzaren eta aurre-baldintzaren konbinazioa ahultzea**
  - $A(1..n)$ bektorea alderanztuta bueltatzen duen programa. Programa honen aurre- eta post-baldintzak honakoak dira:

$$\begin{aligned} \phi &\equiv A(1..n) = (\, a_1,\, a_2,\, \ldots,\, a_{n-1},\, a_n \,) \\ &\equiv \forall j \; (\, 1 \leq j \leq n \rightarrow A(j) = a_j \,) \end{aligned}$$

$$\begin{aligned} \psi &\equiv A(1..n) = (\, a_n,\, a_{n-1},\, \ldots,\, a_2,\, a_1 \,) \\ &\equiv \forall j \; (\, 1 \leq j \leq n \rightarrow A(j) = a_{n-j+1} \,) \end{aligned}$$

Programak binaka trukatzen ditu elementuak: lehenengoa azkenarekin, bigarrena azkenaurrekoarekin, eta abar. Hala, prozesuan zeharreko bektoreak egitura hau du:



Emandako azalpenaren eta irudiaren arabera, asma dezakegu zein den programa honen inbariantea:

$$\begin{aligned} INB \equiv \; & 1 \leq i \leq \frac{n}{2} + 1 \; \wedge \\ & \underbrace{A(1..i-1) = (\, a_n,\, a_{n-1},\, \ldots,\, a_{n-i+2} \,)}_{\text{Post-baldintza ahulduz}} \; \wedge \\ & \underbrace{A(i..n-i+1) = (\, a_i,\, a_{i+1},\, \ldots,\, a_{n-i+1} \,)}_{\text{Aurre-baldintza ahulduz}} \; \wedge \\ & \underbrace{A(n-i+2..n) = (\, a_{i-1},\, a_{i-2},\, \ldots,\, a_1 \,)}_{\text{Post-baldintza ahulduz}} \end{aligned}$$

Ikusten denez, aurreko eta ondoko baldintzak ahuldu dira prozesu hori ondo deskribatzeko inbariantean.

3.9 Programa errekurtsiboak

Programa errekurtsiboek aukera ematen dute prozesu errepikakorrak inplementatzeko. Badira, hortaz, bi alternatiba: egitura iteratiboak eta egitura errekurtsiboak.

Programa errekurtsibo bat bere buruan oinarritzen da. Formari erreparatuta, bere buruari egiten dio dei, eta auto-dei horietako bakoitzari *errekurtsibo* esaten zaio. Dei errekurtsiboen kateaketan garrantzitsua da soluziotik gero eta gertuago egotea. Hori da, azken batean, bukaeraren bermea. Soluziotik gero eta gertuago egote hori datuen sinpletasunarekin lotzen da: zenbat eta gertuago bukaeratik, orduan eta datu sinpleagoak. Hori horrela, dei errekurtsiboak egin ahala sarrerako parametroak (datuak) sinpleagotzen doaz, gero eta sinpleago dei batetik bestera. Dei errekurtsiboetan, hortaz, izaera bereko problema ebazten da, eta ebazpen partzial hori da oinarria datu konplexuagoen emaitza osatzeko.

Esandakoetatik ondoriozta daiteke edozein programa errekurtsibok gutxienez bi adar (baldintzazko) eduki behar dituela:

- *Kasu nabaria* (ez-errekurtsiboa). Gutxienez bat izango da, baina gehiago ere izan daitezke.
- *Kasu induktiboa* (errekurtsiboa). Gutxienez bat, baina ez dago oztoporik baino gehiago izateko ere.

Kasu nabarietan dei errekurtsiborik gabe ebazten diren datuen azpimultzoak tratatzen dira. Kasu induktiboak ebazteko, berriz, kasu nabarietatik gertuago dauden beste datu batzuen ebazpenean hartzen da oinarri. Bi kasu mota hauen konbinazioak matematikako formulazio induktiboan du oinarria.

Bere buruari deitu ahal diezaion, behar-beharrezkoa da azpiprograma errekurtsiboa ondo izendatzea eta parametrizatzea. Irakasgai honetan, notazioa errazteko xedearekin, bi murriztapen ezarriko ditugu azpiprograma errekurtsiboen gainean: batetik, beti funtzioak erabiliko ditugu (inoiz ez prozedurak), eta, bestetik, emaitza bakarra bueltatuko dute beti funtzioek. Notazio hau erabiliko dugu

$$\underline{\texttt{function}}\ \texttt{f(x}_\texttt{1}\texttt{:T}_\texttt{1}\texttt{; ...; x}_\texttt{n}\texttt{:T}_\texttt{n}\texttt{)}\ \underline{\texttt{return}}\ \texttt{y:R}\ \underline{\texttt{is}}$$

non f izeneko funtzioa adierazten den, T_i motetako x_i datuak eta R motako y emaitza dituena.

Adibidez, zenbaki baten faktoriala kalkulatzen duen programa horrelakoa izan daiteke:

```
function fakt(n:integer) return f:integer is
{ n ≥ 0 }
   if n = 0 then
      f := 1;
   else
      f := fakt(n-1);
      f := n*f;
   end if;
end fakt;
{ f = n! }
```

Programa horretan kasu nabari bakar bat dago, $n = 0$ baldintzarekin eta `f := 1;` aginduarekin, eta kasu errekurtsiboa ere bakarra da, $n \neq 0$ ($n = 0$ baldintzaren ezeztapena) eta `f := fakt(n-1); f := n*f;` agindu-segidarekin. Adar errekurtsibo horretan, `f := fakt(n-1);` da dei errekurtsiboa.

Programa errekurtsiboen egiaztapena indukzioan oinarritzen da. Azpiprograma errekurtsibo bat emanda

$$\underline{\texttt{function}}\ \texttt{f(x}_\texttt{1}\texttt{:T}_\texttt{1}\texttt{; ...; x}_\texttt{n}\texttt{:T}_\texttt{n}\texttt{)}\ \underline{\texttt{return}}\ \texttt{y:R}\ \underline{\texttt{is}}$$

f funtzioaren zuzentasuna egiaztatzeko honako baieztapena frogatu behar dugu:

$$\{\ \phi\ \}\ [\ \ \texttt{y := f(x}_\texttt{1}\texttt{, ..., x}_\texttt{n}\texttt{);}\ \ ]\ \{\ \psi\ \}$$

non ϕ eta ψ programa errekurtsiboaren aurre-ondoetako baldintzak diren. Hau da, dei errekurtsibo orokor baten zuzentasun osoa frogatuko dugu. Goazen, bada, urratsez urrats.

Zuzentasun partziala frogatzeko:

- Hasteko, kasu nabariek espezifikazioa betetzen dutela frogatzen da. Horretarako, kasu nabari bakoitzeko frogatu beharko duguna da

  $$\{\ \phi\ \wedge\ B_{KN}\ \}\ \ \mathbf{S_{KN}}\ \ \{\ \psi\ \}$$

  non B_{KN} kasu nabariaren baldintza izango den eta S_{KN} kasu horretan exekutatzen den agindu-segida.

- Ondoren, dei errekurtsiboek espezifikazioa betetzen dutela suposatuz (*indukzio-hipotesia*), kasu induktiboek ere espezifikazioa betetzen dutela frogatzen da. Hau da, kasu induktibo bakoitzeko, hurrengo baieztapena frogatu behar dugu:

  $$\{\ \phi\ \wedge\ B_{KI}\ \}\ \ \mathbf{S_{KI}}\ \ \{\ \psi\ \}$$

  non B_{KI} kasu induktiboaren baldintza izango den eta S_{KI} kasu horretan exekutatzen den agindu-segida. Kasu induktiboa denez, S_{KI} agindu-segidan gutxienez dei errekurtsibo bat egongo da. Horregatik, aurreko baieztapena frogatzeko, `w := f(t1, ..., tn);` dei errekurtsibo bakoitzeko honako indukzio-hipotesia erabiliko dugu:

  $$\textbf{(I.H.)}\ \{\ \phi^{\langle t_1,\ldots,t_n\rangle}_{\langle x_1,\ldots,x_n\rangle}\ \}\ \ \texttt{w := f(t}_\texttt{1}\texttt{, ..., t}_\texttt{n}\texttt{);}\ \ \{\ \psi^{\langle w,t_1,\ldots,t_n\rangle}_{\langle y,x_1,\ldots,x_n\rangle}\ \}$$

  non $\phi^{\langle t_1,\ldots,t_n\rangle}_{\langle x_1,\ldots,x_n\rangle}$ adierazpena ϕ-tik lortzen den $x_1,\ \ldots,\ x_n$ aldagaiak $t_1,\ \ldots,\ t_n$ terminoekin ordezkatuz, eta $\psi^{\langle w,t_1,\ldots,t_n\rangle}_{\langle y,x_1,\ldots,x_n\rangle}$ adierazpena ψ formulatik lortzen den $y,\ x_1,\ \ldots,\ x_n$ aldagaiak $w,\ t_1,\ \ldots,\ t_n$ terminoekin ordezkatuz.

Bukaeraren frogapenari *indukzioaren balidazioa* deitzen zaio eta horrela egiten da:

- Hasteko, E *borne-adierazpena* definitzen da errekurtsioaren parametroen arabera. Errekurtsioaren parametroak dira dei errekurtsiboko datu-parametroak $(x_1,\ \ldots,\ x_n)$.

- Ondoren, frogatu behar da E zenbaki arrunta dela kasu nabari bakoitzean. Hau da, kasu nabari bakoitzeko frogatu behar dugu

  $$(\ \phi\ \wedge\ B_{KN}\ )\ \rightarrow\ E \in \mathbb{N}$$

  non B_{KN} kasu nabariaren baldintza den.

- Bukatzeko, frogatu behar da E gutxitzen doala ($\mathbb{N}$-ren barne) dei errekurtsibo bakoitzean. Hau da, kasu induktibo bakoitzeko frogatu behar dugu:

$$(\phi \wedge B_{KI}) \rightarrow (E^{\langle t_1,\ldots,t_n\rangle}_{\langle x_1,\ldots,x_n\rangle} \in \mathbb{N} \wedge E > E^{\langle t_1,\ldots,t_n\rangle}_{\langle x_1,\ldots,x_n\rangle})$$

  non B_{KI} kasu induktiboaren baldintza den eta $E^{\langle t_1,\ldots,t_n\rangle}_{\langle x_1,\ldots,x_n\rangle}$ adierazpena E-tik lortzen den $x_1,\ldots,x_n$ aldagaiak $t_1,\ldots,t_n$ terminoekin ordezkatuz.

Adibidez, faktorialaren programa errekurtsiboa zuzena dela egiaztatzeko baieztapen hau frogatu behar dugu:

$$\{ \underbrace{n \geq 0}_{\phi} \}\ [\ \texttt{f := fakt(n);}\ ]\ \{ \underbrace{f = \prod_{i=1}^{n} i}_{\psi} \}$$

Frogapena:

- Kasu nabaria:

$$\{ \underbrace{n \geq 0}_{\phi} \wedge \underbrace{n = 0}_{B_{KN}} \}\ \texttt{f := 1;}\ \{ \underbrace{f = \prod_{i=1}^{n} i}_{\psi} \}$$

  1. $(n \geq 0 \wedge n = 0) \rightarrow (n = 0) \rightarrow (1 = \prod_{i=1}^{n} i)$
  2. $\{ 1 = \prod_{i=1}^{n} i \}$
     `f := 1;`
     $\{ f = \prod_{i=1}^{n} i \}$ **(AA)**
  3. $\{ n \geq 0 \wedge n = 0 \}$
     `f := 1;`
     $\{ f = \prod_{i=1}^{n} i \}$ 1, 2 eta **(ODE)**

- Kasu induktiboa.

  Frogatu beharrekoa:

$$\{ \underbrace{n \geq 0}_{\phi} \wedge \underbrace{n \neq 0}_{B_{KI}} \}\ \texttt{f := fakt(n-1); f := n * f;}\ \{ \underbrace{f = \prod_{i=1}^{n} i}_{\psi} \}$$

  Indukzio-hipotesiaren definizioa:

$$\textbf{(I.H.)}\ \{ n-1 \geq 0 \}\ \ \texttt{f := fakt(n-1);}\ \ \{ f = \prod_{i=1}^{n-1} i \}$$

Frogapena:

4. $(n \geq 0 \ \wedge \ n \neq 0) \ \rightarrow \ (n > 0) \ \rightarrow \ (n - 1 \geq 0)$
5. $\{ n - 1 \geq 0 \}$
   `f := fakt(n-1);`
   $\{ f = \prod_{i=1}^{n-1} i \}$ **(I.H.)**
6. $\{ n \geq 0 \ \wedge \ n \neq 0 \}$
   `f := fakt(n-1);`
   $\{ f = \prod_{i=1}^{n-1} i \}$ 4, 5 eta **(ODE)**
7. $(f = \prod_{i=1}^{n-1} i) \ \rightarrow \ (n \times f = \prod_{i=1}^{n} i)$
8. $\{ n \times f = \prod_{i=1}^{n} i \}$
   `f := n*f;`
   $\{ f = \prod_{i=1}^{n} i \}$ **(AA)**
9. $\{ f = \prod_{i=1}^{n-1} i \}$
   `f := n*f;`
   $\{ f = \prod_{i=1}^{n} i \}$ 7, 8 eta **(ODE)**
10. $\{ n \geq 0 \ \wedge \ n \neq 0 \}$
    `f := fakt(n-1);`
    `f := n*f;`
    $\{ f = \prod_{i=1}^{n} i \}$ 6, 9 eta **(KPE)**

- Indukzioaren balidazioa:
  - Borne-adierazpenaren definizioa: $E \equiv n$
  - Kasu nabarian ($n = 0$):

    $$(n \geq 0 \ \wedge \ n = 0) \ \rightarrow \ (n = 0) \ \rightarrow \ n \in \mathbb{N}$$

  - Kasu induktiboan ($n \neq 0$):

    $$(n \geq 0 \ \wedge \ n \neq 0) \ \rightarrow \ (n > 0) \ \rightarrow \ (n - 1 \in \mathbb{N} \ \wedge \ n > n - 1)$$