5. Gaia
Datu-Mota Abstraktuen ekuazio bidezko espezifikazioa

5.1. Sarrera

5.1.1. Helburua

Ekuazio bidezko espezifikazioaren (edo aljebra bidezko espezifikazioaren) teknika erabiliz Datu-Mota Abstraktu berriak (DMA berriak) nola definitzen diren aztertzea da gai honetako helburua.

Datu-mota berri bat definitzeko jarraian aipatzen direnak zehaztu behar dira

- Mota horretako balioak zein diren.
- Mota horretako balioekin buru daitezkeen eragiketak.
- Mota horrentzat definitutako eragiketek betetzen dituzten propietateak.

Adibidez, zenbaki osoei dagokien datu-motaren kasuan, balioak zenbaki osoak beraiek dira (…, -2, -1, 0, 1, 2, 3, …), eragiketak batuketa, kenketa eta abar dira eta eragiketa horien propietateek batuketa trukakorra eta elkarkorra dela, kenketa ez dela trukakorra, biderkaketa banakorra dela batuketarekiko eta abar esaten digute.

Datu-mota bateko balioak zein diren zehazteko eta balio horien gaineko eragiketak definitzeko gehienetan errekurtsibitatea erabiltzen da eta propietateak frogatzerakoan aldiz tresna garrantzitsuena indukzioa da.

5.1.2. Haskell

Haskell programazio-lengoaia funtzionala da.

Haskell lengoaian programa bat funtzio multzo bat izan ohi da. Funtzio bakoitza era independientean exekuta daiteke eta baita beste funtzio batetik deituz ere (kasu bietan parametro formalen ordez argumentu errealak ipiniz).

Haskell-en web orri ofiziala http://haskell.org da eta bertatik sistema eragile desberdinetarako Haskell plataforma instala daiteke: https://www.haskell.org/platform/. Bertsio berriena 2014koa da. Bestalde https://www.haskell.org/downloads orrian beste aukera batzuk ere badira. Gainera GHC konpilatzailea ere eskura daiteke http://www.haskell.org/ghc/ helbidean.

5.2. Haskell-en aurredefinituta dauden oinarrizko motak

5.2.1. Bool

- Balioak: {True, False}
- Eragiketak:
  - ✓ && (and)
  - ✓ || (or)
  - ✓ not (not)
- Propietateak (φ, ψ eta γ expresio boolearrak direla kontsideratuz):
  - ✓ φ && ψ ≡ ψ && φ (&& trukakorra da)
  - ✓ φ || ψ ≡ ψ || φ (|| trukakorra da)
  - ✓ not(φ && ψ) ≡ (not φ) || (not ψ)
  - ✓ not(φ || ψ) ≡ (not φ) && (not ψ)
  - ✓ φ && (ψ || γ) ≡ (φ && ψ) || (φ && γ)
  - ✓ False || φ ≡ φ (False || eragilearentzat elementu neutroa da)
  - ✓ False && φ ≡ False
  - ✓ True || φ ≡ True
  - ✓ True && φ ≡ φ (True && eragilearentzat elementu neutroa da)
  - ✓ …

5.2.2. Int

- Balioak: tarte mugatu bateko zenbaki osoak (minBound, maxBound)
- Eragile aritmetikoak:
  - ✓ - (zeinu negatiboa)
  - ✓ + (batuketa)
  - ✓ – (kenketa)
  - ✓ * (biderketa)
  - ✓ ^ (berredura, 2 ^ 3 = 8)
  - ✓ div (zatiketa osoa; idazteko bi era: 16 `div` 3 = 5 edo div 16 3 = 5, eta ` karakterea P letraren ondoan dagoen tildea da)
  - ✓ mod (zatiketa osoaren hondarra; idazteko bi era: 16 `mod` 3 = 1 edo mod 16 3 = 1, eta ` karakterea P letraren ondoan dagoen tildea da)
- Propietateak:
  - ✓ x + y = y + x Trukakortasuna
  - ✓ x + (y + z) = (x + y) + z Elkarkortasuna
  - ✓ x * (y + z) = (x * y) + (x * z) Banakortasuna
  - ✓ 0 + x = x 0 neutroa da batuketarentzat
  - ✓ 1 * x = x 1 neutroa da biderketarentzat
  - ✓ …

5.2.3. Integer

- Balioak: zenbaki osoak inolako mugarik gabe
- Eragile aritmetikoak: Int kasukoak
- Propietateak: Int kasukoak

5.2.4. Float, Double

- Balioak: zenbaki errealak
- Eragile aritmetikoak: Ez ditugu erabiliko
- Propietateak: Int kasukoen antzekoak

5.2.5. Char

- Balioak: karaktereak ('a', …, 'z', 'A', …, 'Z', …) 0-ren teklaren ondoan dagoen teklako apostrofoa erabiliz
- Eragileak: Ez ditugu erabiliko
- Propietateak: Ez ditugu aztertuko

5.2.6. Oinarrizko motentzako eragile erlazionalak

Jarraian erakusten diren eragile erlazionalak aipatu diren oinarrizko mota denentzat balio dute:

- ✓ == (berdin)
- ✓ /= (ezberdin)
- ✓ > (handiagoa)
- ✓ >= (handiagoa edo berdina)
- ✓ < (txikiagoa)
- ✓ <= (txikiagoa edo berdina)

5.2.7. Oinarrizko motentzat eragiketa berrien definizioa

Lehenago aipatu diren eragiketak erabiltzeaz gain eragiketa berriak ere defini eta erabil daitezke. Eragiketak definitzerakoan ekuazio bakoitzari zenbaki bat egokituko diogu geroago ekuazio hori erreferentziatu ahal izateko. Hiru adibide emango dira jarraian:

- **bikoitia**: zenbaki oso bat emanda, True itzuliko du bikoitia bada eta False bakoitia bada.

  Eragiketaren **mota**:
  bikoitia:: (Int) -> Bool
  Eragiketa definitzen duen **ekuazioa**:
  bikoitia(x) = (x `mod` 2) == 0 (1)

  bikoitia funtzioak x balioarentzat x `mod` 2 eta 0 konparatzearen balioa itzuliko duela adierazten da ekuazio horren bidez (konparazioaren emaitza True edo False izango da).

  Eragiketa bera definitzeko beste aukera bat honako hau izango litzateke:

```
bikoitia(x)
    | (x `mod` 2) == 0    = True     (1)
    | otherwise           = False    (2)
```

  Definizio hori honela ulertu beharko genuke:
  x `mod` 2 eta 0 berdinak badira True itzuli eta bestela False itzuli.

- **bakoitia**: zenbaki oso bat emanda, True itzuliko du zenbaki hori bakoitia bada eta False itzuliko du zenbaki hori bikoitia bada.

  Eragiketaren **mota**:
  bikoitia:: (Int) -> Bool
  Eragiketa definitzen duen **ekuazioa**:
  bakoitia(x) = not(bikoitia(x)) (1)

  Kasu honetan, x zenbaki oso bat emanda, *bakoitia* funtzioak x zenbaki horrentzat itzuliko duena *bikoitia* funtzioak x zenbaki horrentzat itzuliko lukeenaren kontrakoa dela adierazten da. Beraz, hasteko *bikoitia* funtzioak x balioarentzat itzultzen duena kalkulatuko da eta gero balio horren aurkakoa hartuko da *not* eragilearen bidez.

  Adibide honetan ikusten den bezala, funtzio berri bat definitzerakoan aurretik definituta dauden funtzioak ere erabil daitezke.

- **hand3**: hiru zenbaki oso emanda handiena itzuliko du

  Eragiketaren **mota**:
  hand3:: (Int, Int, Int) -> Int
  Eragiketa definitzen duten **ekuazioak**:

```
hand3(x, y, z)
    | x >= y && x >= z      = x      (1)
    | y > x && y >= z       = y      (2)
    | otherwise             = z      (3)
```

  Definizio hori honela ulertu beharko genuke:
  x >= y && x >= z betetzen bada, orduan x balioa itzuli, bestela y > x && y >= z betetzen bada, orduan y itzuli eta bestela z.

  Adibide honetan bezala baldintza-zerrenda bat agertzen denean, baldintzak goitik behera aztertuko dira Haskell-en eta betetzen den lehenengo baldintzaren kasuari dagokion emaitza itzuliko da.

5.3. Zerrendak

Zerrenda bat jarraian erakusten den eran adierazten da: [2, 7, 35, -8, 0]
Zerrenda hori zenbaki osoz osatuta dago eta bere mota [Int] da. Zerrenda hutsa honako era honetan adierazten da: []

Zerrenden mota parametrodun mota da, [t], eta t parametroaren arabera balio boolearrez osatutako zerrendak, zenbaki osoz osatutako zerrendak eta abar eduki ditzakegu. Beraz aurretik definitutako edozein mota erabiliz mota horretako elementuz osatutako zerrendak defini daitezke baina zerrenda bateko elementu denak mota berekoak izan behar dute.

Adibideak:
[True, True, True, False, True] Bere mota [Bool] da (boolearrez osatutako zerrenda)
[[0, 5], [], [77, -50, 3]] Bere mota [[Int]] da (osoez osatutako zerrendez eratutako zerrenda)

5.3.1. Eragiketa eraikitzaileak zerrendentzat

Datu-mota bati dagozkion eragiketa eraikitzaileak mota horretako balioak zein diren adierazteko dira.

Edozein zerrenda jarraian aipatzen diren bi eragileak erabiliz eraiki daiteke:

- [] **Zerrenda hutsa sortu**
- : **Ezkerretik elementu bat erantsi**

Lehenengo eragiketak zerrenda hutsa sortzeko balio du eta bigarrenak ezkerretik elementuak banan-banan erantsiz edo sartuz joateko balio du.

Esate baterako [6, 10] zerrenda honako era honetan eraikiko litzateke: 6:10:[]
Hasteko zerrenda hutsa sortuko litzateke, gero 10 zenbakia gehituko litzaioke zerrenda hutsari eta bukatzeko 6 balioa gehituko litzaioke dagoeneko 10 zenbakia duen zerrendari.

[] → [10] → [6, 10]

[2, 7, 35, -8, 0] zerrenda honako era honetan eraikiko litzateke: 2:7:35:-8:0:[]

Haskell-entzat [6,10] eta 6:10:[] baliokideak dira, biak onartzen ditu, baina espezifikazio ekuazionalaren bidez eragiketa berriak definitzeko eta eragiketa berri horien propietateak frogatzeko bigarren aukera egokiagoa da.

Edozein zerrenda jarraian aipatzen diren bi eskema hauetakoren batera egokitzen da:

- ✓ [] (zerrenda hutsa da)
- ✓ x:s (ez da zerrenda hutsa eta bere lehenengo elementua x da eta beste elementu denek s azpizerrenda osatzen dute).

[2, 7, 35, -8, 0] zerrenda 2:7:35:-8:0:[] eran idatz daiteke
x: s
s azpizerrenda 7:35:-8:0:[] zerrenda da

Zerrenda batek bi elementu edo gehiago dituenean

x:z:s

erakoa dela ere esan daiteke, komeni bada. Lehenengo elementua **x** izango da, bigarrena **z** eta gainontzeko elementuek **s** azpizerrenda osatzen dute.

[2, 7, 35, -8, 0] es 2:7:35:-8:0:[]
x:z:s

Eragiketa eraikitzaileen motak honako hauek dira:

```
[] :: [t]
: :: (t, [t]) -> [t]
```

Lehenengo eragiketaren motak, hau da, []-en motak, eragiketa horrek geuk nahi dugun motako (t motako) zerrenda hutsa sortzen duela adierazten du.

Bigarren eragiketaren kasuan, : eragileari t motako elementu bat eta t motako zerrenda bat emanda t motako zerrenda bat itzultzen duela adierazten da. Eragiketen motak eta elementuen motak ematerakoan **::** notazioa erabiltzen da. Funtzioetan sarrerako datuak parentesi artean joango dira komaz bereiztuta eta emaitza -> sinboloen ondoren joango da. Sarrerako daturik ez badago, emaitzaren mota bakarrik emango da, [] eragilearen kasuan bezala. Funtzioen kasuan gerta daiteke sarrerako daturik ez egotea, datu bakarra egotea edo datu bat baino gehiago egotea, baina beti emaitza bat egongo da.

5.3.2. Eragiketa ez-eraikitzaile batzuk zerrendentzat

Jarraian zerrendekin kalkuluak burutzeko balio duten eragiketa batzuk definituko dira. Funtzio bat edo eragiketa bat definitzeko, funtzioaren mota eta funtzioak zer egiten duen zehazten duten ekuazioak eman behar dira.

- **leh**: Zerrenda bat emanda, zerrendako lehenengo elementua itzuliko du. Zerrenda hutsa bada, errorea sortuko da.

  Adibideak:

  ```
  leh([4, 7, 8]) = 4
  leh([False, True]) = False
  leh([[0, 5], [], [77, -50, 3]]) = [0, 5]
  ```

  Eragiketaren **mota**:

  ```
  leh:: ([t]) -> t
  ```

  Eragiketa definitzen duten **ekuazioak**:

  ```
  leh([]) = error "Zerrenda hutsa. Ez dago lehenengo elementurik."  (1)
  leh(x:s) = x                                                      (2)
  ```

  Errore-mezu bat aurkezteko Haskell-eko *error* funtzioa erabiltzen da eta zein mezu aurkeztu nahi den zehaztu beharko da.

- **hond**: Zerrenda bat emanda, zerrendako lehenengo elementua kenduz lortzen den zerrenda itzuliko du. Zerrenda hutsa bada, errorea sortuko da.

  Adibideak:
  hond([4, 7, 8]) = [7, 8]
  hond([False, True]) = [True]
  hond([[0, 5], [], [77, -50, 3]]) = [[], [77, -50, 3]]

  Eragiketaren **mota**:
  hond:: ([t]) -> [t]

  Eragiketa definitzen duten **ekuazioak**:
  hond([]) = error "Zerrenda hutsa. Ez dago hondarrik." (1)
  hond(x:s) = s (2)

  Zerrenda hutsari dagokion errore-mezua aurkezteko Haskell-eko *error* funtzioa erabili da eta aurkeztu nahi den mezua zein den zehaztu da komatxoen artean.

- **hutsa_da**: Zerrenda bat emanda, True itzuliko du zerrenda hutsa bada eta False itzuliko du zerrenda hutsa ez bada.

  Adibideak:
  hutsa_da([]) = True
  hutsa_da([4, 7, 8]) = False
  hutsa_da([False, True]) = False

  Eragiketaren **mota**:
  hutsa_da:: ([t]) -> Bool

  Eragiketa definitzen duten **ekuazioak**:
  hutsa_da([]) = True (1)
  hutsa_da(x:s) = False (2)

  Emandako zerrenda hutsa bada (hau da, [] erakoa bada), orduan True itzuliko da. Emandako zerrenda hutsa ez bada (hau da, x:s erakoa bada, x zerrendako lehenengo elementua izanda eta s zerrendako gainontzeko elementuez osatutako azpizerrenda izanda), orduan False itzuliko da.

- **badago**: t motako elementu bat eta t motako zerrenda bat emanda, True balioa itzuliko du elementua zerrendan baldin badago eta False itzuliko du ez badago.

  Adibideak:

  badago(8, []) = False
  badago(8, [4, 8, 7, 8]) = True
  badago(False, [False, True, True]) = True
  badago([10, 9], [[0, 5], [], [77, -50, 3]]) = False

  Eragiketaren **mota**:

  badago:: (t, [t]) -> Bool

  Eragiketa definitzen duten **ekuazioak**:

  badago(x, []) = False (1)
  badago(x, y:s)
  | x == y = True (2)
  | x /= y = badago(x, s) (3)

  Ekuazio horien bitartez x elementua zerrenda batean agertzen al den ala ez erabakitzen da. Zerrenda hutsa bada (hau da, [] erakoa bada), x ez da hor egongo eta False itzultzen da zuzenean. Zerrenda hutsa ez bada, (hau da, y:s erakoa bada, zerrendako lehenengo elementua y dela eta zerrendako beste elementu denez osatutako azpizerrenda s dela kontsideratuz) x balioa y:s zerrendako lehenengo elementuaren berdina bada (hau da, x balioa y-ren berdina bada) erantzuna True da zuzenean baina x balioa y:s zerrendako lehenengo elementuaren berdina ez bada (hau da x eta y berdinak ez badira), orduan x elementua s azpizerrendan ba al dagoen begiratu beharko da.

- **++**: t motako bi zerrenda emanda, bata besteari erantsiz edo biak elkartuz lortzen den zerrenda itzuliko du.

  Adibideak:

  [] ++ [] = []
  [4, 8, 7, 8] ++ [] = [4, 8, 7, 8]
  [1, 8] ++ [4, 8, 7] = [1, 8, 4, 8, 7]
  [False, True, True] ++ [False, False] = [False, True, True, False, False]

  Eragiketaren **mota**:

  ++:: ([t], [t]) -> [t]

  Eragiketa definitzen duten **ekuazioak**:

  [] ++ s = s (1)
  (x:r) ++ s = x:(r ++ s) (2)

  Bigarren ekuazioak x:r eta s zerrendak elkartzeko lehenengo r eta s zerrendak elkartu eta gero r ++ s zerrenda berriari x elementua ezkerretik gehitu behar zaiola dio.

- **luzera**: t motako zerrenda bat emanda, elementu-kopurua (zerrenda horretan zenbat elementu dauden) itzuliko du.

  Adibideak:

  luzera([]) = 0
  luzera([5, 8, 7, 8]) = 4
  luzera([False, True, True]) = 3
  luzera([[0, 5], [], [77, -50, 3]]) = 3

  Eragiketaren **mota**:

  luzera:: ([t]) -> Int

  Eragiketa definitzen duten **ekuazioak**:

  luzera([]) = 0 (1)
  luzera(x:r) = 1 + luzera(r) (2)

  Emandako zerrenda hutsa bada (hau da, [] erakoa bada), orduan luzera edo elementu-kopurua 0 da. Emandako zerrenda hutsa ez bada (hau da, x:r erakoa bada), zerrendak gutxienez x elementua du eta gero r azpizerrendako elementu-kopurua zenbatu beharko da.

- **bikop**: Int motako zerrenda bat emanda, zerrendan zenbat elmentu bikoiti dauden itzuliko du.

  Adibideak:

  bikop([]) = 0
  bikop([5, 8, 7, 8]) = 2
  bikop([7, 11, 9]) = 0
  bikop([0, 3, 3, 5]) = 1

  Eragiketaren **mota**:

  bikop:: ([Int]) -> Int

  Eragiketa definitzen duten **ekuazioak**:

  bikop([]) = 0 (1)
  bikop(x:r)
  | bikoitia(x) = 1 + bikop(r) (2)
  | bakoitia(x) = bikop(r) (3)

  Emandako zerrenda hutsa bada (hau da, [] erakoa bada), 0 elementu bikoiti daude zerrenda horretan. Emandako zerrenda hutsa ez bada, hau da, adibidez x:r erakoa bada (zerrendako horretako lehenengo elementua x dela eta zerrendako beste elementu denez osatutako azpizerrenda r dela kontsideratuz), zerrendako lehenengo elementua bikoitia bada (hau da, x bikoitia bada) bikoiti-kopurua 1 gehi "r azpizerrendako bikoiti-kopurua" izango da. Baina zerrendako lehenengo elementua bakoitia bada (hau da, x bakoitia bada) bikoiti-kopurua "r azpizerrendako bikoiti-kopurua" izango da, x ez da zenbatu behar eta, ez baita bikoitia.

bikop funtzioaren definizioan lehendik definituta ditugun *bikoitia* eta *bakoitia* funtzioak erabili dira. Funtzio berriak definitzerakoan lehendik definituta dauden funtzioak erabil daitezke laguntzaile bezala.

- **tartekatu**: t motako bi zerrenda emanda, bi zerrenda horietako elementuak tartekatuz lortzen den zerrenda itzuliko du funtzio honek. Lehenengo zerrendako lehenengo elementua zerrenda berriko lehenengo elementua izango da, bigarren zerrendako lehenengo elementua zerrenda berriko bigarren elementua izango da eta abar. Hasierako zerrenda biak luzera berekoak ez badira, errorea gertatuko da (errore-mezua aurkeztuko da).

  Adibidea:
  tartekatu([**7, 5, 4**], [8, 2, 0]) = [**7**, 8, **5**, 2, **4**, 0]

  Eragiketaren **mota**:
  tartekatu:: ([t], [t]) -> [t]
  Eragiketa definitzen duten **ekuazioak**:

```
tartekatu([], s)
   | luzera(s) /= 0        = error "Luzera desberdineko zerrendak."      (1)
   | otherwise             = []                                          (2)

tartekatu(x:r, s)
   | luzera(x:r) /= luzera(s)    = error "Luzera desberdineko zerrendak." (3)
   | otherwise                   = x:(leh(s): tartekatu(r, hond(s)))      (4)
```

  Kasu honetan 4 ekuazio eman dira *tartekatu* funtzioa definitzeko.

  Lehenengo bi ekuazioetan lehenengo zerrenda hutsa denean ([] erakoa denean) zer egin behar den zehazten da. Bigarren zerrenda (s zerrenda) hutsa ez bada, zerrenda bien luzera desberdina izango denez, errore-mezua aurkeztuko da eta bestela, hau da, bigarren zerrendaren luzera [] zerrendaren luzeraren berdina denean, zerrenda biak hutsak direnez emaitza bezala ere zerrenda hutsa itzuliko da.

  Beste bi ekuazioetan lehenengo zerrenda hutsa ez denean (adibidez x:r erakoa denean, zerrendako lehenengo elementua x dela eta zerrendako gainontzeko elementuez osatutako azpizerrenda r dela kontsideratuz), zer egin behar den zehazten da. Hor (3) ekuazioan x:r zerrendaren luzera eta s zerrendaren luzera desberdinak badira errore-mezua aurkeztuko dela adierazten da. Laugarren ekuazioan x:r eta s zerrendek luzera bera dutenean bi zerrenda horietako elementuak tartekatuz osatutako zerrenda nola eraikitzen den zehazten da. Zerrenda berriko lehenengo elementua x:r zerrendako lehenengo elementua izango da (x elementua). Zerrenda berriko bigarren elementua s zerrendako lehenengo elementua izango da, baina elementu horrek izen berezirik ez duenez, *leh(s)* bezala adierazi behar da. Zerrenda berriaren gainontzeko zatia kalkulatzeko, lehenengo zerrendako (x:r zerrendako) gainontzeko elementuak, hau da, r azpizerrendako elementuak eta bigarren zerrendako (s zerrendako) gainontzeko elementuak tartekatu beharko dira. Baina s zerrendako gainontzeko elementuez osatutako azpizerrendari izenik ez diogunez eman, *hond(s)* bezala adierazi beharko da azpizerrenda hori.

- **tartekatu2**: t motako bi zerrenda emanda, bi zerrenda horietako elementuak tartekatuz lortzen den zerrenda itzuliko du baina kasu honetan lehenengo zerrendako lehenengo elementua zerrenda berriko bigarren elementua izango da, bigarren zerrendako lehenengo elementua zerrenda berriko lehenengo elementua izango da eta abar. Hasierako zerrenda biak luzera berekoak ez badira, errorea gertatuko da (errore-mezua aurkeztuko da).

  Adibidea:

  tartekatu2([**7, 5, 4**], [8, 2, 0]) = [8, **7**, 2, **5**, 0, **4**]

  Eragiketaren **mota**:
  tartekatu2:: ([t], [t]) -> [t]

  Eragiketa definitzen duten **ekuazioak**:

```
tartekatu2([], s)
   | luzera(s) /= 0          = error "Luzera desberdineko zerrendak."   (1)
   | otherwise        = []                                              (2)

tartekatu2(x:r, s)
   | luzera(x:r) /= luzera(s)    = error "Luzera desberdineko zerrendak."   (3)
   | otherwise                   = leh(s):(x: tartekatu2(r, hond(s)))       (4)
```

  Ekuazio hauek ematerakoan aurreko adibideko, hau da, *tartekatu* izeneko funtzioaren kasuko ideia bera jarraitu da, baina (4) ekuazioan zerrenda berria eraikitzerakoan elementuak kontrako ordenean ipini dira lortu nahi den zerrenda lortu ahal izateko. Hori da hain zuzen ere *tartekatu* eta *tartekatu2* izeneko funtzioen arteko desberdintasuna.

5.3.4. Zerrendentzako eragile erlazionalak

Oinarrizko datu-moten atalean aipatutako eragile erlazionalak zerrendekin ere erabil daitezke.

Zerrenda bat beste bat baino txikiagoa edo handiagoa al den erabakitzeko hitzak alfabetikoki ordenatzeko erabiltzen den teknika bera jarraitzen da.

Esate baterako [3, 3, 4] zerrenda [5, 1] zerrenda baino txikiaga da 3 zenbakia 5 baino txikiagoa delako.

5.3.5. Propietateen frogapen formala: Indukzioa

Zerrendentzat definitutako eragiketek betetzen dituzten propietateak normalean indukzioa (zerrenden egituran oinarritutako indukzioa) erabiliz frogatzen dira. Atal honetan adibide batzuk azalduko ditugu eragiketek propietate batzuk betetzen dituztela nola frogatzen den ikusteko.

1. Adibidea

Edozein s zerrendak honako propietate hau beteko du:

$$\text{luzera}(s) \geq 0$$

Indukzioa s zerrendarekiko burutuko da.

- Oinarrizko kasua: s zerrenda hutsa da, hau da, s = []

  $$\text{luzera}([]) \geq 0?$$

  luzera([]) = 0 (luzera eragiketaren lehenengo ekuazioagatik)
  Beraz luzera(s) ≥ 0 bete egiten da s zerrenda hutsa denean.

- Kasu induktiboa edo orokorra: s ez da zerrenda hutsa, hau da, s = x:r

  $$\text{luzera}(x:r) \geq 0?$$

  Kasu honetan r zerrendaren luzera 0ren berdina edo handiagoa dela suposatuz x:r zerrendaren luzera 0ren berdina edo handiagoa dela frogatu behar dugu.

  - **Indukzio-hipotesia** (ih): r zerrendak propietatea bete egiten duela suposatuko dugu, hau da, luzera(r) ≥ 0 dela.
  - **Indukzio-urratsa:** r zerrendak propietatea betetzen duela suposatuz x:r zerrendak ere propietatea bete egiten duela frogatu behar da.

    luzera(x:r) = (luzera-ren bigarren ekuazioagatik)
    = (1 + luzera(r))

    (1 + luzera(r)) ≥ 0 al da? Bai. Indukzio hipotesiagatik badakigu luzera(r) 0 edo handiagoa dela eta ondorioz 1 + luzera(r) balioa 1 edo handiagoa izango da eta beraz ≥ 0 ere izango da.

2. adibidea

Orain s eta r edozein bi zerrenda direla kontsideratuz, honako propietate hau betetzen dutela frogatuko dugu indukzioaren bidez:

$$\text{luzera}(s \mathbin{++} r) = \text{luzera}(s) + \text{luzera}(r)$$

Indukzioa s zerrendari aplikatuko zaio.

- Oinarrizko kasua: s zerrenda hutsa da, hau da, s = []

  luzera([] ++ r) = luzera([]) + luzera(r)?

  Ekuazioaren alde biek balio bera dutela frogatu behar da.

  luzera([] ++ r) = (++ eragilearen 1. ekuazioagatik)
  = luzera(r)

  luzera([]) + luzera(r) = (luzera eragiketaren 1. ekuazioagatik)
  = 0 + luzera(r) = (batuketarentzat 0 neutroa delako)
  = luzera(r)

  Ekuazioaren alde bietan emaitza bera lortzen da eta ondorioz s zerrenda hutsa denean propietatea bete egiten dela frogatu dugu.

- Kasu induktiboa: s ez da zerrenda hutsa, hau da, s = x:w

  Honako hau frogatu behar dugu

  luzera((x:w) ++ r) = luzera(x:w) + luzera(r).

  - **Indukzio-hipotesia** (ih): w eta r-rentzat propietatea bete egiten dela suposatuko dugu

    luzera(w ++ r) = luzera(w) + luzera(r)

  - **Indukzio-urratsa:** w eta r zerrendentzat propietatea bete egiten dela suposatuz, x:w eta r zerrendentzat ere bete egiten dela frogatuko dugu. Horretarako ekuazioaren alde biek balio bera dutela egiaztatu beharko da:

    luzera((x:w) ++ r) = (++ eragilearen 2. ekuazioa erabiliz)
    = luzera(x:(w ++ r)) = (luzera eragiketaren 2. ekuazioa erabiliz)
    = **1 + luzera(w ++ r)**

    luzera(x:w) + luzera(r) = (luzera eragiketaren 2. ekuazioa erabiliz)
    = (1 + luzera(w)) + luzera(r) = (batuketa elkarkorra delako)
    = 1 + (luzera(w) + luzera(r)) = (indukzio-hipotesiagatik)
    = **1 + luzera(w ++ r)**

    Ekuazioaren alde bietan gauza bera lortzen da eta ondorioz s zerrenda x:w erakoa denean ere propietatea bete egiten dela frogatu dugu.

Oinarrizko kasuan s zerrenda hutsa denean ([] erakoa denean) propietatea bete egiten dela frogatu da eta kasu induktiboan s hutsa ez denean ere (hau da, x:w erakoa denean ere) propietatea bete egiten dela frogatu da. Beraz propietatea beti betetzen dela frogatu dugu, s eta r edozein bi zerrenda izanda.

3. adibidea

Edozein balio (x) eta edozein bi zerrenda (s eta r) hartuta, jarraian adierazten den propietatea betetzen dela frogatuko dugu adibide honetan:

badago(x, s ++ r) = badago(x, s) || badago(x, r)

Indukzioa s zerrendarekiko egingo da.

- Oinarrizko kasua: s zerrenda hutsa da, hau da, s = []

  badago(x, [] ++ r) = badago(x, []) || badago(x, r)?

  Ekuazioaren alde biek balio bera dutela egiaztatu behar da.

  badago(x, [] ++ r) = (++ eragilearen 1. ekuazioagatik)
  = **badago(x, r)**

  badago(x, []) || badago(x, r) = (badago eragiketaren 1. ekuazioagatik)
  = False || badago(x, r) = (|| eragilearentzat False neutroa delako)
  = **badago(x, r)**

  Ekuazioaren alde bietan emaitza bera lortzen da eta ondorioz s zerrenda hutsa denean propietatea bete egiten dela frogatu dugu.

- Kasu induktiboa: s ez da zerrenda hutsa, hau da, s = z:w

  Honako hau frogatu behar da

  badago(x, (z:w) ++ r) = badago(x, z:w) || badago(x, r)

  - **Indukzio-hipotesia** (ih): w eta r zerrendentzat propietatea bete egiten dela suposatuko dugu

    badago(x, w ++ r) = badago(x, w) || badago(x, r)

  - **Indukzio-urratsa:** w eta r zerrendek propietatea bete egiten dutela suposatuz, z:w eta r zerrendek ere bete egiten dutela frogatu behar da. Horretarako ekuazioaren alde biek balio bera dutela egiaztatu behar da. Kasu honetan *badago* eragiketa definitzerakoan bigarren parametro bezala agertzen den zerrenda hutsa ez denean bi azpikasu bereiztu dira eta ondorioz bi ekuazio eman behar izan dira. Azpikasu bakoitzeko froga aparte egin beharko da.

    (2) x = z

    badago(x, (z:w) ++ r) = (++ eragilearen 2. ekuazioa erabiliz)
    = badago(x, z:(w ++ r)) = (badago eragilearen 2. ekuazioagatik)
    = **True**

badago(x, z:w) || badago(x, r) = (badago eragilearen 2. ekuazioagatik)
= True || badago(x, r) = (True || φ ≡ True delako)
= **True**

Ekuazioaren alde bietan emaitza bera lortzen da eta ondorioz s zerrenda z:w erakoa denean eta x = z denean propietatea bete egiten dela frogatu dugu.

(3) x ≠ z

badago(x, (z:w) ++ r) = (++ eragilearen 2. ekuazioa erabiliz)
= badago(x, z:(w ++ r)) = (badago eragilearen 3. ekuazioagatik)
= **badago(x, w ++ r)**

badago(x, z:w) || badago(x, r) = (badago eragilearen 3. ekuazioagatik)
= badago(x, w) || badago(x, r) = (indukzio-hipotesiagatik)
= **badago(x, w ++ r)**

Ekuazioaren alde bietan emaitza bera lortzen da eta ondorioz s zerrenda z:w erakoa denean eta x ≠ z denean propietatea bete egiten dela frogatu dugu.

Oinarrizko kasuan s zerrenda hutsa denean ([] erakoa denean) propietatea bete egiten dela frogatu da eta kasu induktiboan s hutsa ez denean ere (hau da, z:w erakoa denean ere) propietatea bete egiten dela frogatu da (bai x = z denean eta baita x ≠ z denean ere). Beraz propietatea beti betetzen dela frogatu dugu, x edozein balio eta s eta r edozein bi zerrenda izanda.

4. adibidea

Edozein hiru zerrenda hartuta (s, r eta q) honako hau betetzen dela frogatuko dugu:

$$(s \mathbin{++} r) \mathbin{++} q = s \mathbin{++} (r \mathbin{++} q)$$

Propietate honek ++ eragilea elkarkorra dela dio.

Indukzioa s zerrendarekiko burutuko da.

- Oinarrizko kasua: s zerrenda hutsa da, hau da, s = []

  $$([] \mathbin{++} r) \mathbin{++} q = [] \mathbin{++} (r \mathbin{++} q)?$$

  Ekuazioaren alde biek balio bera dutela egiaztatu behar da.

  ([] ++ r) ++ q = (++ eragilearen 1. ekuazioagatik)
  = **r ++ q**

  [] ++ (r ++ q) = (++ eragilearen 1. ekuazioagatik)
  = r ++ q

Ekuazioaren alde bietan emaitza bera lortzen da eta ondorioz s zerrenda hutsa denean propietatea bete egiten dela frogatu dugu.

- Kasu induktiboa: s ez da zerrenda hutsa, hau da, s = x:w

  Honako hau frogatu behar dugu:
  ((x:w) ++ r) ++ q = (x:w) ++(r ++ q)

  - **Indukzio-hipotesia** (ih): w, r eta q zerrendek propietatea bete egiten dutela suposatuko dugu

    (w ++ r) ++ q = w ++(r ++ q)

  - **Indukzio-urratsa:** w, r eta q zerrendek propietatea bete egiten dutela suposatuz, x:w, r eta q zerrendek ere bete egiten dutela frogatuko da. Horretarako ekuazioaren alde biek balio bera dutela egiaztatu behar da:

    ((x:w) ++ r) ++ q = (++ eragilearen 2. ekuazioagatik)
    = (x:(w ++ r)) ++ q = (++ eragilearen 2. ekuazioagatik)
    = **x:((w ++ r) ++ q)**

    (x:w) ++ (r ++ q) = (++ eragilearen 2. ekuazioagatik)
    = x:(w ++ (r ++ q)) = (Indukzio-hipotesiagatik)
    = **x:((w ++ r) ++ q)**

    Ekuazioaren alde bietan emaitza bera lortzen da eta ondorioz s zerrenda hutsa ez denean, hau da, x:w erakoa denean propietatea bete egiten dela frogatu dugu.

Oinarrizko kasuan s zerrenda hutsa denean ([] erakoa denean) propietatea bete egiten dela frogatu da eta kasu induktiboan s hutsa ez denean ere (hau da, x:w erakoa denean ere) propietatea bete egiten dela frogatu da. Beraz propietatea beti betetzen dela frogatu dugu, s, r eta q edozein hiru zerrenda izanda.

5. adibidea

Mota berekoak diren eta luzera bera duten edozein bi zerrenda hartuta (r eta s) honako propietate hau betetzen dela frogatuko dugu:

luzera(tartekatu(r, s)) = 2 * luzera(r)

Indukzioa r zerrendarekiko burutuko da.

- Oinarrizko kasua: r zerrenda hutsa da, hau da, r = []

  luzera(tartekatu([], s)) = 2 * luzera([])?

  Ekuazioaren alde biak kalkulatu behar dira propietatea bete egiten dela egiaztatzeko. Gogoratu r eta s zerrendek luzera bera dutela eta ondorioz r hutsa bada s ere hutsa izango da:

  luzera(tartekatu([], s)) = (tartekatu eragiketaren 1. ekuazioagatik)
  = luzera([]) = (luzera eragiketaren 1. ekuazioagatik)
  = 0

  2 * luzera([]) = (luzera eragiketaren 1. ekuazioagatik)
  = 2 * 0 =
  = 0

  Ekuazioaren alde bietan emaitza bera lortzen da eta ondorioz r zerrenda hutsa denean propietatea bete egiten dela frogatu dugu.

- Kasu induktiboa: r ez da zerrenda hutsa, hau da, r = x:w

  Honako hau frogatu behar da:

  luzera(tartekatu(x:w, s)) = 2 * luzera(x:w)

  Kasu honetan w eta hond(s) zerrendentzat propietatea bete egiten dela suposatuz, x:w eta s-rentzat ere bete egiten dela frogatu behar da. Horretarako ekuazioaren alde biak kalkulatu behar dira berdinak direla egiaztatuz.

  - **Indukzio-hipotesia** (ih): w eta hond(s) zerrendek propietatea bete egiten dutela suposatuko dugu. Kasu honetan w eta hond(s) zerrendek luzera bera izango dute:

    luzera(tartekatu(w, hond(s))) = 2 * luzera(w)

- **Indukzio-urratsa:** w eta hond(s) zerrendek propietatea bete egiten dutela suposatuz, x:w eta s zerrendek ere bete egiten dutela frogatu behar da.

  luzera(tartekatu(x:w, s)) = 2 * luzera(x:w)?

  Propietatea bete egiten dela frogatzeko ekuazioaren alde biak kalkulatu beharko dira emaitza bera ematen dutela egiaztatuz.

  luzera(tartekatu(x:w, s)) = (tartekatu eragiketaren 2. ekuazioagatik)
  = luzera(x:leh(s):tartekatu(w, hond(s))) =
  (luzera eragiketaren 2. ekuazioagatik)
  = 1 + luzera(leh(s):tartekatu(w, hond(s))) =
  (luzera eragiketaren 2. ekuazioagatik)
  = 1 + (1 + luzera(tartekatu(w, hond(s)))) =
  (batuketa elkarkorra delako)
  = 2 + luzera(tartekatu(w, hond(s)))

  2 * luzera(x:w) = (luzera eragiketaren 2. ekuazioagatik)
  = 2 * (1 + luzera(w)) = (* banakorra delako batuketarekiko)
  = 2 + 2 * luzera(w) = (Indukzio-hipotesiagatik)
  = 2 + luzera(tartekatu(w, hond(s)))

  Ikus daitekeenez, ekuazioaren alde biak garatuz gauza bera lortzen da eta ondorioz r zerrenda hutsa ez denean, hau da, x:w erakoa denean ere propietatea bete egiten dela frogatu dugu.

Beraz r zerrenda hutsa denean ([] erakoa denean) propietatea bete egiten dela frogatu da eta r hutsa ez denean ere (hau da, x:w erakoa denean ere) bete egiten dela frogatu da, ondorioz beti betetzen da. Baina propietatea zerrenda biek (r eta s) luzera bera dutenean bakarrik beteko da, izan ere luzera berekoak ez badira *tartekatu* funtzioak errorea itzuliko du. Froga egiterakoan hasieratik zerrenda biak luzera berekoak direla kontsideratu da.

5.3.6. Espresioen sinplifikazioa

Eragiketak definitzen dituzten ekuazioak eta eragiketen propietateak kontuan hartuz espresioak sinplifikatzeko aukera egoten da kasu batzuetan. Jarraian lau adibide erakusten dira eta adibide horietan sinplifikazio-urrats bakoitzean zein ekuazio edo zein propietate erabili den zehaztu da:

1. adibidea

luzera(hond (x:s)) → (hond eragiketaren 2. ekuazioagatik)
luzera(s)

2. adibidea

luzera(hond(7:90:5:[])) → (hond eragiketaren 2. ekuazioagatik)
luzera(90:5:[]) → (luzera eragiketaren 2. ekuazioagatik)
1 + luzera(5:[]) → (luzera eragiketaren 2. ekuazioagatik)
1 + (1 + luzera([])) → (batuketa elkarkorra delako)
2 + luzera([]) → (luzera eragiketaren 1. ekuazioagatik)
2 + 0 → (0 batuketaren elementu neutroa delako)
2

3. adibidea

luzera((x:s) ++ (hond(y:z:r))) → (hond eragiketaren 2. ekuazioagatik)
luzera((x:s) ++ (z:r)) → (++ eragiketaren 2. ekuazioagatik)
luzera(x:(s ++ (z:r))) → (luzera eragiketaren 2. ekuazioagatik)
1 + luzera(s ++ (z:r)) → (Aurreko ataleko 2. adibidean ikusitako propietateagatik)
1 + (luzera(s) + luzera(z:r)) → (luzera eragiketaren 2. ekuazioagatik)
1 + (luzera(s) + (1 + luzera(r))) → (batuketa elkarkorra delako)
1 + ((luzera(s) + 1) + luzera(r)) → (batuketa elkarkorra delako)
(1 + (luzera(s) + 1)) + luzera(r) → (batuketa trukakorra delako)
(1 + (1 + luzera(s))) + luzera(r) → (batuketa elkarkorra delako)
(2 + luzera(s)) + luzera(r)

4. adibidea

hond(hond(8:7:hond(1:2:[]))) → (hond eragiketaren 2. ekuazioagatik)
hond(hond(8:7:2:[])) → (hond eragiketaren 2. ekuazioagatik)
hond(7:2:[])) → (hond eragiketaren 2. ekuazioagatik)
2:[]

5.4. Pilak

Pila bat honela adierazi ohi da grafikoki:

2
7
35
-8
0

Pilen mota Haskell-en ez dago aurredefinituta.

5.4.1. Eragiketa eraikitzaileak pilentzat

Datu-mota bati dagozkion eragiketa eraikitzaileak mota horretako balioak zein diren adierazteko dira.

Datu-mota berri bat definitzeko bere eragiketa eraikitzaileak zein diren zehaztu behar da.

Pilen mota honela definituko dugu:

```
data Pila t = Phutsa | Pilaratu (t, Pila t)
```

Pilentzat eragiketa eraikitzaileak Phutsa eta Pilaratu izango dira.

Definizio horren bidez t motako pila bat pila hutsa izango dela edo bestela t motako pila baten gainean t motako elementu bat ipiniz lortutako pila bat izango dela adierazten da. Gogoratu t parametroak edozein mota ordezkatzen duela (Int, Bool, eta abar).

Phutsa eta Pilaratu eragiketa eraikitzaileen motak honako hauek dira:

```
Phutsa :: -> Pila t
Pilaratu:: (t, Pila t) -> Pila t
```

Phutsa eragiketak t motako pila bat sortzen du ezerezetik (hau da, sarrerako daturik erabili gabe). Pilaratu eragiketak, t motako elementu bat eta t motako pila bat emanda, elementua pilaren gainean ipiniz lortzen den t motako pila itzuliko du.

Har dezagun honako pila hau:

2
7
35

Pila hori honela adieraziko genuke Phutsa eta Pilaratu eragiketak erabiliz:

```
Pilaratu(2, Pilaratu(7, Pilaratu(35, Phutsa)))
```

Espresio horrek honako hau adierazten du:

Hasteko pila hutsa sortuko da:

Pila hutsaren gainean 35 zenbakia ipiniko da (hau da, 35 zenbakia pilaratuko da) eta horrela 35 zenbakia duen pila izango dugu:

35

Pila horren gainean 7 zenbakia ipiniko da, honako pila hau lortuz:

7
35

Bukatzeko pila horren gainean 2 zenbakia ipiniko da honako pila hau lortuz:

2
7
35

Pilen datu-mota parametrikoa da, t parametroaren arabera boolearrez osatutako pilak edo zenbaki osoz osatutako pilak edo zenbaki osozko zerrendez osatutako pilak, edo zenbaki osozko pilez osatutako pilak eta abar defini baititzakegu. Beraz edozein motatako pilak defini daitezke baina pila batean elementu denek mota berekoak izan behar dute.

Adibideak:

Honako hau **Pila Bool** motako pila bat da:

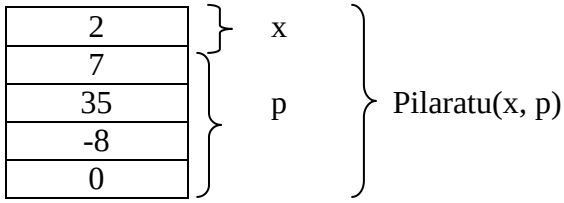
False
True
False
False

Beste hau **Pila [Int]** motakoa da

[0, 5]
[]
[77, -50, 3]

Edozein pila hartuta, pilak honako bi egitura hauetako bat izango du:

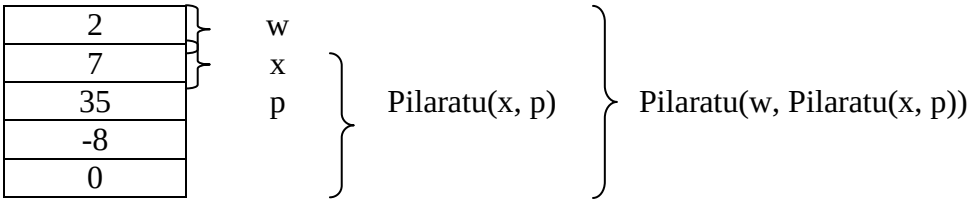
- ✓ Phutsa (pila hutsa denean)
- ✓ Pilaratu(x, p) (p pila baten gainean x elementua ipiniz lortutako pila ez hutsa da) (x eta p izenak erabili beharrean beste izen batzuk ere erabil daitezke)



Pila batek bi elementu edo gehiago dituenean honako egitura hau duela ere esan daiteke:

Pilaratu(w, Pilaratu(x, p))

Gailurreko elementua **w** izango da, gailurretik hasita bigarren elementua **x** izango da eta gainontzeko elementuek **p** azpi-pila osatzen dute.



5.4.2. Eragiketa ez-eraikitzaile batzuk pilentzat

Jarraian pilekin kalkuluak burutzeko balio duten eragiketa batzuk definituko dira. Funtzio bat edo eragiketa bat definitzeko, funtzioaren mota eta funtzioak zer egiten duen zehazten duten ekuazioak eman behar dira.

- **gailurra**: Pila bat emanda, pilaren gailurrean dagoen elementua itzuliko du funtzio honek. Pila hutsa baldin bada, errore mezu bat aurkeztuko da.

  Adibideak:
  gailurra(Pilaratu(2, Pilaratu(7, Pilaratu(35, Phutsa)))) = 2
  gailurra(Phutsa) = error

  Eragiketaren mota:
  gailurra:: (Pila t) -> t

  Eragiketa definitzen duten ekuazioak:
  gailurra(Phutsa) = error "Pila hutsa. Ez dago gailurrik." (1)
  gailurra(Pilaratu(x, p)) = x (2)

  Errore-mezu bat aurkezteko Haskell-eko *error* funtzioa erabiltzen da eta zein mezu aurkeztu nahi den zehaztu beharko da.

- **despilatu**: Pila bat emanda, pilaren gailurrean dagoen elementua kenduz gelditzen den pila itzuliko du funtzio honek. Pila hutsa baldin bada, errore mezu bat aurkeztuko da.

  Adibideak:
  despilatu(Pilaratu(2, Pilaratu(7, Pilaratu(35, Phutsa)))) =
  = Pilaratu(7, Pilaratu(35, Phutsa))
  despilatu(Phutsa) = error

  Eragiketaren mota:
  despilatu:: (Pila t) -> Pila t

  Eragiketa definitzen duten ekuazioak:
  despilatu(Phutsa) = error "Pila hutsa. Ezin da despilatu." (1)
  despilatu(Pilaratu(x, p)) = p (2)

  Errore-mezu bat aurkezteko Haskell-eko *error* funtzioa erabiltzen da eta zein mezu aurkeztu nahi den zehaztu beharko da.

- **hutsa_da**: Pila bat emanda, pila hori hutsa baldin bada, funtzioak True itzuliko du eta bestela False itzuliko du.

  Adibideak:
  hutsa_da(Phutsa) = True
  hutsa_da(Pilaratu(2, Pilaratu(7, Pilaratu(35, Phutsa)))) = False

  Eragiketaren mota:
  hutsa_da:: (Pila t) -> Bool

  Eragiketa definitzen duten ekuazioak:
  hutsa_da(Phutsa) = True (1)
  hutsa_da(Pilaratu(x, p)) = False (2)

  Emandako pila hutsa baldin bada, hau da, Phutsa egitura badu, funtzioak True itzuliko du. Emandako pila hutsa ez bada, hau da, Pilaratu(x, p) egitura badu, orduan False itzuliko du.

5.5. Ilarak

Ilaren datu-mota zerrenden antzekoa da baina ilaren kasuan elementuak bukaeran ipini behar dira (eskuineko ertzean). Ilarak grafikoki honela adieraziko ditugu:

<< 2, 7, 35, -8, 0 >>

Ilaren datu-mota ez dago definituta Haskell-en eta Haskell-ek ez du onartzen ilarak << 2, 7, 35, -8, 0 >> eran adieraztea. Beraz ilarak adierazteko era hori eragiketen esanahia grafikoki errazago azaltzeko erabiliko dugu baina gero Haskell-en ezingo da erabili.

5.5.1. Eragiketa eraikitzaileak ilara datu-motarentzat

Datu-mota bati dagozkion eragiketa eraikitzaileak mota horretako balioak zein diren adierazteko dira.

Datu-mota berri bat definitzeko bere eragiketa eraikitzaileak zein diren zehaztu behar da.

Ilara mota honela definituko dugu:

```
data Ilara t = Ihutsa | Ipini(Ilara t, t)
```

Ilara datu-motarentzat eragiketa eraikitzaileak Ihutsa eta Ipini dira.

Definizio horren bidez t motako ilara bat ilara hutsa izango dela edo bestela t motako ilara baten bukaeran t motako elementu bat ipiniz lortutako ilara bat izango dela adierazten da. Gogoratu t parametroak edozein mota ordezkatzen duela (Int, Bool, eta abar).

Ihutsa eta Ipini eragiketa eraikitzaileen motak honako hauek dira:

```
Ihutsa :: -> Ilara t
Ipini:: (Ilara t, t) -> Ilara t
```

Ihutsa eragiketak t motako ilara bat sortzen du ezerezetik (hau da, sarrerako daturik erabili gabe). Ipini eragiketak, t motako ilara bat eta t motako elementu bat emanda, elementua ilararen bukaeran (eskuineko ertzean) ipiniz lortzen den t motako ilara itzuliko du.

Har dezagun honako ilara hau:

<< 35, -8, 0 >>

Ilara hori honela adieraziko genuke eragiketa eraikitzaileen bidez:

Ipini(Ipini(Ipini(Ihutsa, 35), -8), 0)

Espresio horren bidez ilara hori sortzeko honako urratsak eman direla adierazten da:

Hasteko ilara hutsa sortu:

<< >>

Ilara hutsaren bukaeran 35 zenbakia ipini, 35 zenbakia duen ilara lortuz:

<< 35 >>

Ilara horren bukaeran 7 zenbakia ipini, honako ilara hau lortuz:

<< 35, 7 >>

Bukatzeko, ilararen bukaeran 2 zenbakia ipini eta honako ilara hau izango dugu:

<< 35, 7, 2 >>

Ilaren datu-mota parametrikoa da, t parametroaren arabera boolearrez osatutako ilarak edo zenbaki osoz osatutako ilarak edo zenbaki osozko zerrendez osatutako ilarak, edo zenbaki osozko pilez osatutako ilarak eta abar defini baititzakegu. Beraz edozein motatako ilarak defini daitezke baina ilara batean elementu denek mota berekoak izan behar dute.

Adibideak:

Honako ilara hau **Ilara Bool** motakoa da

<< True, False, False, False >>

Honako ilara hau **Ilara [Int]** motakoa da

<< [0, 5], [], [77, -50, 3] >>

Edozein ilara hartuta, ilarak honako bi egitura hauetako bat izango du:

- ✓ Ihutsa (ilara hutsa denean)
- ✓ Ipini(c, x) (ilara ez da hutsa, ilarako azken elementua x izango da eta ilarako beste elementu denek c azpi-ilara osatzen dute) (c eta p izenak erabili beharrean beste izen batzuk ere erabil daitezke)

Ilara batek bi elementu edo gehiago dituenean honako egitura hau duela ere esan daiteke:

Ipini(Ipini(c, x), w)

Ilarako azken elementua **w** da, azken aurrekoa **x** eta beste elementu denek **c** azpi-ilara osatzen dute.

5.5.2. Eragiketa ez-eraikitzaile batzuk ilarentzat

Jarraian ilarekin kalkuluak burutzeko balio duten eragiketa batzuk definituko dira. Funtzio bat edo eragiketa bat definitzeko, funtzioaren mota eta funtzioak zer egiten duen zehazten duten ekuazioak eman behar dira.

- **hasiera**: Ilara bat emanda, ilararen hasieran (ezkerreko ertzean) dagoen elementua itzuliko du. Ilara hutsa baldin bada, errore mezu bat aurkeztuko da.

  Adibideak:
  hasiera(Ipini(Ipini(Ipini(Ihutsa, 3), 8), 34), 10) = 3
  hasiera(Ihutsa) = error

  Eragiketaren mota:
  hasiera:: (Ilara t) -> t

  Eragiketa definitzen duten ekuazioak:
  hasiera(Ihutsa) = error "Ilara hutsa. Ez dago elementurik." (1)
  hasiera(Ipini(c, x))
  | c == Ihutsa = x (2)
  | otherwise = hasiera (c) (3)

  Errore-mezu bat aurkezteko Haskell-eko *error* funtzioa erabiltzen da eta zein mezu aurkeztu nahi den zehaztu beharko da.

- **ihutsa_da**: Ilara bat emanda, ilara hutsa baldin bada, True itzuliko du eta bestela False itzuliko du.

  Adibideak:
  ihutsa_da(Ihutsa) = True
  ihutsa_da (Ipini(Ipini(Ipini(Ihutsa, 23), 45), 0), 10) = False

  Eragiketaren mota:
  ihutsa_da:: (Ilara t) -> Bool

  Eragiketa definitzen duten ekuazioak:
  ihutsa_da(Ihutsa) = True (1)
  ihutsa_da(Ipini(c, x)) = False (2)

  Emandako ilara hutsa baldin bada, hau da, Ihutsa egitura badu, funtzioak True itzuliko du. Emandako ilara hutsa ez bada, hau da, Ipini(c, x) egitura badu, orduan False itzuliko du.

- **kendu**: Ilara bat emanda, ilarako hasierako (ezkerreko ertzeko) elementua kenduz gelditzen den ilara itzuliko du funtzio honek. Ilara hutsa baldin bada, errore mezua aurkeztuko da.

  Adibideak:
  kendu(Ipini(Ipini(Ipini(Ihutsa, 2), 5), 20)) =
  = Ipini(Ipini(Ihutsa, 5), 20)
  kendu(Ihutsa) = error

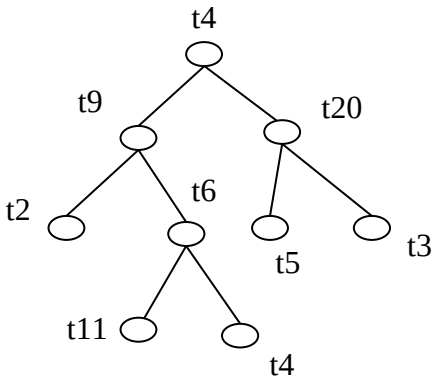
  Eragiketaren mota:
  kendu:: (Ilara t) -> Ilara t

  Eragiketa definitzen duten ekuazioak:
  kendu(Ihutsa) = error "Ilara hutsa. Ezin da elementurik kendu." (1)
  kendu(Ipini(c, x))
  | ihutsa_da(c) = Ihutsa (2)
  | otherwise = Ipini(kendu(c), x) (3)

  Errore-mezu bat aurkezteko Haskell-eko *error* funtzioa erabiltzen da eta zein mezu aurkeztu nahi den zehaztu beharko da.

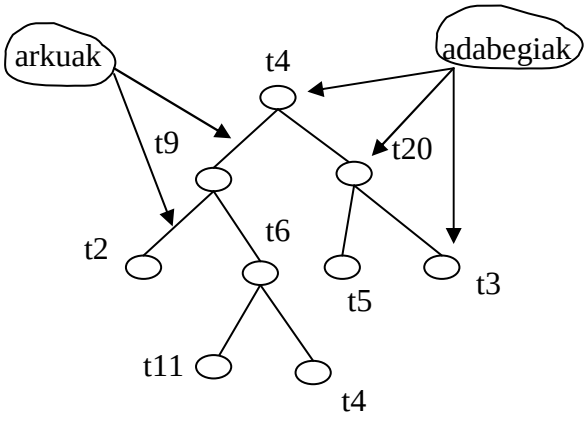
5.6. Zuhaitz bitarrak

t motako zuhaitz bitar bat t motako elementuz osatutako egitura ez-lineal bat da:

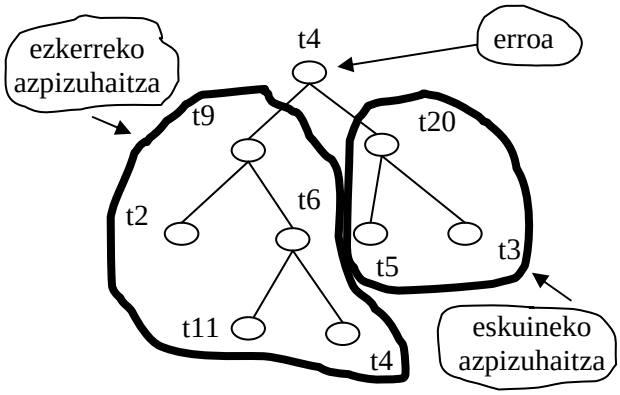


Irudi horretan t2, t3, t4 eta besteak t motako balioak direla suposatuko dugu.

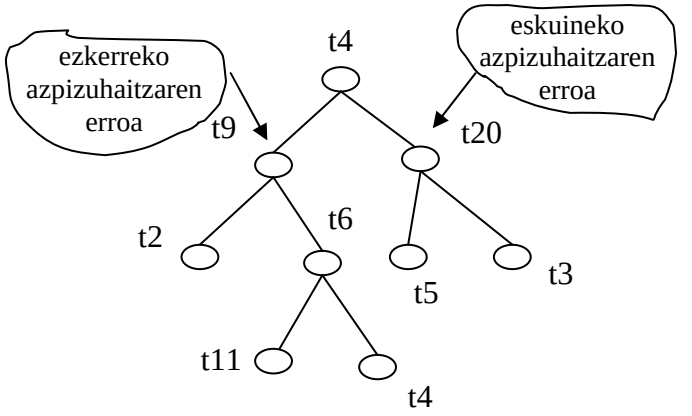
Zuhaitz bitar bat adabegiz eta zuzendutako arkuez (norantza bakarra duten arkuez) osatuta egoten da. Adabegi bakoitzari t motako balio bat dagokio.

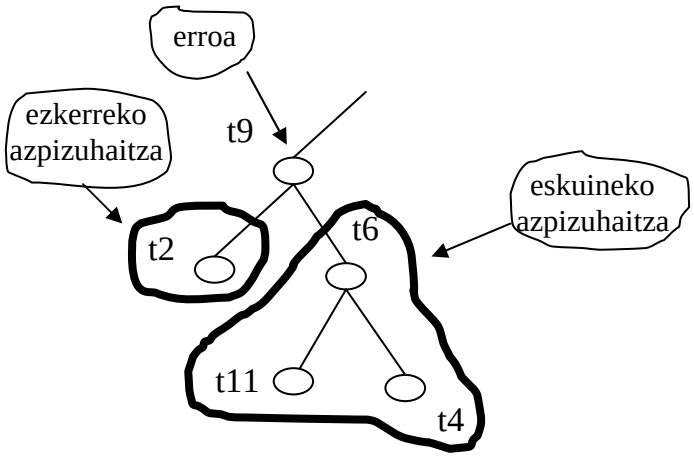


Zuhaitz bitar batean erroa, ezkerreko azpizuhaitza eta eskuineko azpizuhaitza bereiz daitezke:



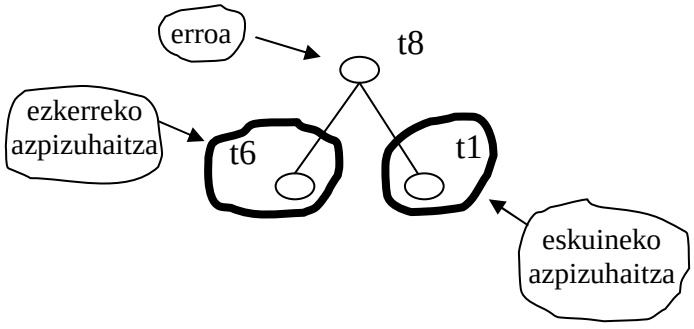
Era berean ezkerreko azpizuhaitzak bere erroa, bere ezkerreko azpizuhaitza eta bere eskuineko azpizuhaitza izango ditu. Eta gauza bera gertatzen da eskuineko azpizuhaitzarekin ere:



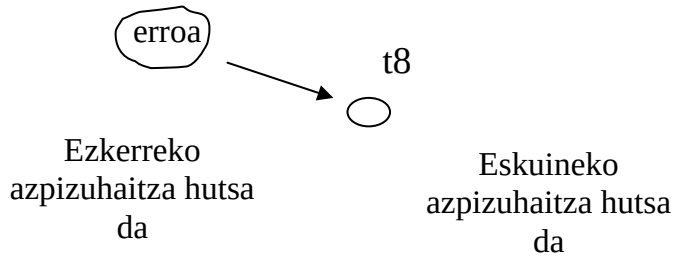


Era horretan zuhaitz batean erroa, ezkerreko azpizuhaitza eta eskuineko azpizuhaitza bereiz daitezke. Eta azpizuhaitzek aldi berean bereizketa bera onartzen dute eta horrela joan gaitezke azpizuhaitz bezala zuhaitz hutsak dituen zuhaitz batera iritsi arte.

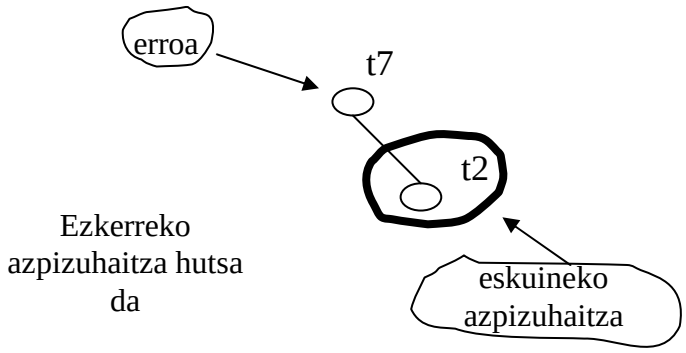
Jarraian ikus daitekeen zuhaitzean bai ezkerreko azpizuitza eta bai eskuineko azpizuhaitza adabegi bakar batez osatuta daude:



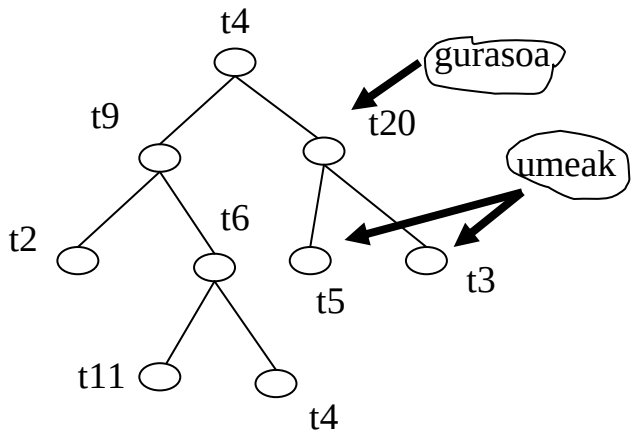
Jarraian datorren zuhaitz honek adabegi bakarra du (erroa) eta ezkerreko eta eskuineko azpizuitzak hutsak dira:



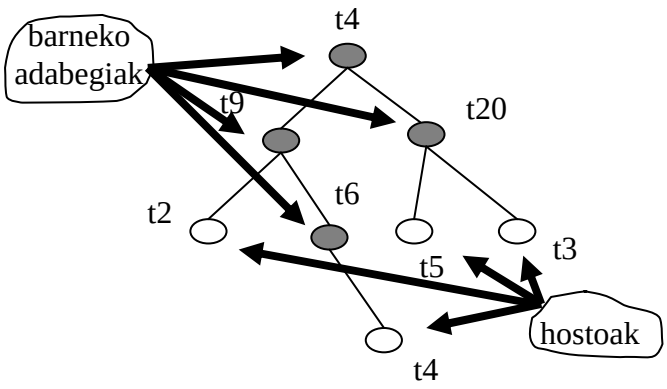
Jarraian ikus daitekeen zuhaitzak bi adabegi ditu. Ezkerreko azpizuhaitza hutsa da:



Adabegi bat bere ezkerreko eta eskuineko azpizuhaitzetako erroen gurasoa dela esaten da:



Azpizuhaitz biak hutsak dituen adabegiak hostoak direla esango dugu.
Gutxienez azpizuhaitzetako bat hutsa ez duten adabegiei barneko adabegi deituko diegu.



5.6.1. Eragiketa eraikitzaileak zuhaitz bitarrentzat

Datu-mota bati dagozkion eragiketa eraikitzaileak mota horretako balioak zein diren adierazteko dira.

Zuhaitz bitarren datu-mota ez dago aurredefinituta Haskell-en.

Datu-mota berri bat definitzeko bere eragiketa eraikitzaileak zein diren zehaztu behar da.

Zuhaitz bitarren datu-mota honela definituko dugu:

```
data Zuhbit t = Zhutsa | Eraiki(t, Zhutsa t, Zhutsa t)
```

Zuhaitz bitarrentzat eragiketa eraikitzaileak Zhutsa eta Eraiki dira.

Definizio horren bidez t motako zuhaitz bitar bat zuhaitz bitar hutsa izango dela (hori Zhutsa-ren bidez adieraziko da) edo bestela t motako elementu bat eta t motako bi zuhaitz bitar elkartuz eraikitako zuhaitz bat izango dela adierazten da. Elementu hori zuhaitz berriaren erroa izango da, lehenengo zuhaitza zuhaitz berriaren ezkerreko azpizuhaitza izango da eta bigarren zuhaitza zuhaitz berriaren eskuineko azpizuhaitza izango da. Gogoratu t parametroak edozein mota ordezkatzen duela (Int, Bool, eta abar).

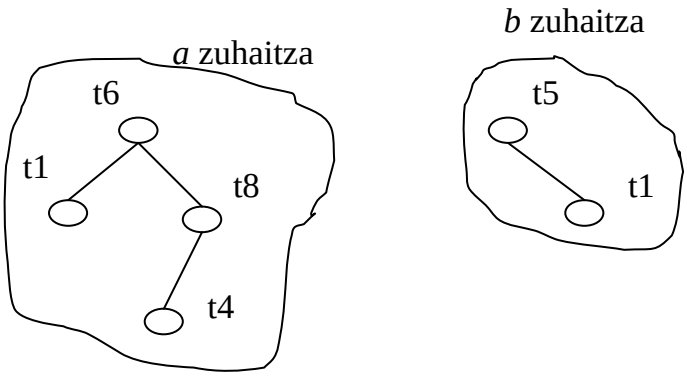
Beste era batera esanda, eragiketa eraikitzaileak ezerezetik zuhaitz hutsa sortzen duen eragiketa eta t motako balio bat eta bi zuhaitz emanda, zuhaitz berria eraikitzen duen eragiketak dira.

Zhutsa eta Eraiki eragiketa eraikitzaileen motak honako hauek dira:

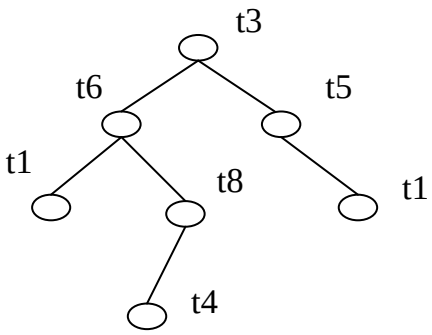
```
Zhutsa :: -> Zuhbit t
Eraiki:: (t, Zuhbit t, Zuhbit t) -> Zuhbit t
```

.

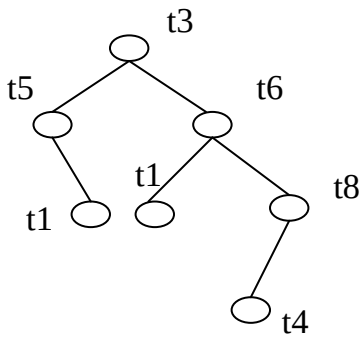
Demagun honako *a* eta *b* zuhaitz bitarrak ditugula:



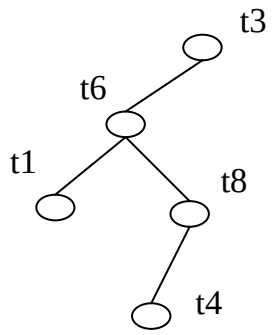
Eraiki(t3, *a*, *b*) zuhaitza honako zuhaitz hau da:



Eraiki(t3, *b*, *a*) zuhaitza honako zuhaitz hau da:



Eraiki(t3, *a*, Zhutsa) zuhaitza honako zuhaitz hau da:



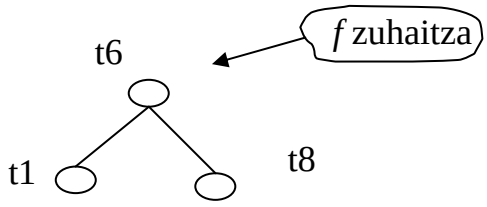
Eraiki(t3, Zhutsa, Zhutsa) zuhaitza honako zuhaitz hau da:

Edozein zuhaitz bitar eragiketa eraikitzaileen bidez adieraz daiteke.

Zuhaitz hutsa honela adieraziko genuke:

Zhutsa

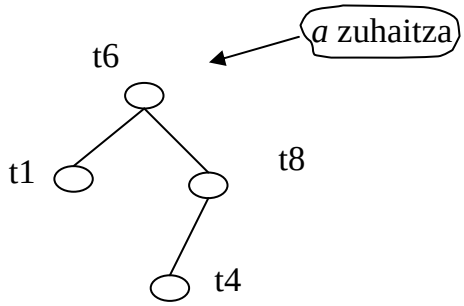
Har dezagun orain honako zuhaitz bitar hau:



f zuhaitz bitarra honela adieraziko genuke:

f = Eraiki(t6, Eraiki(t1, Zhutsa, Zhutsa), Eraiki(t8, Zhutsa, Zhutsa))

Orain har dezagun beste zuhaitz bitar hau



Eragiketa eraikitzaileen bidez honela adieraziko genuke:

a = Eraiki(t6, Eraiki(t1, Zhutsa, Zhutsa),
Eraiki(t8, Eraiki(t4, Zhutsa, Zhutsa),
Zhutsa
)
)

5.6.2. Eragiketa ez-eraikitzaile batzuk zuhaitz bitarrentzat

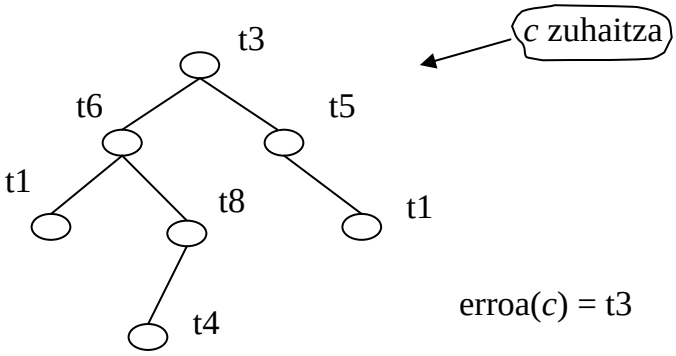
Zuhaitz bitarrekin kalkuluak burutzeko balio duten eragiketa ez eraikitzaile batzuk definituko dira jarraian. Eragiketa edo funtzio bat definitzeko funtzioaren mota eta funtzioa definitzen duten ekuazioak eman behar dira.

- **erroa**: zuhaitz bitar bat emanda, bere erroari dagokion balioa itzuliko du. Zuhaitza hutsa baldin bada errore-mezua aurkeztuko da.

  Adibideak:

  erroa(Zhutsa) = error

  Har dezagun honako zuhaitz hau:



erroa(*c*) = t3

Eragiketaren mota:
erroa:: (Zuhbit t) -> t

Eragiketa definitzen duten ekuazioak:
erroa(Zhutsa) = error "Zuhaitz hutsa. Ez dago errorik." (1)
erroa(Eraiki(x, a, b)) = x (2)

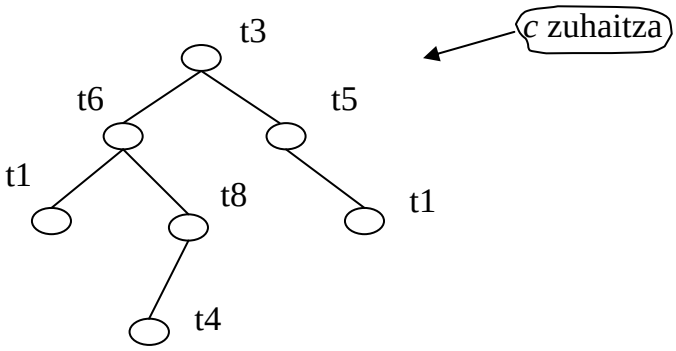
Errore-mezu bat aurkezteko Haskell-eko *error* funtzioa erabiltzen da eta zein mezu aurkeztu nahi den zehaztu beharko da.

- **ezker**: zuhaitz bitar bat emanda, funtzio honek ezkerreko azpizuhaitza itzuliko du. Datu bezala emandako zuhaitza hutsa baldin bada, errore mezua aurkeztu beharko du.

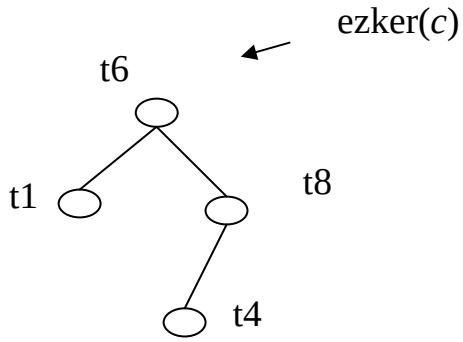
Adibideak:

ezker(Zhutsa) = error

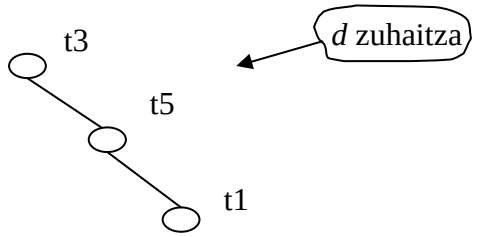
c zuhaitza honako hau baldin bada:



*ezke*r izeneko eragiketak ezkerreko azpizuhaitza itzuliko du:



d honako zuhaitz bitar hau baldin bada:



ezker(*d*) = Zhutsa

ezker izeneko eragiketak ezkerreko azpizuhaitza itzuliko du. Kasu honetan ezkerreko azpizuhaitza zuhaitz hutsa da:

Eragiketaren mota:
ezker:: (Zuhbit t) -> Zuhbit t

Eragiketa definitzen duten ekuazioak:
ezker(Zhutsa) = error "Zhuhaitz hutsa. Ez dago ezkerreko azpizuhaitzik." (1)
ezker(Eraiki(x, a, b)) = a (2)

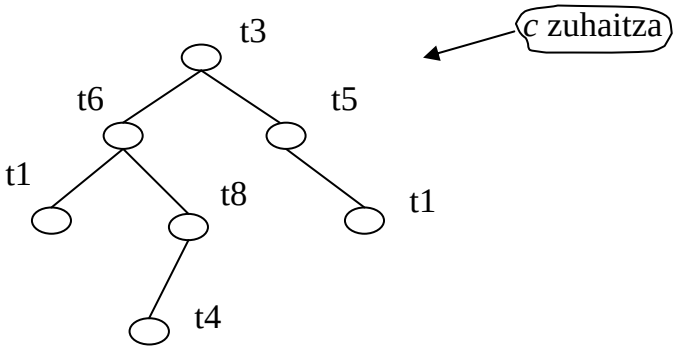
Errore-mezu bat aurkezteko Haskell-eko *error* funtzioa erabiltzen da eta zein mezu aurkeztu nahi den zehaztu beharko da.

- **<u>eskuin</u>**: zuhaitz bitar bat emanda, funtzio honek eskuineko azpizuhaitza itzuliko du. Datu bezala emandako zuhaitza hutsa baldin bada, errore mezua aurkeztu beharko du.

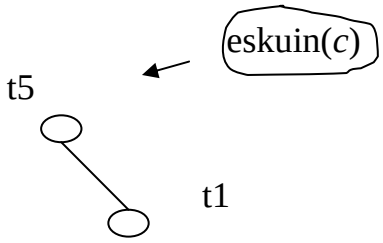
<u>Adibideak:</u>

eskuin(Zhutsa) = error

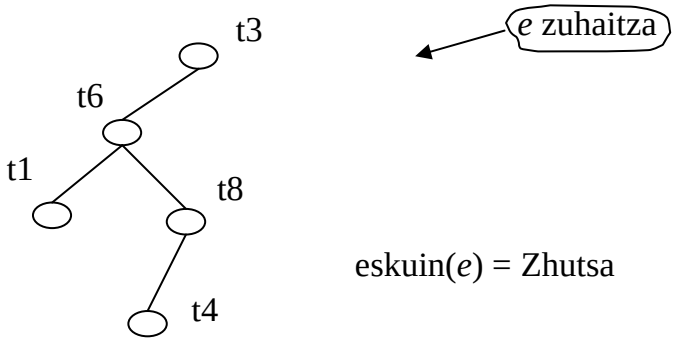
c zuhaitza honako hau baldin bada:



eskuin izeneko eragiketak eskuineko azpizuhaitza itzuliko du:



e zuhaitza honako hau baldin bada:



eskuin(*e*) = Zhutsa

Eragiketaren mota:
eskuin:: (Zuhbit t) -> Zuhbit t

Eragiketa definitzen duten ekuazioak:
eskuin(Zhutsa) = error "Zuhaitz hutsa. Ez dago eskuineko azpizuhaitzik." (1)
eskuin(Eraiki(x, a, b)) = b (2)

Errore-mezu bat aurkezteko Haskell-eko *error* funtzioa erabiltzen da eta zein mezu aurkeztu nahi den zehaztu beharko da.

- **zhutsa_da**: Zuhaitz bitar bat emanda, True itzuliko du zuhaitza hutsa baldin bada eta False itzuliko du hutsa ez bada.

Adibideak:
zhutsa_da(Zhutsa) = True
zhutsa_da(Eraiki(2, Eraiki(7, Eraiki(6, Zhutsa, Zhutsa), Zhutsa), Zhutsa)) = False

Eragiketaren mota:
zhutsa_da:: (Zuhbit t) -> Bool

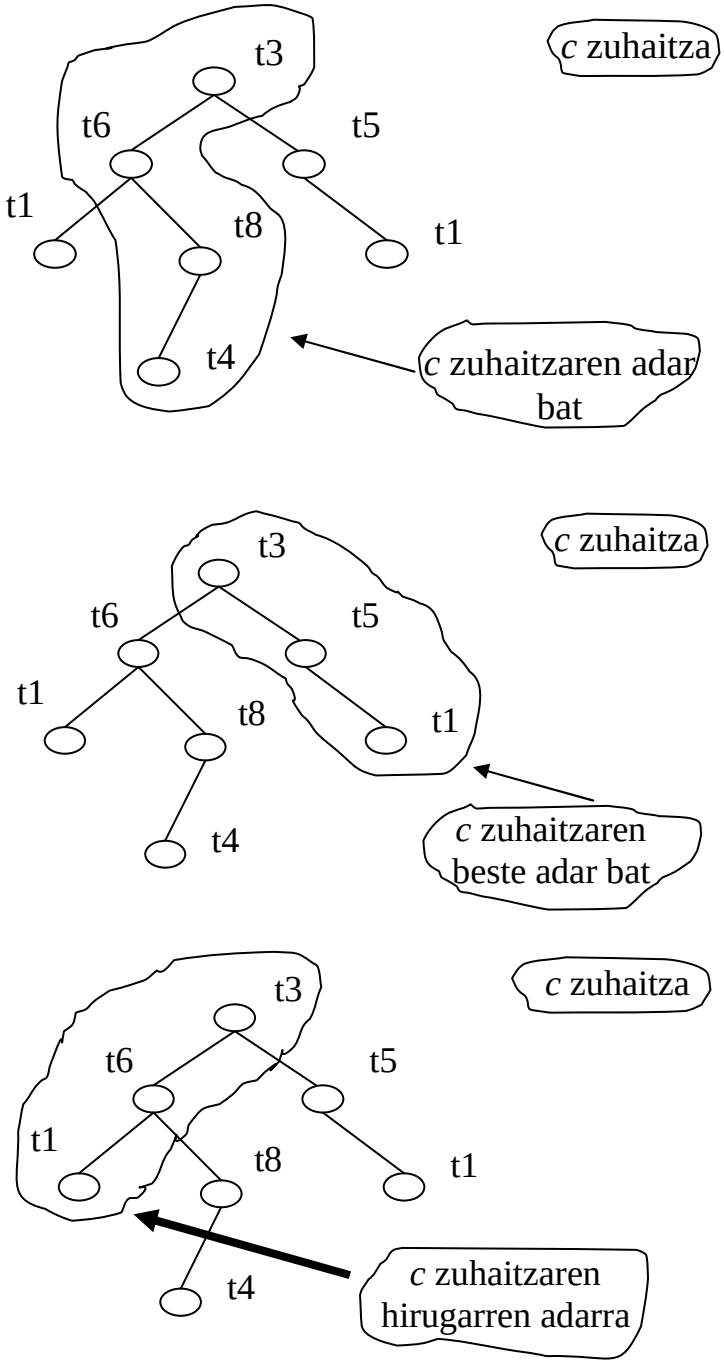
Eragiketa definitzen duten ekuazioak:
zhutsa_da(Zhutsa) = True (1)
zhutsa_da(Eraiki(x, a, b)) = False (2)

- **sakon**: zuhaitz bitar bat emanda, zuhaitzaren sakonera itzuliko du funtzio honek. Sakonera adar luzeenaren luzera da. Zuhaitza hutsa baldin bada, 0 itzuli beharko du. Adar bat errotik hosto bateraino dauden adabegiek osatzen dute.

  Adibideak:

  sakon(Zhutsa) = 0

  c honako zuhaitz hau baldin bada:



c zuhaitzak hiru adar ditu eta adar luzeenak lau adabegi ditu, beraz:

sakon(c) = 4

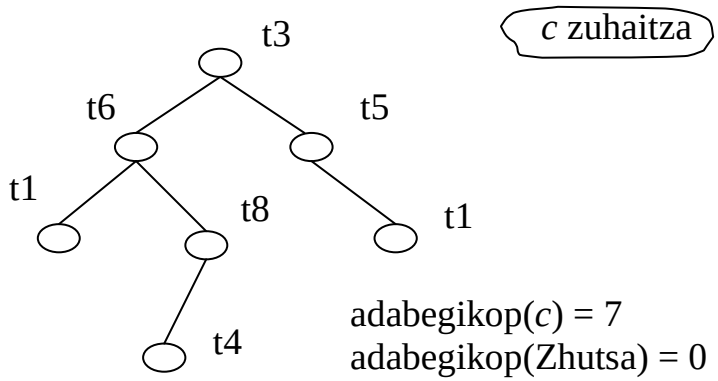
Eragiketaren mota:
sakon:: (Zuhbit t) -> Int

Eragiketa definitzen duten ekuazioak:

sakon(Zhutsa) = 0		(1)
sakon(Eraiki(x, a, b))		
| sakon(a) >= sakon(b)	= 1 + sakon(a)	(2)
| otherwise	= 1 + sakon(b)	(3)

- **adabegikop**: Zuhaitz bitar bat emanda, adabegi kopurua itzuliko du.

Adibideak:



Eragiketaren mota:
adabegikop:: (Zuhbit t) -> Int

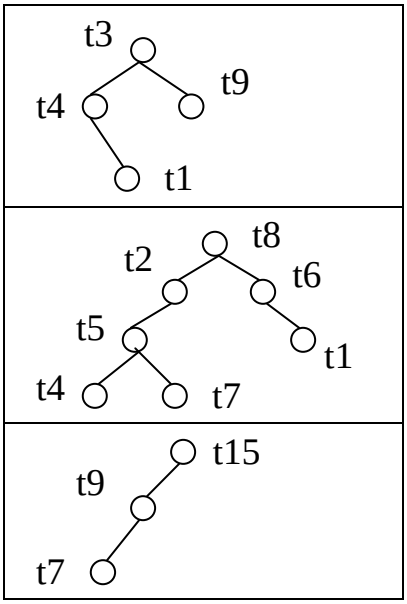
Eragiketa definitzen duten ekuazioak:

adabegikop(Zhutsa) = 0	(1)
adabegikop(Eraiki(x, a, b)) = 1 + adabegikop(a) + adabegikop(b)	(2)

5.7. Datu-mota abstraktuak nahasian dituzten adibideak

Zuhaitz bitarrez osatutako pila bat emanda, *sakonmax* eragiketak sakonera handiena duen zuhaitzaren sakonera itzuliko du:

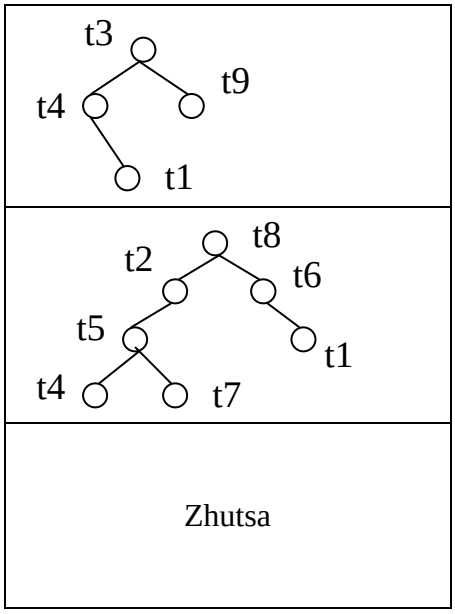
p zuhaitz bitarrez osatutako honako pila hau baldin bada:



Sakonera handiena erdian dagoen zuhaitzari dagokio eta 4 da:

$$\text{sakonmax}(p) = 4$$

Horrelako pila batean zuhaitz hutsak ere ager daitezke. Demagun q honako pila hau dela:



sakonmax(q) = 4

Eragiketaren mota:
sakonmax:: (Zuhbit (Pila t)) -> Int

Eragiketa definitzen duten ekuazioak:

```
sakonmax(Phutsa) = 0                                        (1)
sakonmax(Pilaratu(a, p))
        | sakon(a) >= sakonmax(p)       = sakon(a)          (2)
        | otherwise                     = sakonmax(p)       (3)
```

Hor a zuhaitz bitar bat da, p zuhaitz bitarrez osatutako pila bat da eta *sakon* zuhaitz bitar baten sakonera kalkulatzen duen funtzioa da.

5.8. Modularitatea Haskell-en

5.8.1. Bi moduluren definizioa aurreko ataletan emandako eragiketak erabiliz

Haskell-en modulu desberdinak definitzea eta modulu batetik beste modulu batean definitutako funtzioak erabiltzea oso erraza da.

Esate baterako, gai honetan definitu ditugun funtzioak kontuan hartuz bi modulu defini ditzakegu:

a) Modulu bat oinarrizko motentzat definitutako funtzioekin (bikoitia, bakoitia, hand3)
b) Beste modulu bat zerrendentzat definitutako funtzioekin (leh, hond, hutsa_da, badago, luzera, bikop, tartekatu, tartekatu2)

Modulu bakoitza fitxategi desberdin batean joango da. Orain fitxategi bakoitza nola gelditzen den erakutsiko da.

a) "Oinarrizkoak.hs" fitxategia

```
module Oinarrizkoak where

bikoitia:: (Int) -> Bool
bikoitia (x) = (x `mod` 2) == 0

bakoitia:: (Int) -> Bool
bakoitia (x) = not (bikoitia(x))

hand3:: (Int, Int, Int) -> Int
hand3 (x, y, z)
                | x >= y && x >= z          = x
                | y > x && y >= z           = y
                | otherwise                 = z
```

b) "Zerrendak.hs" fitxategia

```
module Zerrendak where
import Oinarrizkoak

        leh:: ([t]) -> t
        leh([]) = error "Zerrenda hutsa. Ez dago lehenengo elementurik."
        leh(x:s) = x

        hond:: ([t]) -> [t]
        hond([]) = error "Zerrenda hutsa. Ez dago hondarrik."
        hond(x:s) = s

        hutsa_da:: ([t]) -> Bool
        hutsa_da([]) = True
        hutsa_da(x:s) = False

        badago:: (t, [t]) -> Bool
        badago(x, []) = False
        badago(x, y:s)
                | x == y        = True
                | x /= y        = badago(x, s)

        luzera:: ([t]) -> Int
        luzera([]) = 0
        luzera(x:r) = 1 + luzera(r)

        bikop:: ([Int]) -> Int
        bikop([]) = 0
        bikop(x:r)
                | bikoitia(x)           = 1 + bikop(r)
                | bakoitia(x)   = bikop(r)

        tartekatu:: ([t], [t]) -> [t]
        tartekatu([], s)
                | luzera(s) /= 0        = error "Luzera desberdineko zerrendak."
                | otherwise             = []
        tartekatu(x:r, s)
                | luzera(x:r) /= luzera(s)      = error "Luzera desberdineko zerrendak."
                | otherwise                     = x:(leh(s): tartekatu(r, hond(s)))

        tartekatu2:: ([t], [t]) -> [t]
        tartekatu2([], s)
                | luzera(s) /= 0        = error "Luzera desberdineko zerrendak."
                | otherwise             = []
        tartekatu2(x:r, s)
                | luzera(x:r) /= luzera(s)      = error "Luzera desberdineko zerrendak."
                | otherwise                     = leh(s):(x: tartekatu2(r, hond(s)))
```

[] eta : eragile eraikitzaileak eta ++ eragilea aurredefinituta daude Haskell-en.

"Zerrendak" izeneko modulua exekutatuz gai honetan definitutako funtzio denak erabil daitezke, modulu horretatik beste modulu denak inportatzen baitira.

5.8.2. Modulu nagusiaren definizioa

Modulu bakoitza era independientean erabil daiteke baina modulu denetan definitutako funtzio denak erabiltzeko aukera eduki nahi denean modulu nagusi bat definitzea izaten da onena. Kasu honetan "Nagusia.hs" modulua definitu da helburu horrekin eta bertan beste modulu denak inportatzen dira. Hala ere "Zerrendak" izeneko modulua exekutatuz gai honetan definitutako funtzio denak erabil daitezke, modulu horretatik Oinarrizkoak modulua inportatzen baita.

```
module Nagusia where
import Oinarrizkoak
import Zerrendak
```

5.9. Oharrak

5.9.1. Maiuskula eta minuskulen erabilera izenetan

Maiuskulak eta minuskulak ondo erabiltzeko honako arau hauek jarraitu behar dira:

- Datu moten izenak maiuskulaz hasten dira: Int, Bool, Char, …
- Moten ordez ipintzen diren aldagaien izenak minuskulaz hasten dira: t.
- True eta False konstanteak maiuskulaz hasten dira.
- Eraikitzaileak ez diren beste eragile denak eta aldagaien izenak minuskulaz hasten dira (maiuskulak erdian ipintzea egon arren): bikoitia, bakoitia, leh, hond, badago, luzera, x, s, r, p, q, zatb, kenduLuz, kenduLuzBik, …

5.9.2. Azalpenak ipintzeko bi era

Haskell-ez idatzitako programetan azalpenak ipini nahi izanez gero, bi aukera daude:

- -- erabiliz:

  -- sinboloen atzetik lerroa bukatu arte datorren dena azalpena izango da.

  Adibidea:

  --Hau azalpen bat da eta lerroa bukatzen denean bukatuko da azalpena

- {- eta -} erabiliz:

  {- eta -} sinboloen artean doan dena azalpena da. Lerro batetik bestera pasatuta ere azalpenak aurrera jarraitzen du.

  Adibidea:

  {- Hau azalpena da baina
  lerroz alda gaitezke. -}