Rökfræði í hugbúnaðargerð, vor 2011

Tvíundartré, heiltölur og kvik líkön í Alloy

Inngangur

Á blöðum um Alloy frá 1. mars er rætt um notkun Alloy til að lýsa kyrrum (*static*) líkönum en hér verður sagt frá hvernig búa má til kvik (*dynamic*) líkön. Jafnframt verður rætt aðeins meira um heiltölur, og tekin fleiri dæmi um Alloy líkön. Sér í lagi verður dæmi um tvíundartré notað til að skýra málin. Þar sem þessi blöð eru einskonar framhald af blöðunum frá 1. mars, sem enduðu með dæmi 4, það verður byrjað með dæmi 5 hér.

Dæmi 5. Tvíundartré.

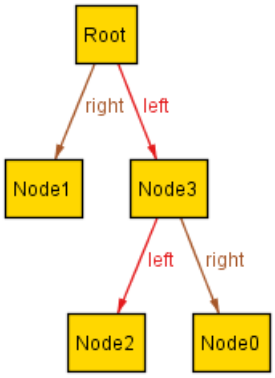
Tvíundartré er gagnagrind sem samanstendur af hnútum (*nodes*) og venslum milli þeirra, þannig að hver hnútur getur átt vinstra barn og/eða hægra barn. Einn af hnútunum er foreldralaus rót (*root*) og enginn hnútur er eiginn afkomandi. Einföld lýsing á tvíundartré í Alloy er:

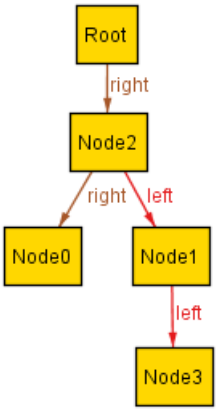
```
module bintree5
sig Node {left, right: lone Node}
one sig Root extends Node {}
fun parent[n:Node]: Node {n.~(left+right)}

fact all_below_root       {all n: Node | n in Root.*(left+right)}
fact not_cyclic           {no n: Node | n in n.^(left+right)}
fact left_right_different {no n: Node | some n.left & n.right}
fact all_one_parent       {all n: Node | lone parent[n]}

run {} for exactly 5 Node
```

Þegar þessi lýsing er keyrð fást m.a. eftirfarandi myndir:





Dæmið sýnir m.a. notkun á falli (`fun parent`) og spegilvenslum (~) sem ekki hafa áður sést. Tökum sérstaklega eftir hvernig fallið `parent` er skilgreint og hvernig kallað er á það.

Athugum líka hvernig segðin `n.~(left+right)` er túlkuð. Skilgreiningin `sig Node{...}` skilgreinir tvenn vensl með víðvært gildissvið, `left` og `right`.[1] Segðin `left+right` er sammengi þeirra, þ.e. mengi allra para (*foreldri*`,` *barn*), og þegar ~ er bætt við fást öll pör (*barn*`,` *foreldri*). Með því að rita `n.` framan við það fæst að lokum frámengi {n}, sem sé mengi allra foreldra sem hafa barnið `n`.

Takmörkun á hæð trés með heiltölutagi: Dæmi 6

Til að geta reiknað og vísað í hæð tvíundartrés þarf að skilgreina breytu af heiltölutagi, og ein leið er að hafa eina slíka breytu, `level`, fyrir hvern hnút. Síðan má setja tvær staðreyndir í lýsinguna, nefnilega að rótin sé á fyrsta þrepi, og að sérhver hnútur sé einu þrepi neðar en foreldrið. Í framhaldi mætti svo krefjast þess að allir hnútar hafi þrep ≤ 3 (til dæmis). Til að koma þessu í kring má breyta skilgreiningu á `Node` í:

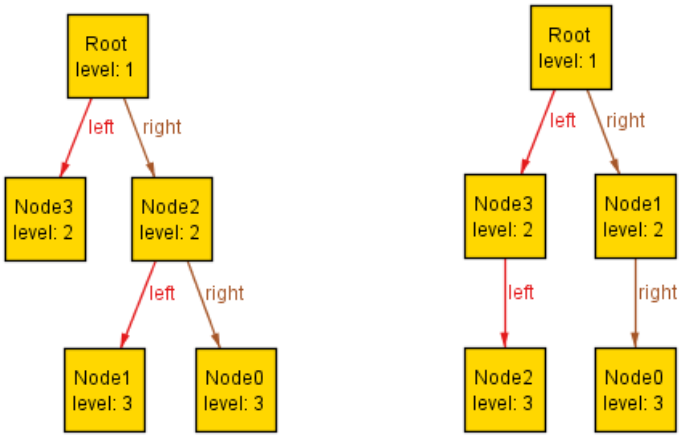
[1] Bæði `left` og `right` eru hlutmengi í `Node` × `Node`. Þrátt fyrir að *sig*-skilgreiningin líkist skilgreiningu klasa eða færslu, þá þarf sem sé ekki að rita `n.left` eða `n.right` fyrir eitthvert tiltekið stak `n` í `Node` (og ef það er gert fást auk þess ekki vensl, heldur hlutmengi í `Node`, nánar tiltekið mengi staka sem eru vensluð við `n`).

```
sig Node {left, right: lone Node, level: Int}
```

Síðan er þremur staðreyndum bætt við Alloy-lýsinguna:

```
fact {Root.level = 1}
fact {all n: Node | n.level = parent[n].level + 1}
fact {all n: Node | n.level <= 3}
```

Þegar þessi lýsing er keyrð teiknast sérstakur kassi með tölu í fyrir hvert þrep, og örvar frá hnútunum yfir í þessa kassa. Betri mynd má fá með því að smella á *Theme* hnapp, velja *relations–level*, taka burtu hak við *Show-as-arcs*, og haka í staðinn við *Show-as-attribute*. Vegna hæðartakmörkunarinnar teiknast nú aðeins tvö tré með 5 hnútum, nefnilega:



Mörg tvíundartré í sama líkani: Dæmi 7

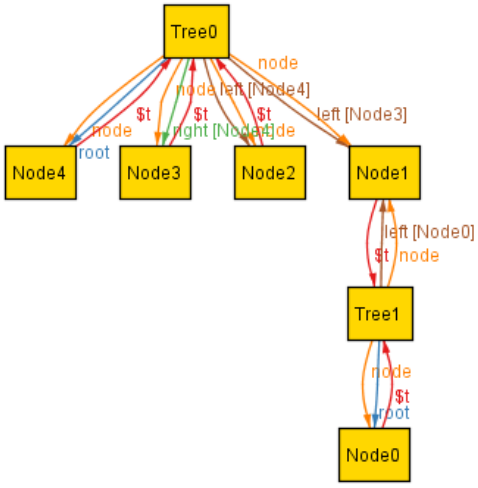
Í kennslubókinni, á bls. 148–156 er kafli þar sem m.a. er sýnt hvernig hægt er að nota Alloy til að líkja eftir því að bætt sé við pakka í hugbúnaðarkerfi og einnig að pakka sé eytt. Í svona lýsingum þarf að vera hægt að hafa líkan af kerfi fyrir breytingu, og líkan eftir breytingu. Þetta krefst þess að sérstök *sig*-skilgreining sé fyrir kerfið sem lýsa skal, og þar verða að vera svið fyrir alla eiginleika kerfisins, svo hægt sé að smíða samtímis tvö eða fleiri kerfi. Byrjum á að breyta lýsingunni á tvíundartrénu í þessa veru.

```
module bintree7
sig Node {}
sig Tree {
   node: set Node,
   root: node,
   left: node -> lone node,
   right: node -> lone node,
   parent: node -> lone node
}{                                    -- Constraints:
   parent = ~(left + right)           -- mirror relation
   all n: node |                      -- all nodes...
      n in root.*(left+right) &&      --   are below the root
      no n & n.^(left + right) &&     --   are not descendants of themselves
      no n.left & n.right             --   have distinct left and right children
}
fact {all n: Node|some t:Tree|n in t.node}  -- all nodes belong to some tree
run {} for exactly 5 Node, exactly 2 Tree
```

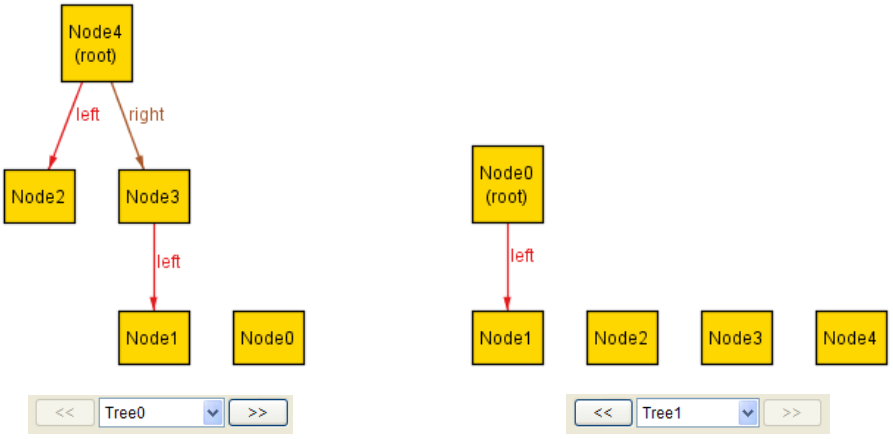
Í staðinn fyrir að hafa sérstakar *fact*-klausur hefur verið valin sú leið að hafa skorðu-klausu aftast í *sig*-skilgreiningunni, eins og nefnt var að væri hægt á bls. 4 á fyrri Alloy-blöðunum. Í slíkri skorðuklausu er undanskilið að breyturnar eru svið í *Tree*. Hægt væri að setja þessar skorður fram í svolítið lengra máli í *fact*-klausum. Til dæmis mætti rita:

```
fact {all t:Tree, n:t.node | n in (t.root).*(t.left+t.right)} -- all below root
fact {all t:Tree, n:t.node | no n & n.^(t.left+t.right)}      -- not cyclic
```

Þegar lýsingin `bintree7` er keyrð fæst eftirfarandi mynd:



Til að laga þessa mynd má velja fyrist *Projected over Tree*, smella svo á *Theme*-hnapp og velja *types-and-sets*, *set-$t* og taka burtu hak við *Show-as-labels* og velja að lokum *types-and-sets*, *set-node* og taka samsvarandi hak burtu þar líka. Þá fást eftirfarandi myndir:



Kvik líkön: Dæmi 8

Til að búa til kvik líkön má gera það með aðstoð umsagnar sem hefur þrjá (og stundum fleiri) stika: kerfi fyrir breytingu, kerfi eftir breytingu, og hlutur sem bætist við eða hverfur við breytingu. Fyrstu tveir stikarnir eru gjarnan skírðir sama nafni, nema sá seinni er með merki, t.d. `x` og `x'` (breytur í Alloy mega innihalda `a`–`z`, `A`–`Z`, `0`–`9`, `_` og `'`). Dæmi um svona umsagnir eru `AddComponent` og `RemoveComponent` á bls. 151 í kennslubók. Dæmi um kvika umsögn með fleiri stika er `AGuidedSimulation` á bls. 156 í kennslubók, sem hefur 5 stika (þrjú kerfi og tvo hluti). Umsögnin lýsir því hvernig kerfin eiga að tengjast. Einföld umsögn af þessu tagi fyrir tvíundartrén er:

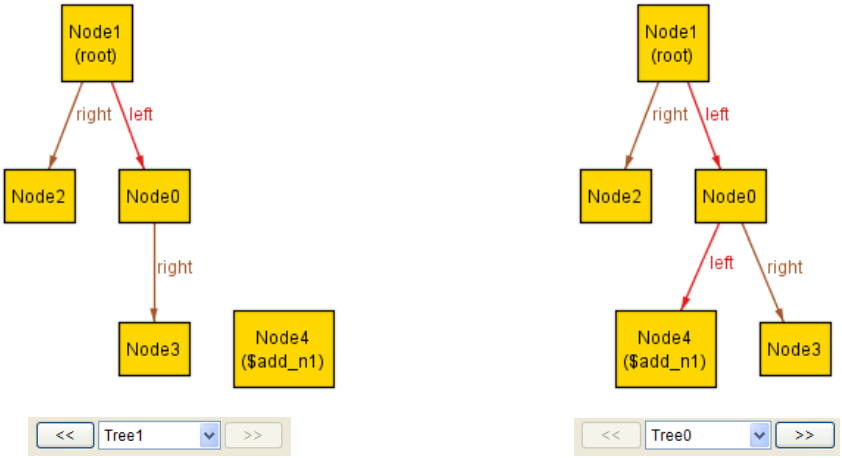
```
pred add[t, t': Tree, n1: Node] {
   t'.root = t.root
   t'.node = t.node + n1
   not n1 in t.node
   t.left in t'.left
   t.right in t'.right
}
```

Trén hafa sömu rót og sömu hnúta, nema hvað `n1` bætist við hnútamengið í `t'`. Síðan eru öll vensl sem eru milli hnúta í `t` líka fyrir hendi í `t'`. Aðrar skorður eiga að tryggja að ekki þurfi að tilgreina að það bætist nákvæmlega eitt par við annaðhvort `left` eða `right` í `t'` (nefnilega að allir hnútar eru tengdir nákvæmlega einum öðrum hnút).

Ef nýja lýsingin er keyrð með:

```
run add for exactly 5 Node, exactly 2 Tree
```

fást m.a. eftirfarandi myndir:



Hér hefur verið lagfært með því að taka burtu hak við *Show-as-labels* fyrir *set-$add_t*. Við sjáum að í hægra trénu hefur `Node4` bæst við miðað við vinstra tréð. Tökum líka eftir að trén koma í öfugri röð: vinstra tréð er númer 1 en það hægra númer 0. Þessi öfuga röð verður lagfærð í dæmi 9 að neðan.

Kvik líkön sem sýna runu breytinga: Dæmi 9

Hægt er að búa til runu af líkönum, þannig að aðliggjandi tré í rununni séu tengd með tiltekinni breytingar-umsögn með því að nota *ordering*-pakka sem fylgir með Alloy. Vísað er í *ordering*-pakkann með *open*-línu sem er á eftir *module*-línu, og þar er jafnframt tekið fram hverju á að raða. Í okkar tilviki skal raða hlutum af taginu `Tree` og *open*-línan verður

```
open util/ordering[Tree]
```

Umsögnin sem tengir saman kerfi fyrir og eftir breytingu verður `add`, óbreytt frá dæmi 8, en við bætist staðreynd:

```
fact show_change_trace {
   #first[].node = 1
   all t: Tree - last[] | let t' = next[t] |
      some n1: Node | add[t, t', n1]
}
```

Hér er notuð *let*-skipun sem ekki hefur verið minnst á áður á þessum blöðum, en hún kemur við sögu í ýmsum dæmum í kennslubókinni. Það sem hún gerir er að láta `t'` vera samheiti fyrir `next[t]` (þannig að það mætti sleppa *let*-klausunni og skrifa í staðinn „`add[t, next[t], n1]`“). Föllin `first`, `next` og `last` eru úr *ordering*-pakkanum. Fyrsta staðreyndin segir að ætlunin sé að byrja með tré sem hefur nákvæmlega einn hnút (sem hlýtur þá að verða rótin). Síðan er sagt að öll tré nema það síðasta eigi að tengjast næsta tré með umsögninni `add`.

Ef þessi lýsing er keyrð býr hún fyrst til tré sem breiða ekkert úr sér, heldur eru öll á hæðina, með einn hnút á hverju þrepi. Þetta má laga með því að setja takmörkun á hæð trjánna eins og gert var í dæmi 6 að framan. Við skilgreininguna `Sig Tree {...}` er bætt línunni:

```
level: node -> one Int
```

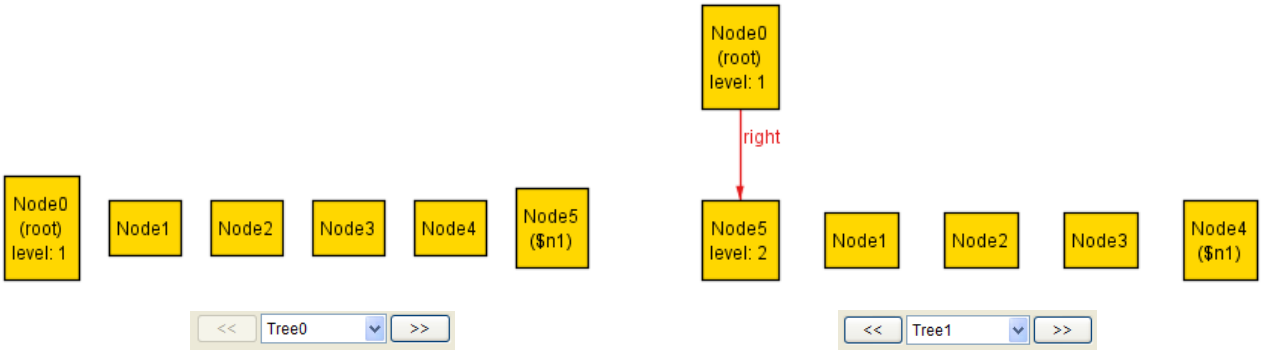
og við skorðurnar í þeirri skilgreiningu er bætt

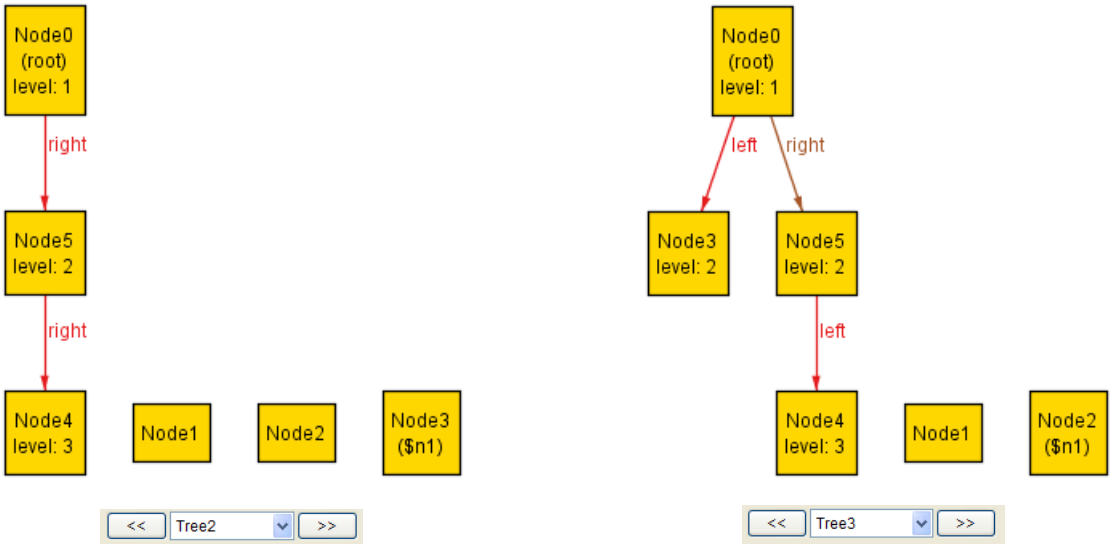
```
root.level = 1
all n: node | n.level = 1 + parent.level
all n: node | n.level <= 3
```

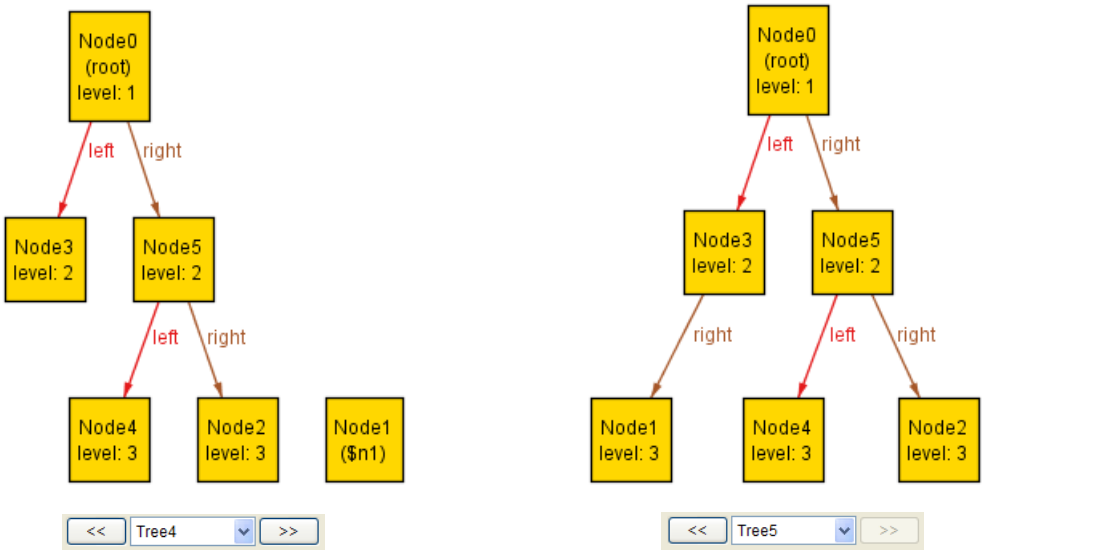
Ef þetta forritið er síðan keyrt með:

```
run {} for 6
```

fást myndirnar á næstu síðu (að vísu eftir nokkrar stillingar, sem fela í sér að velja *Projected over Tree*, að sleppt sé að sýna *$t*, *node*, og *parent*, og að *level* sé sýnt sem *attribute*).







Dæmin á þessum blöðum má finna á heimasíðu námskeiðsins, http://cs.hi.is/rokhug11.

Kristján Jónasson, 14. mars 2011.