Alloy
Specifikace pomocí relační logiky

Radek Mařík

ČVUT FEL, K13132

October 2, 2014

Obsah

I Am My Own Grandpa - píseň

- Výzvou je vytvořit situaci muže, který je svým vlastním dědečkem, aniž by byl spáchán incest nebo bylo potřeba cestování časem.
- Slova písně popisují řešení.

I Am My Own Grandpa

Many many years ago, when I was twenty-three,
I was married to a widow as pretty as can be,
This widow had a grown-up daughter who had hair of red,
My father fell in love with her and soon the two were wed.

I'm my own grandpa, I'm my own grandpa.
It sounds funny, I know, but it really is so
I'm my own grandpa.

This made my dad my son-in-law and changed my very life,
For my daughter was my mother, for she was my father's wife.
To complicate the matter, even though it brought me joy,
I soon became the father of a bouncing baby boy.

My little baby thus became a brother-in-law to dad,
And so became my uncle, though it made me very sad,
For if he was my uncle then that also made him brother
To the widow's grown-up daughter, who of course was my step-mother.

. . . by Dwight B. Latham and Moe Jaffe

Father's wife then had a son who kept them on the run.
And he became my grandchild for he was my daughter's son.
My wife is now my mother's mother and it makes me blue,
Because although she is my wife, she's my grandmother, too.

Oh, if my wife's my grandmother then I am her grandchild.
And every time I think of it, it nearly drives me wild.
For now I have become the strangest case you ever sawn
As the husband of my grandmother, I am my own grandpa.

I'm my own grandpa, I'm my own grandpa.
It sounds funny, I know, but it really is so I'm my own grandpa.
I'm my own grandpa, I'm my own grandpa.
It sounds funny, I know, but it really is so
I'm my own grandpa.

I Am My Own Grandpa - Alloy řešení

```
module grandpa
abstract sig Person {
  father: lone Man,
  mother: lone Woman }
sig Man extends Person { wife: lone Woman }
sig Woman extends Person { husband: lone Man }
fact Biology { no p: Person | p in p.^(mother+father) }
fact Terminology { wife = ~husband }
fact SocialConvention {
  no wife & *(mother+father).mother
  no husband & *(mother+father).father }
fun grandpas [p: Person]: set Person {
  let parent = mother + father + father.wife + mother.husband |
  p.parent.parent & Man }
pred ownGrandpa [m: Man] { m in grandpas[m]  }
run ownGrandpa for 4 Person expect 1
```

Alloy - pro co se používá?

- Alloy je modelovací jazyk pro návrh softwaru.
- Není však určen pro modelování architektury (jako např. UML).
- Je dostatečně obecný, aby mohl modelovat
  - jakoukoliv doménu individuálů,
  - relace mezi nimi.

Atomy

- Vše je postaveno na atomech a relacích.
- **Atom** je primitivní entita, která je
  - **nedělitelná:** nemůže být rozdělena na menší části,
  - **neměnná:** její vlastnosti se nemění v čase,
  - **neiterpretovaná:** nemá žádnou vestavěnou vlastnost,
- **Relace** je struktura, která zachycuje vztahy mezi atomy. Je to množina *n*-**tic**, každá *n*-tice je sekvence atomů

Signatura

- **Signatura** zavádí množinu atomů.
- Deklarace definující množinu pojmenovanou *A*

```
sig A { }
```

- Množina může být zavedena jako **rozšíření** jiné množiny, takto *A*1 je podmnožinou množiny *A*.

```
sig A1 extends A { }
```

- Signatury deklarované nezávisle na jakékoliv ostatní signatuře je tzv. **vrcholová** signatura.
- Rozšíření té samé signatury jsou vzájemně disjunktní, tak jako vrcholové signatury.
- Množina může být zavedena jako **podmnožina** jiné množiny

```
sig A1 in A { }
```

- **Abstraktní signatura** nemá žádné elementy s vyjímkou těch, které patří do jejích rozšíření nebo podmnožin.

```
abstract sig A { }
```

Signatura - příklad

```
abstract sig Person { }
sig Man extends Person { }
sig Woman extends Person { }
sig Married in Person { }
```

Pole

- Relace se deklarují jako **pole** v signatuře.
- Deklarace definující relaci f, jejíž doména je A a jejíž obraz je dán výrazem e

```
sig A { f: e }
```

- Příklady
  - Binární relace ... $f1$ je podmnožinou $A \times A$

```
sig A { f1: A }
```

  - Ternární relace ... $f2$ je podmnožinou $B \times A \times A$

```
sig B { f2: A -> A }
```

Násobnost

- Dovoluje nám omezit velikost množin.
  - Klíčové slovo násobnosti umístěné před deklarací signaturou omezuje počet elementu v množině signatury

```
m sig A { f: e }
```

  - Můžeme omezit násobnosti polí

```
sig A { f:  m e }
sig A { f: e1  m ->  n e2 }
```

- Existují čtyři násobnosti
  - `set`: jakýkoliv počet,
  - `some`: jeden nebo více,
  - `lone`: nula nebo jeden (L),
  - `one`: přesně jeden,
- Implicitní klíčové slovo, pokud je vynecháno, je `one`. Takže následující jsou ekvivalentní zápisy:

```
sig A { f:  e }
sig A { f: one e }
```

Kvantifikátory

- Alloy poskytuje bohatou kolekci kvantifikátorů
  `all x: S | F` : F platí pro každé x v S,
  `some x: S | F` : F platí pro nějaké x v S,
  `no x: S | F` : F selže pro každé x v S,
  `lone x: S | F` : F platí pro nejvíce jedno x v S,
  `one x: S | F` : F platí pro právě jedno x v S,

Logické operátory

- Jsou k dispozici běžně používané logické operátory

`not`	`!`	negace
`and`	`&&`	konjukce
`or`	`||`	disjunkce
`implies`	`=>`	implikace
`else`		alternativa
	`<=>`	iff (ekvivalence)

- Příklad
  `a != b` je ekvivalentní `not a = b`

Množiny a operátory

- Předdefinované množinové konstanty
  - `none` : prázdná množina,
  - `univ` : univerzální množina,
  - `ident` : identita,
- Množinové operátory
  - `+` : sjednocení
  - `&` : průnik
  - `-` : rozdíl
  - `in` : podmnožina
  - `=` : rovnost
  - Příklad: ženatí muži

    ```
    Married & Man
    ```
- Vymezená množina (angl. set comprehension)
  - Množina hodnot množiny S, pro které platí F

    ```
    { x : S | F }
    ```

Relační operátory

- `->` šipka (součin)
- `~` transpozice
- `.` dot join
- `[]` box join
- `^` tranzitivní uzávěr
- `*` reflexině-tranzitivní uzávěr
- `<:` omezení domény
- `:>` omezení obrazu
- `++` přepsání

Součin, Transpozice

- Šipkový součin `p -> q`
  - p a q jsou dvě relace
  - `p -> q` je relace, která vezme všechny kombinace n-tic relace p a m-tic relace q a spojí je (konkatenace).
  - Příklad

```
Name = { (N0), (N1) }
Addr = { (D0), (D1) }
Book = { (B0) }
Book -> Name -> Addr = { (B0,N0,D0), (B0,N0,D1),
                         (B0,N1,D0), (B0,N1,D1) }
```

- Transpozice `~p`
  - produkuje zrcadlový obraz relace p
  - tzn. reverzuje pořadí atomů v každé n-tici.
  - Příklad

```
example = { (a0,a1,a2,a3), (b0,b1,b2,b3) }
~example = { (a3,a2,a1,a0), (b3,b2,b1,b0) }
```

Spojení n-tic

- `p.q` Co je spojením těchto dvou n-tic?
  - $p = (s_1, \ldots, s_n)$
  - $q = (t_1, \ldots, t_m)$
  - Jestliže $s_n ! = t_1$, potom výsledek je prázdný
  - Jestliže $s_n = t_1$, potom výsledkem je k-tice $(s_1, \ldots, s_{m-1}, t_2, \ldots, t_m)$
- Příklad

```
{(a,b)}.{(a,c)} = {}
{(a,b)}.{(b,c)} = {(a,c)}
```

- Co se stane v případě `{(a)}.{(a)}`?
  - Není definováno!
  - $p.s$ je definováno, tehdy a jen tehdy, jestliže p a s nejsou obě unární relace

Spojení relací, Uzávěry, Omezení relací

- `p.q`
  - p a q jsou dvě relace a nejsou obě unární.
  - `p.q` je relace, která vznikne kombinací všech n-tic z p a m-tic z q a přidáním jejich spojení, pokud tento existuje.
- `p[q]` (box join)
  - sémanticky identické spojení dot join, ale bere své argumenty v opačném pořadí

$$p[q] \equiv q.p$$

- `^r = r + r.r + r.r.r + ...`
- `*r = ^r + iden`
- `s<:r` obsahuje n-tice z r **začínající** elementem v s
- `r:>s` obsahuje n-tice z r **končící** elementem v s
- `p++q = p - (domain(q) <: p) + q`

Let

- Lze zjednodušit výrazy

```
let x =  e | A
```

  - Každý výskyt proměnné x je nahrazen výrazem e v A
- Příklady
  - "Člověk v manželství má právě jednoho partnera"

```
sig Married in Person { spouse: one Married }
```

  - "Každý ženatý muž (žena) má manželku (manžela)"

```
all p: Married |
  let q = p.spouse |
    (p in Man => q in Woman) and
    (p in Woman => q in Man)
```

Skaláry

- Vše je množina v Alloy
  - neexistují skaláry
  - používáme singleton relaci

```
let matt =  one Person
```

  - Interpretace kvantifikací:

```
all x : S | ... x ...
```

    $x = \{t\}$ pro nějaký element t z S

Fakta

- Další omezení na signatury a pole lze vyjádřit v Alloy jako **fakta**
- AA (Alloy Analyzer) hledá instance modelu, které také splňují všechna omezení určená fakty
- Příklad: "Žádná osoba nemůže být svým vlastním předchůdcem."

```
fact selfAncestor {
  no p: Person | p in p.^parents
}
```

Funkce a predikáty

- Mohou být použity jako "makra"
  - Mohou být pojmenované a násobně použité v různých kontextech (fakta, tvrzení, podmínky běhu)
  - Mohou být parametrizována, používají se ke zkrácení zápisů
- **Funkce**:
  - Pojmenovaný výraz s žádným či více argumenty.
  - Vrací výraz jako návratovou hodnotu
  - Během analýzy se vyvolávají pouze, pokud je na ně odkázano jménem.
  - Příklad: "Relace rodiče."

```
fun parents []: Person -> Person { ~children }
```

  - Příklad: "Sestry."

```
fun sisters [p: Person]:
  { {w: Woman | w in p.siblings } }
```

  - Příklad: "Žádný člověk nemůže být svým vlastním předkem nebo sestrou."

```
all p: Person |
  not (p in p.^parents or p in sisters[p])
```

Predikáty

Predikáty jsou dobré v situacích:

- Omezení, která nechcete zaznamenat jako fakta.
- Omezení, která chcte použít vícekrát
- Během analýzy se vyvolávají pouze, pokud je na ně odkázano jménem.
- Příklad: "Dvě osoby jsou pokrevní příbuzní, jestliže mají společného předka."

```
pred BloodRelated [p: Person, q: Person] {
  some p.*parents & q.*parents
}
```

- Příklad: "Osoba nemůže být v manželství s pokrevním příbuzným."

```
no p: Married | BloodRelated [p, q.spouse]
```

Příkaz Run

- **run** příkaz způsobí, že AA jeho provedením analyzuje model.
- Příkazuje nástroji, aby hledal instance modelu.
- AA provede pouze první příkaz **run** v souboru.
- AA hledá pouze v omezeném prostoru instancí určeném rozsahem.
- **Rozsah** (angl. scope) reprezentuje maximální počet n-tic v každé vrcholové signatuře.
- Přednastavená hodnota rozsahu = 3
- Příklady

```
run {}              /* the scope is 3 */
run {} for 5        /* the scope is 5 */
run {some Man && no Married}    /* with conditions */
```

Tvrzení

- Často věříme, že náš model splňuje jisté omezení, vlastnost, která není přímo vyjádřena.
- Můžeme nadefinovat taková dodatečná omezení jako tvrzení a použít AA k jejich ověření.
- Pokud omezení vyjádřené daným tvrzením není splněno, AA vyprodukuje instanci protipříkladu.
- Příklady
  - "Žádná osoba nemá rodiče, který je zároveň bratrancem či sestřenicí."

```
assert a1 {all p: Person |
  no p.parents & p.siblings }
```

I Am My Own Grandpa - Alloy řešení

```
module grandpa
abstract sig Person {
  father: lone Man,
  mother: lone Woman }
sig Man extends Person { wife: lone Woman }
sig Woman extends Person { husband: lone Man }
fact Biology { no p: Person | p in p.^(mother+father) }
fact Terminology { wife = ~husband }
fact SocialConvention {
  no wife & *(mother+father).mother
  no husband & *(mother+father).father }
fun grandpas [p: Person]: set Person {
  let parent = mother + father + father.wife + mother.husband |
  p.parent.parent & Man }
pred ownGrandpa [m: Man] { m in grandpas[m]  }
run ownGrandpa for 4 Person expect 1
```

Literatura I