MFES - Métodos Formais em Engenharia de Software

Alloy

Ana Paiva

apaiva@fe.up.pt   www.fe.up.pt/~apaiva

Join

.	*dot join*
[]	*box join*

```
e1[e2] = e2.e1
a.b.c[d] = d.(a.b.c)
```

```
Book = {(B0)}
Name = {(N0), (N1), (N2)}
Addr = {(A0), (A1), (A2)}
Host = {(H0), (H1)}

myName = {(N1)}
myAddr = {(A0)}

address = {(B0, N0, A0), (B0, N1, A0), (B0, N2, A2)}
host = {(A0, H0), (A1, H1), (A2, H1)}

Book.address = {(N0, A0), (N1, A0), (N2, A2)}
Book.address[myName] = {(A0)}
Book.address.myName = {}

host[myAddr] = {(H0)}
address.host = {(B0, N0, H0), (B0, N1, H0), (B0, N2, H1)}
```


Cartesian product

->	*cross product*

```
Name = {(N0), (N1)}
Addr = {(A0), (A1)}
Book = {(B0)}

Name->Addr = {(N0, A0), (N0, A1),
              (N1, A0), (N1, A1)}
Book->Name->Addr =
  {(B0, N0, A0), (B0, N0, A1),
   (B0, N1, A0), (B0, N1, A1)}
```

```
b  = {(B0)}
b' = {(B1)}
address  = {(N0, A0), (N1, A1)}
address' = {(N2, A2)}

b->b' = {(B0, B1)}

b->address + b'->address' =
  {(B0, N0, A0), (B0, N1, A1), (B1, N2, A2)}
```


Restriction and override

<:	*domain restriction*
:>	*range restriction*
++	*override*

```
p ++ q =
p - (domain(q) <: p) + q
```

```
Name    = {(N0), (N1), (N2)}
Alias   = {(N0), (N1)}
Addr    = {(A0)}
address = {(N0, N1), (N1, N2), (N2, A0)}

address :> Addr  = {(N2, A0)}
Alias <: address = address :> Name  = {(N0, N1), (N1, N2)}
address :> Alias =

workAddress = {(N0, N1), (N1, A0)}
address ++ workAddress = {(N0, N1), (N1, A0), (N2, A0)}
```

```
m' = m ++ (k -> v)
```
update map m with key-value pair (k, v)

Unary operators

~	*transpose*
^	*transitive closure*
*	*reflexive transitive closure*

apply only to binary relations

```
^r = r + r.r + r.r.r + …
*r = iden + ^r
```

```
Node = {(N0), (N1), (N2), (N3)}
next = {(N0, N1), (N1, N2), (N2, N3)}

~next = {(N1, N0), (N2, N1), (N3, N2)}
^next = {(N0, N1), (N0, N2), (N0, N3),
         (N1, N2), (N1, N3),
         (N2, N3)}
*next = {(N0, N0), (N0, N1), (N0, N2), (N0, N3),
         (N1, N1), (N1, N2), (N1, N3),
         (N2, N2), (N2, N3), (N3, N3)}
```

```
first = {(N0)}
rest = {(N1), (N2), (N3)}

first.^next = rest
first.*next = Node
```


Transpose (*inversa*)

A binary relation *r* is *symmetric* if, whenever it contains the tuple *a*->*b*, it also contains the tuple *b*->*a*, or more succinctly as a relational constraint:

~r **in** r

A binary relation is *transitive* if, whenever it contains the tuples *a*->*b* and *b*->*c*, it also contains *a*->*c*, or more succinctly as a relational constraint:

r.r **in** r

A binary relation *r* is *reflexive* if it contains the tuple *a*->*a* for every atom *a*, or as a relational constraint,

iden in r

Transpose (*inversa*)

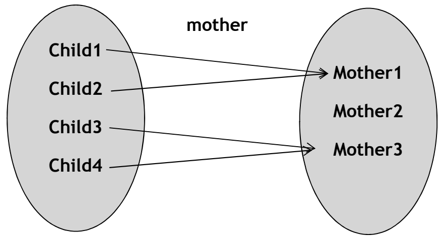
Some useful facts about transpose:

- *s.~r* is equal to *r.s*, and is the image of the set *s* navigating backward through the relation *r*;
- *r.~r* is the relation that associates two atoms in the domain of the relation *r* when they map to a common element; when *r* is a function, *r.~r* is the equivalence relation that equates atoms with the same image.
- *r.~r in iden* therefore says that *r* is injective, and *~r.r in iden* says that *r* is functional.

Transpose (*inversa*)

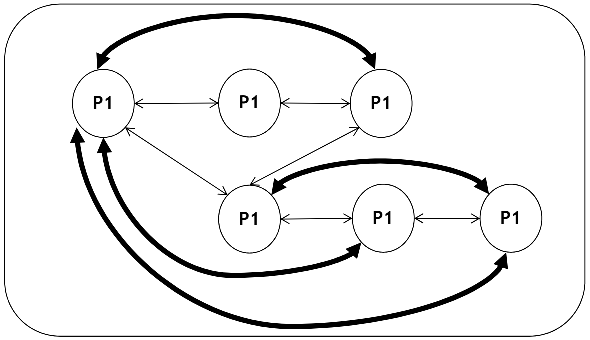
Example:

- If mother is the relation that maps a child to its mother, the expression mother.~mother is the sibling (“irmãos”) relation that maps a child to its siblings (and also to itself).



Closures

- No recursion… but we have closures
- ^R = R + R.R + R.R.R + …
- *R = ^R + iden



Transitive closure ^

- The transitive closure **^r** of a binary relation **r**, or just the closure for short, is the smallest relation that contains **r** and is transitive. You can compute the closure by taking the relation, adding the join of the relation with itself, then adding the join of the relation with that, and so on

^r = r + r.r + r.r.r + ...

Transitive-reflexive closure *

- The reflexive-transitive closure ***r** is the smallest relation that contains **r** and is both transitive and reflexive, and is obtained by adding the identity relation to the transitive closure

***r = ^r + iden**

Closures - example

R = {(G0, A0), (G0, G1), (A0, D0), (G1, D0), (G1, A1),

(A1, D1), (A2, D2)}

^R = {(G0, A0), (G0, G1), (A0, D0), (G1, D0), (G1, A1),

(A1, D1), (A2, D2),

(G0, D0), (G0, A1), (G0, D1)

(G1, D1)}

Closures - example

- Example:
  b = {(B0)}
  addr = {(B0, N0, N1), (B0, N1, D0), (B1, N1, D1)}
- The expression **^(b.addr)**, denoting the direct and indirect mapping of names in book b to the names and addresses reachable, will map names to names and addresses:
  b.addr = {(N0,N1), (N1,D0)}
  ^(b.addr) = {(N0, N1), (N1, D0), (N0, D0)}
- The expression ***(b.addr)** will include the tuples of both these relations. In addition to tuples such as (N0, N0), which are expected, it will also includes tuples such as (B0, B0).

Operators - examples

File = {(F1),(F2),(F3)}
Dir = {(D1),(D2)}
root = {(D1)}
new = {(F3,D2),(F1,D1),(F2,D1)}
parent = {(F1,D1),(D2,D1),(F2,D2)}
File + Dir = {(F1),(F2),(F3),(D1),(D2)}
parent + new = {(F1,D1),(D2,D1),(F2,D2),(F3,D2),(F2,D1)}
parent ++ new = {(F1,D1),(D2,D1),(F3,D2),(F2,D1)}
parent - new = {(D2,D1),(F2,D2)}
parent & new = {(F1,D1)}
parent :> root = {(F1,D1),(D2,D1)}
File -> root = {(F1,D1),(F2,D1),(F3,D1)}
new -> Dir = {(F3,D2,D1),(F3,D2,D2),(F1,D1,D1),...}
~parent = {(D1,F1),(D1,D2),(D2,F2)}