Julien Brunel, David Chemouil, Alcino Cunha, Eunsuk Kang, Nuno Macedo

FORMAL SOFTWARE DESIGN WITH ALLOY AND ELECTRUM

RELATIONAL LOGIC

Universidade do Minho & INESC TEC

ONERA DTIS & Université fédérale de Toulouse

Carnegie Mellon University

3rd World Congress on Formal Methods, Porto, Portugal, October 2019

RELATIONS

EXAMPLE: SOCIAL NETWORK

SOCIAL NETWORK MODEL

```
sig User {}
sig Post {}
```

A signature defines a set of objects.

EVERYTHING IS A RELATION!

```
User = {(U0),(U1)}
```

User
U0
U1

```
Post = {(P0),(P1),(P2)}
```

Post
P0
P1
P2

RELATIONS AS TABLES

```
posts = {(U0,P0),(U0,P0),(U1,P1)}
```

User	Post
U0	P0
U0	P2
U1	P0
U1	P1

RELATIONAL JOIN

$$p.q = \{(p_1, \dots\ p_{n-1}, q_2, \dots\ q_m) \mid (p_1, \dots\ p_{n-1}, p_n) \in p \wedge (q_1, q_2, \dots\ q_m) \in q \wedge p_n = q_1\}$$

`posts = {(U0,P0),(U0,P0),(U1,P1)}`

User	Post
U0	P0
U0	P2
U1	P0
U1	P1

`U0.posts` = ??

`posts.P0` = ??

`User.posts` = ??

TRANSITIVE CLOSURE

$\hat{}R = R \cup R.R \cup R.R.R \cup R.R.R.R \cup \ldots$

```
friends = {(U0,U1),(U1,U0),(U1,U2),(U2,U1)}
```

User	User
U0	U1
U1	U0
U1	U2
U2	U1

`^friends` = ??

`U0.^friends` = ??

SIGNATURE FIELDS

```
sig SocialNetwork {
  friends : User -> User,
  posts : User -> Post
}
```

`posts` is a ternary relation from `SocialNetwork` to `User` to `Post`

SocialNetwork	User	Post
N0	U0	P0
N0	U0	P2
N0	U1	P0
N0	U1	P1

SUBTYPING

```
abstract sig Post {}
sig Photo, Video, Text extends Post {}
```

SEMANTICS

Meaning of an Alloy model: The set of all satisfying instances

- Each instance contains a universe of objects from signatures.
- Each relation is interpreted with a number of tuples (possibly empty).
- Each instance satisfies all given constraints.

CONSTRAINTS

```
fact friendshipIsSymmetric {
    all n : SocialNetwork, u1, u2 : User |
        u1 -> u2 in s.friends implies
            u2 -> u1 in s.friends
}
```

or

```
fact friendshipIsSymmetric {
    friends = ~friends
}
```

A fact imposes a constraint that must be satisfied by every instance.

PREDICATES

```
pred invariant[n : SocialNetwork] {
    // Each post is owned by at most one user
    all p : Post | lone n.posts.p
    // A user cannot be his or her own friend
    all u : User | u -> u not in n.friends
    // Friendship is a symmetric relation
    n.friends = ~(n.friends)
}
```

A predicate is a construct for packaging and reusing constraints.

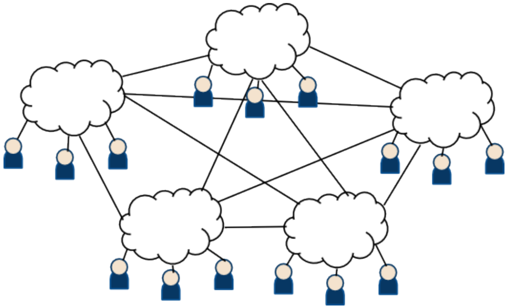
GENERATING AN INSTANCE

```
run generateValidSocialNetwork {
    some n : SocialNetwork | invariant[n]
}
```

or

```
run invariant
```

SOCIAL NETWORK AS A DISTRIBUTED SYSTEM



User data is distributed across multiple servers

DISTRIBUTED SOCIAL NETWORK

```
sig User {}
sig Post {}
sig DistributedSN {
    servers : set Server,
    friends : User -> User
}
sig Server {
    posts : User -> Post,
    capacity : Int
}
```

DISTRIBUTED SOCIAL NETWORK

What are the invariants for the distributed version?

EXERCISES

https://github.com/haslab/Electrum2/wiki/Social-Network

(exercises 1-2)