

# Specification with Alloy — Foundations

Jan van Eijck

[jve@cwi.nl](mailto:jve@cwi.nl)

Master SE, 11 November 2008

## Automating First Order Relational Logic

- Relational logic = First Order Logic plus Relational Operators.
- Most relational operations are expressible in first order logic, but not all of them.
- Relation composition and relation transpose can be expressed in first order logic:
  - $r.s$  can be expressed as  $\{(x, y) \mid \exists z R(x, z) \wedge S(z, y)\}$ .
  - In Alloy:  
 $x \rightarrow y$  in  $r.s$  iff some  $z$  |  $x \rightarrow z$  in  $r$  and  $z \rightarrow y$  in  $s$
- $\sim R$  can be expressed as  $\{(x, y) \mid R(y, x)\}$ .
- In Alloy:  
 $x \rightarrow y$  in  $\sim r$  iff  $y \rightarrow x$  in  $r$

## Beyond First Order Logic: Transitive Closure

- Transitive closure and reflexive transitive closure cannot be expressed in first order logic.
- The transitive closure of  $R$  is the **smallest relation**  $S$  for which:
  - $R \subseteq S$ ,
  - $S$  is transitive.
- To express  $\hat{r}$  one would need an 'infinite formula':

$$\{(x, y) \mid R(x, y) \vee \exists z(R(x, z) \wedge R(z, y)) \\ \vee \exists z, v(R(x, z) \wedge R(z, v) \wedge R(v, y)) \\ \vee \exists z, v, w(R(x, z) \wedge R(z, v) \wedge R(v, w) \wedge R(w, y)) \\ \vee \dots \}$$

## Propositional Logic

- Propositional logic: logic of propositions.
- Example formulas:
  - $p$ ,
  - $p \vee q$ ,
  - $p \rightarrow p \vee q$ ,
  - $p \vee q \Leftrightarrow \neg(\neg p \wedge \neg q)$ .
- Why is propositional logic decidable and first order logic undecidable?
- Let us first see how first order logic is different from propositional logic.

## SAT

- The following questions about propositional formulas are equivalent:
  - $F_1$  implies  $F_2$ ,
  - $F_1 \rightarrow F_2$  is true for every valuation.
  - $F_1 \wedge \neg F_2$  is not satisfiable.
- The satisfiability problem for propositional logic is called SAT.

## Decidability of Propositional Logic

- SAT is decidable
- Decision algorithm to check SAT of  $F$ :  
Any propositional formula mentions only a finite number of proposition letters. Say the proposition letters mentioned in  $F$  are  $p_1, \dots, p_n$ .  
There are  $2^n$  possible valuations for these letters.  
Just work out the truth value of  $F$  for each valuation (using the truth table method) to see if one of them is satisfiable.  
If this is the case, answer 'yes', otherwise answer 'no'.

## The Truth Table Method

For instance, look at the formula

$$\neg p \wedge ((p \rightarrow q) \Leftrightarrow \neg(q \wedge \neg p)).$$

Suppose  $p$  has value **1** and  $q$  has value **0**, then we get (using the truth tables):

- $\neg p$  has **0**,
- $p \rightarrow q$  has **0**,
- $q \wedge \neg p$  has **0**;
- $\neg(q \wedge \neg p)$  has **1**;
- $(p \rightarrow q) \Leftrightarrow \neg(q \wedge \neg p)$  has **0**,
- the whole expression has **0**.

$$\begin{array}{cccccccccccc}
\neg & p & \wedge & ((p & \rightarrow & q) & \Leftrightarrow & \neg & (q & \wedge & \neg & p)) \\
\vdots & \mathbf{1} & \vdots & \mathbf{1} & \vdots & \mathbf{0} & \vdots & \vdots & \mathbf{0} & \vdots & \vdots & \mathbf{1} \\
\mathbf{0} & & \vdots & & \mathbf{0} & & \vdots & \vdots & & \vdots & \mathbf{0} & \\
& & \vdots & & & & \vdots & \vdots & & \mathbf{0} & & \\
& & \vdots & & & & \vdots & \mathbf{1} & & & & \\
& & \vdots & & & & \mathbf{0} & & & & & \\
& & \mathbf{0} & & & & & & & & & 
\end{array}$$

In compressed form:

$$\begin{array}{cccccccccccc}
\neg & p & \wedge & ((p & \rightarrow & q) & \Leftrightarrow & \neg & (p & \wedge & \neg & p)) \\
\mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}
\end{array}$$

The method given above (the truth table method) can also be used as a decision algorithm for propositional consequence.

## First Order Logic

- Now look at the case of first order logic.
- A truth table method does not work here.
- Instead of **valuations** we need **models**.
- A model has a **domain**, and **interpretations** for all predicates and relations.
- Look at  $\forall x, y, z (Rxy \wedge Ryz \rightarrow Rxz)$
- Alloy version:

```
all x,y,z: Entity |  
    x->y in r and y->z in r implies x->z in r
```

A model for this has a domain and a **transitive** relation on that domain.

- Try this out in Alloy ...

## Alloy Example

```
module myexamples/rel_t

abstract sig Entity { r: set Entity }

one sig A, B, C, D extends Entity {}

fact r_transitive {
  all x,y,z: Entity |
    x->y in r and y->z in r implies x->z in r }

pred show () {}
run show
```

## Formulas and Models

- Consider the above Alloy specification.
- The result of executing **run show** for this specification is a model with a transitive relation  $r$  on it.
- Think of the **Alloy specification** as a **formula**.
- Think of the **picture** that results from executing **run show** as a **model** for that formula.

## Another Way to Express Transitivity

```
module myexamples/rel_tt

abstract sig Entity { r: set Entity }

one sig A, B, C, D extends Entity {}

fact r_transitive {
  all x,y,z: Entity |
    x->y in r and y->z in r implies x->z in r }

assert r_transitive' { r.r in r }
check r_transitive'
pred show () {}
run show
```

## Formulas with Only Infinite Models

- Consider the conjunction of:
  - $\forall x, y(Rxy \rightarrow \neg Ryx)$  ( $R$  is asymmetric)
  - $\forall x \exists y Rxy$  ( $R$  is serial)
  - $\forall x, y, z(Rxy \wedge Ryz \rightarrow Rxz)$  ( $R$  is transitive).
- Suppose our domain is non-empty.
- Then every model of this conjunction is **infinite**. Why?
- The task of checking **all relational structures** (including infinite ones) in search for a model of a formula cannot be finished in a finite amount of time.

## Consistency, Refutation of Consistency

- A first order formula is **consistent** if it has a model.
- The existence for formulas with only infinite models suggests that first order consistency is not decidable.
- In fact, we have a semi-decision method: if a formula is inconsistent the method will determine this after finitely many steps.
- The method consists of constructing a so-called semantic tableau. This boils down to a systematic search for an inconsistency.
- There are consistent formulas for which the method loops. The refutation method for consistency is not an algorithm.
- Note that nothing we have said above is a **proof** that a decision method for first order consistency **cannot exist**.

## Undecidable Queries

- The deep reason behind the undecidability of first order logic is the fact that its expressive power is so great that it is possible to state undecidable queries.
- One of the famous undecidable queries is the **halting problem**.
- Here is what a **halting algorithm** would look like:
  - Input: a specification of a computational procedure  $P$ , and an input  $C$  for that procedure
  - Output: an answer 'yes' if  $P$  halts when applied to  $C$ , and 'no' otherwise.

## Undecidability of the Halting Problem

- Suppose there is an algorithm to solve the halting problem. Call this  $H$ .
- Then  $H$  takes a computational procedure as input, together with an input to that procedure, and decides.
- Consider the following new procedure  $N$  for processing inputs  $C$ : If  $H$  says that  $C$  applied to  $C$  halts, then loop, otherwise (if  $H$  says that  $C$  applied to  $C$  does not halt) print 'halted' and terminate.

## Undecidability of the Halting Problem (ctd)

- What does  $N$  do when applied to  $N$  itself?
- Suppose  $N$  halts on input  $N$ . Then  $H$  should answer 'yes' when  $H$  is applied to  $N$  with input  $N$ , for  $H$  is supposed to be a correct halting algorithm. But then, by construction of the procedure,  $N$  should loop.
- Suppose  $N$  loops on input  $N$ . Then  $H$  should answer 'no' when  $H$  is applied to  $N$  with input  $N$ , for  $H$  is supposed to be a correct halting algorithm. But then, by construction of the procedure,  $N$  should print 'halted' and stop.
- We have a contradiction. Therefore a halting algorithm  $H$  cannot exist. A language that allows the specification of 'universal procedures' such as  $H$  and  $N$  cannot be decidable.

## Proof of Undecidability of First Order Logic

- The formal proof of the undecidability of first order logic consists of
  - A **very general** definition of **computational procedures**.
  - A demonstration of the fact that such computational procedures can be expressed in first order logic.
  - A demonstration of the fact that the halting problem for computational procedures is undecidable (see the above sketch).
  - A formulation of the halting problem in first order logic.
- This formal proof was provided by Alan Turing in **Turing [1936]**. The computational procedures he defined for this purpose were later called **Turing machines**.

## Back to Alloy

- Alloy is not a decision method for first order logic
- Alloy translates first order logic with (small) finite scopes into propositional logic.
- Consistency of these translations can be decided, for propositional logic is decidable.
- SAT is intractable.
- More precisely, if one adds a single proposition letter, the computation time used by the truth table method doubles (for the truth tables become twice as big).
- This means that the truth table method is an exponential algorithm.

- It is very likely that all other methods for solving SAT are exponential, for SAT is NP-hard.

## Sat and P=NP

- If we can solve SAT in polynomial time, we have solved the  $P = NP$  problem.
- $P = NP$  is widely believed to be false, although nobody has been able to give a proof of this.
- What this means it that we should not expect the Alloy method to scale up.
- All future versions of Alloy will still only work for small domains.
- For a domain of size  $k$ , the result of adding an extra binary relation  $R$  to the signature is that  $2^{k^2}$  possibilities for the interpretation of  $R$  have to be investigated.

## The Alloy Type System

$$\frac{E \vdash a : S \quad E \vdash b : S}{E \vdash a \text{ in } b}$$

$$\frac{E, v : T \vdash F}{E \vdash \text{all } v : T \mid F}$$

$$\frac{E \vdash a : S \rightarrow T \quad E \vdash b : S \rightarrow T}{E \vdash a + b : S \rightarrow T}$$

$$\frac{E \vdash a : S \rightarrow U \quad E \vdash b : U \rightarrow T}{E \vdash a.b : S \rightarrow T}$$

$$\frac{E \vdash a : S \rightarrow T}{E \vdash \sim a : T \rightarrow S} \quad \frac{E \vdash a : T \rightarrow T}{E \vdash \hat{a} : T \rightarrow T}$$

$$\frac{E, v : T \vdash F}{E \vdash \{v : T \mid F\} : T}$$

## Steps of the Alloy Analysis

1. Conversion to negation normal form and skolemization.
2. Translation (for a chosen scope) to a formula of propositional logic (a Boolean formula). Mapping between relational variables and Boolean variables is preserved.
3. Conversion of Boolean formula to conjunctive normal form.
4. CNF of Boolean formula fed to SAT solver.
5. If SAT solver finds a model, a first order version of this is constructed using the mapping from 2.

## Checking Relational Properties with Alloy

- Asymmetry of a relation:  $\forall x \forall y (Rxy \rightarrow \neg Ryx)$ .
- Irreflexivity of a relation:  $\forall x \neg Rxx$ .
- Every asymmetric relation is irreflexive.

```
module myexamples/rel_asym

abstract sig Entity { r : set Entity }
one sig A, B, C, D extends Entity {}
fact r_asymmetric { no ~r & r }
assert r_irreflexive { no iden & r }
check r_irreflexive
pred show () { }
run show
```

## Checking Relational Properties (ctd)

Do transitivity and symmetry together imply reflexivity?

```
module myexamples/rel_ts

abstract sig Entity { r : set Entity }
one sig A, B, C, D extends Entity {}

fact r_transitive { r.r in r }
fact r_symmetric { ~r in r }

pred show () { }
run show
assert r_reflexive { iden in r }
check r_reflexive
```

## Checking Relational Properties (ctd)

Do transitivity, symmetry and seriality together imply reflexivity?

```
module myexamples/rel_tss
abstract sig Entity { r : set Entity }
one sig A, B, C, D extends Entity {}

fact r_transitive { r.r in r }
fact r_symmetric { ~r in r }
fact r_serial
  { all x: Entity | some y: Entity | x->y in r }

pred show () { }
run show
assert r_reflexive { iden in r }
check r_reflexive
```

## Oops, Unexpected Alloy Behaviour

We get: counterexample found. Assertion is invalid.

This is strange, for the assertion **is** valid. If we check inspect counterexample, then we see what is in fact a reflexive relation. **Very strange.**

## Debugging Alloy

Let's try a variation on the program:

```
module myexamples/rel_tss
```

```
abstract sig Entity { r : set Entity }
```

```
one sig A, B, C, D extends Entity { }
```

```
pred r_trans { r.r in r }
```

```
pred r_symm { ~r in r }
```

```
pred r_serial { all x: Entity | some y: Entity | x->y in r }
```

```
pred r_refl { iden in r }
```

```
TSSimplifiesR: check {
```

```
r_trans and r_symm and r_serial => r_refl  
}
```

Again, we get a report of a counterexample.

Again, if we inspect the 'counterexample', it turns out to be an example of a reflexive relation.

## Debugging Alloy (ctd)

Let us do a further check. Select 'show' and open the Evaluator.

We get:

```
Eval> r
```

```
{A$0->A$0, A$0->C$0, B$0->B$0, C$0->A$0, C$0->C$0,
D$0->D$0}
```

This is our example of a reflexive relation. Let us check the **iden** relation:

```
Eval> iden
```

```
{-8->-8, -7->-7, -6->-6, -5->-5, -4->-4, -3->-3,
-2->-2, -1->-1, 0->0, 1->1, 2->2, 3->3, 4->4,
5->5, 6->6, 7->7, A$0->A$0, B$0->B$0, C$0->C$0,
D$0->D$0}
```

Oops, the identity is taken over a larger universe. Now it is no surprise that `iden` in `r` fails:

```
Eval> iden - r
```

```
{-8->-8, -7->-7, -6->-6, -5->-5, -4->-4, -3->-3,  
-2->-2, -1->-1, 0->0, 1->1, 2->2, 3->3, 4->4,  
5->5, 6->6, 7->7}
```

## Debugging Alloy (end)

OK, so the bug was in our definition of reflexivity. Now that we see this we can solve the problem:

```
module myexamples/rel_tss
```

```
abstract sig Entity { r : set Entity }
```

```
one sig A, B, C, D extends Entity { }
```

```
pred r_trans { r.r in r }
```

```
pred r_symm { ~r in r }
```

```
pred r_serial { all x: Entity | some y: Entity | x->y in r }
```

```
pred r_refl { iden in r }
```

```
pred r_refl' { all x: Entity | x -> x in r }
```

```
TSSimpliesR: check {  
  r_trans and r_symm and r_serial => r_refl  
}
```

```
TSSimpliesR': check {  
  r_trans and r_symm and r_serial => r_refl'  
}
```

## Alloy Lab Exercise for This Week

Make up a specification of the Amsterdam metro system in Alloy. The following stations should be mentioned:

CS, Nieuwmarkt, Waterlooplein, Amstel, Duivendrecht, Arena, Bijlmer, Gein, Overamstel, WTC, VU, Westwijk, Lelylaan, Sloterdijk, Isolatorweg, Diemen, Gaasperplas.

Right now, the Amsterdam metro consists of four lines: 50 (Isolatorweg – Gein), 51 (CS – Westwijk), 53 (CS – Gaasperplas), and 54 (CS – Gein). Line 52, the so-called north-south line, is under construction. See <http://www.gvb.nl/reizigers/plattegronden/Pages/metrokaart.aspx> for the current situation and <http://urbanrail.net/eu/ams/amsterdm.htm> for a map that includes (the as yet non-existing) line 52.

Your specification should formalize statements like the following:

- One can get from station  $S_1$  to station  $S_2$  without a stopover. (e.g., from CS to VU).
- Station  $S$  is served by line  $L$ .
- One can get from  $S_1$  to  $S_2$  by means of a single stopover at  $S_3$ .
- One can get from  $S_1$  to  $S_2$  by means of two stopovers, at  $S_3$  and  $S_4$ .
- The ends of line  $L$  are stations  $S_1$  (at the start) and  $S_2$  (at the end).

The snapshot of the system should be correct. Model the system in such a way that it can be extended by putting in additional stations (note that the above list of stations is not complete). Finally, put in some appropriate assertions to test your specification.

Deliverable: Alloy file, plus indication of the time spent.

## More Debugging

Chapters 5 and 6 of Zeller, with exercises

## Next Week

- More on Alloy and on What is Behind it
- Further Examples of Alloy Use
- Learning to use Assertions for Random Test Generation
- Specification Based Automated Testing ...

## References

A.M. Turing. On computable real numbers, with an application to the Entscheidungsproblem. [Proceedings of the London Mathematical Society](#), 2(42):230–265, 1936.