Alloy - Part II
Dynamic Models and Automation

Klaus Reisenberger

JKU Linz

Klaus.Reisenberger@gmx.at

06.03.2015

Outline

Dynamic Modeling

Translation of a First-Order Formula to a Quantifier-free Boolean Formula

Section 1

Dynamic Modeling

Static vs Dynamic Modeling

- Static models
  - Describe states (file system)
- Dynamic models
  - Describe transitions between states

Example - River Crossing Part I

```
/* Impose an ordering on the State. */
open util/ordering[State]

/* Farmer and his possessions are objects. */
abstract sig Object { eats: set Object }
one sig Farmer, Fox, Chicken, Grain extends Object {}

/* Defines what eats what and the farmer is not around. */
fact { eats = Fox->Chicken + Chicken->Grain}

/* Stores the objects at near and far side of river. */
sig State { near, far: set Object }

/* In the initial state, all objects are on the near side. */
fact { first.near = Object && no first.far }

/* At most one item to move from 'from' to 'to' */
pred crossRiver [from, from', to, to': set Object] {
  one x: from | {
    from' = from - x - Farmer - from'.eats
    to' = to + x + Farmer
  }
}
```

Example - River Crossing Part II

```
/* crossRiver transitions between states */
fact {
  all s: State, s': s.next {
    Farmer in s.near =>
      crossRiver [s.near, s'.near, s.far, s'.far]
    else
      crossRiver [s.far, s'.far, s.near, s'.near]
  }
}

/* the farmer moves everything to the far side of the river. */
run { last.far=Object } for exactly 8 State
```

Section 2

Translation of a First-Order Formula to a Quantifier-free Boolean Formula

Subsection 1

Introduction

Introduction

Automatic analysis for relational logic:

Input:

Formula and scope

Output:

Checks whether a model exists and if so returns it

- Undecidable
- Can be used in 2 ways:
  - Check consistency of a formula
  - Check validity of a theorem

Overview

- Syntax
- Semantics
- Analysis
- Performance
- Future work

Syntax

```
problem ::= decl* formula
decl ::= var : typexpr
typexpr ::=
  type                        set
  | type -> type              relation
  | type => typexpr           function

formula ::=
  expr in expr                subset
  | ! formula                 negation
  |  formula && formula       conjunction
  |  formula || formula       disjunction
  |  all v : type | formula   universal
  |  some v : type | formula  existential
```

```
expr ::=
  | expr + expr               union
  | expr & expr               intersection
  | expr - expr               difference
  | expr . expr               navigation
  | ~ expr                    transpose
  | + expr                    closure
  | {v : t | formula}         comprehension
  | Var

Var ::=
  | var                       variable
  | Var [var]                 application
```

Semantics

M : formula $\rightarrow$ env $\rightarrow$ boolean
X : expr $\rightarrow$ env $\rightarrow$ value
env = (var + type) $\rightarrow$ value
value = P (atom $\times$ atom) + (atom $\rightarrow$ value)

M [a in b] e = X[a] e $\subseteq$ X[b] e
M [! F] e = $\neg$ M [F] e
M [F && G] e = M [F] e $\wedge$ M [G] e
M [F || G] e = M [F] e $\vee$ M [G] e
M [all v: t | F] e = $\bigwedge$ {M[F](e$\oplus$ v $\mapsto$ x) | (x,unit) $\in$ e(t)}
M [some v: t | F] e = $\bigvee$ {M[F](e$\oplus$ v $\mapsto$ x) | (x, unit) $\in$ e(t)}

X [a + b] e = X[a]e $\cup$ X[b]e
X [a & b] e = X[a]e $\cap$ X[b]e
X [a - b] e = X[a]e \ X[b]e
X [a . b] e = {(x,z) | $\exists$y. (y,z) $\in$ X[a]e $\wedge$ (y,x) $\in$ X[b]e}
X [~a] e = {(x,y) | (y,x) $\in$ X[a]e}
X [+a] e = the smallest r such that r ; r $\subseteq$ r $\wedge$ X[a]e $\subseteq$ r
X [{v: t | F}] e = {(x,unit) $\in$ e(t) | M[F](e$\oplus$ v $\mapsto$ x)}
X [v] e = e(v)
X [a[v]] e= (e(a))(e(v))

Example

```
all x: X | some y: Y | x.r = y
```

Subsection 2

Analysis

5 Steps of Analysis

- Conversion to negation normal form and skolemization
- Translation
- Conversion to CNF
- Handover to SAT solver
- Construction of a model of the relational formula

Normalization of the Relational Formula

- Convert to NNF (negation normal form)
- Skolemize it

Example

```
!all x: X | some y: Y | x.r=y

some x: X | all y: Y | !x.r=y

all y: Y | !x.r=y
some z: X | z=x
```

Overview of the Translation

Input:
Relational formula
Output:
Boolean formula for a given scope

- Represent relations as matrices of boolean values
- Constraints on relations can be expressed as boolean formulas

Translation Rules

MT : formula $\rightarrow$ booleanFormula tree
XT : expr $\rightarrow$ value tree
α tree = (var $\times$ (index $\rightarrow$ α tree)) + α
value = booleanFormulaMatrix + (index $\rightarrow$ value)

MT [a in b] = merge (MT[a], MT[b], $\lambda p,q. \wedge_{ij} \{p_{ij} \Rightarrow q_{ij}\}$)
MT [! f] = map (MT[f], $\neg$)
MT [f && g] = merge (MT[f], MT[f], $\wedge$)
MT [f || g] = merge (MT[f], MT[f], $\vee$)
MT [all v: t | f] = fold (MT[f], $\bigwedge$)
MT [some v: t | f] = fold (MT[f], $\bigvee$)

XT [a + b] = merge (XT[a], XT[b], $\lambda p,q.\mu r.\ r_{ij} = p_{ij} \vee q_{ij}$)
XT [a & b] = merge (XT[a], XT[b], $\lambda p,q.\mu r.\ r_{ij} = p_{ij} \wedge q_{ij}$)
XT [a - b] = merge (XT[a], XT[b], $\lambda p,q.\mu r.\ r_{ij} = p_{ij} \wedge \neg q_{ij}$)
XT [a . b] = merge (XT[a], XT[b], $\lambda p,q.\mu r.\ r_{ij} = \exists k.\ p_{ik} \wedge q_{kj}$)
XT [~a] = map (XT[a], $\lambda p.(r \mid r_{ij} = p_{ji})$)
XT [+a] = map (XT[a], closure)
XT [{v: t | f] = fold (MT[f], $\lambda f.\ \mu r.\ r_{0i} = f(i)$)
XT [a[v]] = merge (XT[a], XT[v], $\lambda s,x.\ \mu s_{i}.\ x_{ji}$)
XT [v] = (v, $\lambda i.(\mu r.\ r_{0j} = (i = j))$) — when v is quantified
XT [v] = create (v) — otherwise

merge : α tree, α tree, $(\alpha,\alpha \rightarrow \beta) \rightarrow \beta$ tree
merge (x, y, o) = o(x, y)
merge ((u,t1), (u,t2), o) = (u, λi. merge(t1(i),t2(i),o))
merge ((u,t1), (v,t2), o) = (u, λi. merge(t1(i),(v,t2),o)) — when u < v
merge ((u,t1), (v,t2), o) = (v, λi. merge((u,t1),t2(i),o)) — when v < u
merge ((u,t), y, o) = (u, λi. merge(t(i),y ,o))
merge (x, (v,t), o) = (v, λi. merge(x,t(i),o))

map : α tree, $(\alpha \rightarrow \alpha) \rightarrow \alpha$ tree
map (x, o) = o(x)
map ((u,t), o) = (u, λi.map (t(i),o))

fold : α tree, $((\text{index} \rightarrow \alpha) \rightarrow \beta) \rightarrow \beta$ tree
fold ((u,t), o) = o(t) — when t(i) elementary
fold ((u,t), o) = (u, λi. fold(t (i),o)) — otherwise

create: var $\rightarrow$ value
create (v) = ($r \mid r_{0i}$ = a fresh boolean variable F(v,i)) — for v: S
create (v) = ($r \mid r_{ij}$ = a fresh boolean variable F(v,i,j)) — for v: S $\rightarrow$ T
create (v) = ($r \mid r_{i}$ = create (v_i)) — for v: S $\Rightarrow$ t

Example Translation

Example

```
all y: Y | !x.r = y
```

with a scope of 2

Conversion to CNF, Solving and Mapping Back

The following steps are:

- Convert to CNF and pass to the solver
- If a solution exists we can reconstruct a model of the relational formula

Performance

Three models originally written in NP

- Finder, a toy model for a Macintosh file system
- Style, a model of an aspect of the paragraph style mechanism of Microsoft Word
- Mobile IP, a model that exposed a flaw in an internet protocol for forwarding messages to mobile hosts

Example	*Scope*	*Space*	*Old* [16]	*New*
Finder	5	160 bits	3s	9s
	6	216	162s	13s
Style	3	90	1s	3s
	4	156	2s	3s
	5	250	11s	6s
Mobile IP	3	175	0s	1s
	4	280	3s	6s
	5	600	8s	29s

Future Work - KodKod

Kodkod is a new relational engine designed expressly as a plugin component that can easily be incorporated as a backend of another tool.

- Want to use Alloy as a backend engine
- Disadvantages of current implementation:
  - A clean API
  - Support for partial instances
  - A mechanism for sharing subformulas and subexpressions

References

Jackson, Daniel:
Software Abstractions : Logic, Language, and Analysis.
Cambridge: MIT Press, 2012.

Alloy MIT Online Tutorial
Retrieved November 19, 2014, from
http://alloy.mit.edu/alloy/tutorials/online/

Jackson , Daniel:
Automating First-Order Relational Logic