Interactive Theorem Provers & Applications to Network Problems

Andreas Voellmy
andreas.voellmy@yale.edu

Yale University

April 11, 2010

Overview

1. Interactive theorem provers
2. Tutorial on Isabelle/HOL.
3. Formalization of a tiny fragment of BGP.
4. References & further work

Interactive Theorem Proving

An *interactive theorem prover* (a.k.a. a *proof assistant*) is a program which takes as input a formalized mathematical statement and a putative proof, and checks whether the proof is valid.

Key ingredients:

1. An *expressive* formal language and logic, typically a variant of higher order logic.
2. A program to check proofs and to aid in their construction.
3. A programming language to extend the system with new proof procedures (e.g. decision procedures).

Formalizing maths

- You have “proved” some mathematical theorem, but have found mistakes in it several times; now you want to be sure there are no more mistakes.
- You have made some argument, and believe the conclusion, but find that the foundations of the argument are unclear. You want to clarify the foundations.

How confident are you in Gödel’s Incompleteness Theorem? How about the Four-Color Theorem?

A machine checked proof improves our confidence in a mathematical claim.

Some significant formalized theorems

1. Gödel's First Incompleteness Theorem (N. Shankar 1986, others later)
2. Jordan Curve Theorem (T. Hales 2005, others later)
3. Prime Number Theorem (J. Avigad 2004, others later)
4. Four Color Theorem (G. Gonthier 2004)
    - In 1890, Heawood showed Kempe's 1879 "proof" was flawed.
    - In 1891, Peterson showed Tait's 1880 "proof" was flawed.

System verification

- You developed a program/protocol/hardware, but keep finding bugs. You want to be sure it is correct.
- A program/protocol/hardware seems correct, but the principles of the correctness argument are unclear. Can you clarify the logic?

How confident are you in the correctness of your C compiler / OS / TCP library / Chord protocol / security protocol / firewall policy?

Do you understand *why* your security protocol is correct and on what assumptions it depends?

Some significant system verifications

- Intel verification project: verification of floating point algorithms assuming correctness of hardware primitives. (Harrison). Started after an error in a floating-point division instruction of some Intel Pentium processors cost Intel $500 million in 1994.
- CompCert: verified compiler of a realistic subset of the C language; the generated machine code behaves according to the semantics of the source C program. (Leroy, CACM 2009)
- seL4: Formal verification of an OS Kernel, assuming correctness of C compiler, assembly code and hardware. (Klein et al. SOSP 2009)

Verification method - in general

1. Model the system - detailed and complex!;
2. Express all the assumptions;
3. Formalize the specification - complex again; getting this wrong means building the wrong program!;
4. Verify: Prove that the modelled system satisfies the spec - Long detailed proofs are error-prone; may involve lots of math (e.g. floating point verification).

Computers can help:

1. Machine check proofs;
2. Automate proof tasks.

Limitations of Automatic Techniques

In special cases, we have automatic verification techniques:

- Decision procedures for certain theories (e.g. Presburger arithmetic).
- Model checking temporal formulae over finite labelled transition systems.

Many systems and specifications can not be modelled in these special cases, and the maths needed do not fall in the decidable portions of first order logic.

Theorem Proving - semi-automated

A more general method:

1. Expressive formal language, e.g. ZF set theory, Higher-order logic (HOL).
2. Proof system + proof checking program.
3. Provide known decision procedures and theorem proving algorithms.
4. Programmable proof assistant; the system can be extended with new decision procedures - in a safe way!

Theorem Provers

There are lots of theorem provers; the most widely used (arguably): HOL Light, Mizar, Isabelle, Coq.

The Edinburgh LCF project introduced a solution that many other provers followed:

- The system is implemented in an interactive programming language, giving the user power to develop new proof procedures.
- Theorems are represented by an abstract type, instances of which can only constructed by applying the logic's rules of inference, ensuring the system is safe.

Limitations of First Order Logic (FOL)

Things you *can not* say in FOL:

- One relation is the transitive closure of another relation.
- Every bounded set of reals has a least upper bound.
- A relation is a well-ordering.

One can say these in *second order logic*, e.g.:

$$\forall X(\exists y Xy \to \exists y(Xy \wedge \forall z(Xz \to y \leq z)))$$

Alternatively, one can use set theory.

Higher Order Logic (HOL) is a practical logic

HOL generalizes second order logic to *any finite order*

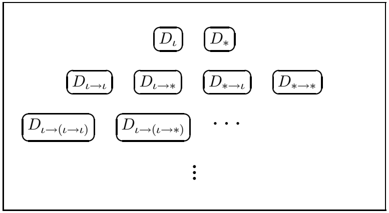
- Aka *Type Theory*. Developed by A. Church, L. Henkin, P. Andrews and others.
- Expressive - terms denote individuals, sets, relations, functions, sets of sets, sets of sets of functions, etc.
- Uniform syntax
- Simple semantics
- Simple, elegant proof system
- HOL is *practical* (W. Farmer).

HOL: Sketch of Simple Type Theory

Types $(\alpha, \beta, \ldots)$:

- denote sets;
- ι (individuals),
- $*$ (truth values),
- $\alpha \to \beta$ (functions)

Models:



Expressions $(A_\alpha, B_\beta, \ldots)$ over (C, τ):

- have a type and denote members of their type.
- x_α (variable), $c_{\tau(c)}$ (constants),
- $(\lambda x_\alpha B_\beta)_{\alpha\to\beta}$
- $(f_{\alpha\to\beta} x_\alpha)_\beta$
- $A_\alpha = B_\alpha$

(Diagram from W. Farmer “Seven Virtues of Simple Type Theory”)

Higher Order Logic - Examples

Sets of elements of type α are represented by their characteristic functions, of type $\alpha \rightarrow *$.

Functions of two arguments represented in curried form $\alpha \rightarrow \alpha \rightarrow \beta$.

Binary: Relations: $\alpha \rightarrow \alpha \rightarrow *$.

Completeness principle of the real numbers:

$$\forall S.((\exists x.S(x)) \wedge (\exists x.x\ \mathit{ub}\ S)) \longrightarrow \exists x.x\ \mathit{lub}\ S$$

Isabelle: a generic theorem prover

Architecture:

- *Standard ML*: Supports meta-logic implementation; Allows users to extend the system with new decision procedures
- *Meta Logic*: Generic, interactive theorem prover
- *Object logics*: FOL, HOL, ZF
- *Proof General*: User Interface

HOL is the most important object logic:

$$\textit{Isabelle/HOL} = \textit{Logic} + \textit{Functional Programming}$$

(Following few slides' contents taken from C. Ballarin and G. Klein http://isabelle.in.tum.de/coursematerial/IJCAR04/index.html)

Isabelle Syntax: types & terms

$$\begin{array}{rcl} \tau ::= & & 'a \mid 'b \mid \ldots \\ & \mid & \tau \Rightarrow \tau \\ & \mid & \mathit{bool} \mid \mathit{nat} \mid \ldots \\ & \mid & \tau \times \tau \\ & \mid & \tau\,\mathit{list} \\ & \mid & \ldots \end{array}$$

$$\begin{array}{rcl} \mathit{term} ::= & & x \mid y \mid \ldots \\ & \mid & \mathit{term}\ \mathit{term} \\ & \mid & \lambda x.\ \mathit{term} \\ & \mid & \ldots \end{array}$$

typedecl *name*
types *gate* $= \mathit{bool} \Rightarrow \mathit{bool} \Rightarrow \mathit{bool}$

consts *sq* $:: \mathit{nat} \Rightarrow \mathit{nat}$
defs *sq n* $\equiv n * n$

Meta-logic

Meta-logic provides basic constructs that allow one to encode other logics:

- Implication: $\Longrightarrow$
- Equality: $\equiv$
- Universal quantifier: $\bigwedge$

$$[\![A_1; \ldots; A_n]\!] \Longrightarrow B$$

is the same as

$$A_1 \Longrightarrow (A_2 \ldots \Longrightarrow (A_n \Longrightarrow B) \ldots)$$

HOL Inference Rules

A few HOL rules:

- $\bigwedge x.f(x) = g(x) \Longrightarrow \lambda x.f(x) = \lambda x.g(x)$
- $[\![P \longrightarrow Q; P]\!] \Longrightarrow Q$

Derived rules: propositional logic

- $[\![P \wedge Q]\!] \Longrightarrow P$ and $[\![P \wedge Q]\!] \Longrightarrow Q$
- $[\![P; Q]\!] \Longrightarrow P \wedge Q$

Derived rules: equality

- $[\![r = s; s = t]\!] \Longrightarrow r = t$
- $[\![s = t]\!] \Longrightarrow f(s) = f(t)$

Proof - “Apply Style”

lemma *name*: <goal>
apply <method>
apply <method>
...
done

Proof state:
1. $\bigwedge x_1 \ldots x_p . [\![A_1 ; \ldots ; A_m]\!] \Longrightarrow B$
2. $\bigwedge y_1 \ldots y_q . [\![C_1 ; \ldots ; C_n]\!] \Longrightarrow D$

$x_1, \ldots x_p$ are parameters, $A_1, \ldots A_m$ are assumptions, B is the subgoal.

Short Isabelle Example

Declarative Proofs

```
theorem "∃S. S ∉ range (f :: 'a ⇒ 'a set)"
proof
  let ?S = "{x. x ∉ f x}"
  show "?S ∉ range f"
  proof
    assume "?S ∈ range f"
    then obtain y where "?S = f y" ..
    show False
    proof cases
      assume "y ∈ ?S"
      hence "y ∉ f y"   by simp
      hence "y ∉ ?S"    by(simp add: `?S = f y`)
      with `y ∈ ?S` show False by contradiction
    next
      assume "y ∉ ?S"
      hence "y ∈ f y"   by simp
      hence "y ∈ ?S"    by(simp add: `?S = f y`)
      with `y ∉ ?S` show False by contradiction
    qed
  qed
qed
```

(From T. Nipkow “A Tutorial Introduction to Structured Isar Proofs”)

Application: Verifying a BGP Policy

We will apply Isabelle/HOL to formalize a correctness argument for a simple, but nontrivial traffic engineering BGP policy.

My goal: Clarify the reasoning used by network operators.

Not a goal: To provide a practical tool.

Isabelle/HOL's role: To increase confidence that I've formalized the reasoning correcly.

Border Gateway Protocol (BGP)

The Internet's interdomain routing protocol; It must:

- scale to the size of the Internet - link-state flooding is not feasible!
- give autonomous systems wide latitude - it cannot impose a single optimality condition on all systems.
- carry enough information to prevent forwarding loops.

BGP solves this through hierarchical routing and by allowing each AS to choose its locally optimal routes.

BGP: Rough Sketch

- A BGP advertisement pertains to an IP prefix and carries various attributes, including AS-level path and next hop IP address.
- Each network chooses a single “best” route to each IP prefix that it knows of.
- Among those best routes, some are advertised to neighbors.
- “best” is defined through *BGP policy*.

BGP Policy Knobs

- Filter input routes; reject “bogus” routes
- Rank routes for each IP prefix; choose where to send traffic
- Filter best routes before advertising; prevent others from sending you traffic
- Attach attributes to advertisements; communicate extra information about the route to neighbor.

Even though these knobs are simple, their effects depend on other network components, in particular, forwarding behavior.

Can we bridge the gap?

- What is the intended behavior of an autonomous system? We need:
    - A language with which describe the intended behavior.
- Can we prove that a policy achieves the intended behavior? We need:
    - A model of BGP and policy semantics.
    - A model of forwarding behavior.

We will work from a case study, and develop the tools needed to model and verify that example.

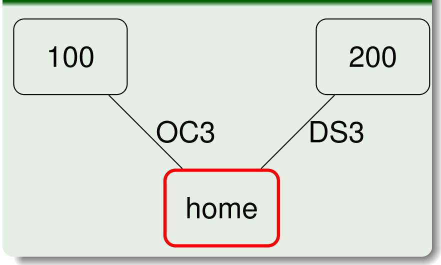
Case study - Informally

The case study appears in *BGP Design and Implementation* by Zhang and Bartell.

Objectives:

- Use OC3 primarily, DS3 as a backup
- Put some regular traffic on DS3 - we might as well use it since we have it. Which traffic?

Multi-homed network



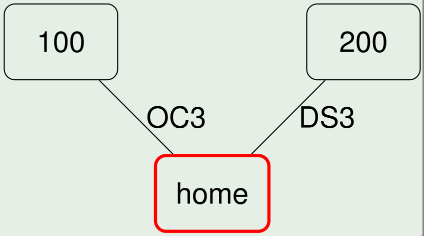
Case study - Informally

The case study appears in *BGP Design and Implementation* by Zhang and Bartell.

Objectives:

- Use OC3 primarily, DS3 as a backup
- Put some regular traffic on DS3 - we might as well use it since we have it. Which traffic?

Multi-homed network

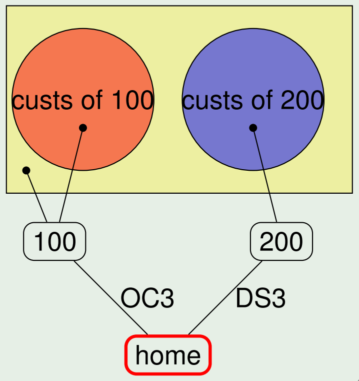


Case study - Informal Specification

Zhang & Bartell's solution:

- Request "default and customer" routes from providers.
- Use OC3 for traffic to non-customers of 100 and 200.
- Use OC3 for traffic to customers of 100
- Use DS3 for traffic to customers of 200.
- We take the above as our *informal specifications*. We will formalize this later.

Multi-homed network

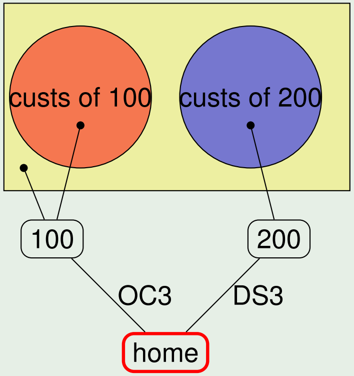


Case study - Informal Specification

Zhang & Bartell's solution:

- Request “default and customer” routes from providers.
- Use OC3 for traffic to non-customers of 100 and 200.
- Use OC3 for traffic to customers of 100
- Use DS3 for traffic to customers of 200.
- We take the above as our *informal specifications*. We will formalize this later.

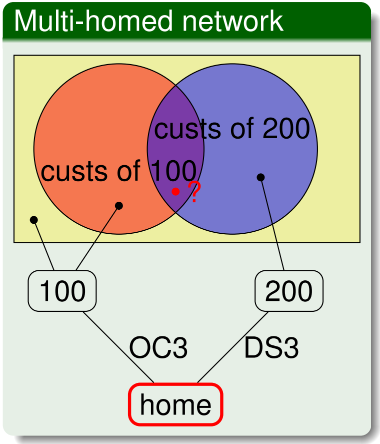
Multi-homed network



Case study - Informal Specification is Inconsistent

Zhang & Bartell's solution:

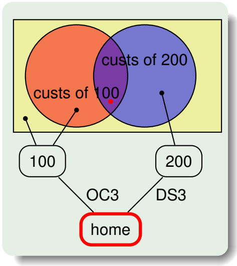
- Use OC3 for traffic to non-customers of 100 and 200.
- Use OC3 for traffic to customers of 100
- Use DS3 for traffic to customers of 200.
- What about customers of both? The above specifications are impossible to realize!



Case study - Informal Implementation

Zhang & Bartell's implementation:

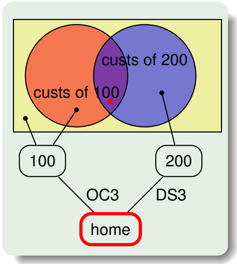
- Rank routes learned over the OC3 at preference level 120.
- Rank routes learned over the DS3 at preference level 100.
- This results in:
  - best route to default prefix is over OC3,
  - best route to customers of 100 is over OC3,
  - best route to customers of 200 (that are not custs of 100) is over DS3.



Case study - Informal Implementation

Zhang & Bartell's implementation:

- Rank routes learned over the OC3 at preference level 120.
- Rank routes learned over the DS3 at preference level 100.
- This results in:
  - best route to default prefix is over OC3,
  - best route to customers of 100 is over OC3,
  - best route to customers of 200 (that are not custs of 100) is over DS3.



But does this really accomplish the forwarding behavior we are trying to achieve? Are we sure? For this, a model will help...

Model

The model basically defines a map:

$$(\mathit{Topology}, \mathit{RoutingPolicy}, \mathit{Announcements}) \Rightarrow \mathit{ForwardingBehavior}$$

Model: Addresses, Prefixes, Routes

```
types address   = bool list
      prefix    = bool list
      asnumber = nat
      asseq     = asnumber list
      route     = prefix * asseq * link

definition Address :: address => bool
where Address a == (length a = 32)

fun addressInPrefix :: (address * prefix) => bool
where addressInPrefix (x,p) = prefixOf p x

definition AllPrefixes :: prefix set
where AllPrefixes == { p | p. length p <= 32}
```

Model: Topology & Policy

```
datatype link = OC3 | DS3

primrec linkID :: link => nat
where
linkID OC3 = 1 |
linkID DS3 = 2

primrec linkTo :: link => asnumber
where
linkTo OC3 = 100 |
linkTo DS3 = 200

fun rank :: route => nat
where
rank(s,p,OC3) = 120 |
rank(s,p,DS3) = 100
```

Model: Announcements

```
consts announce :: (asnumber * route) set

definition knows :: route set where
 knows == { r | asn r. announce (asn,r) }

definition mostSpecific :: address => prefix => bool
where
mostSpecific a p ==
    addressInPrefix(a,p) &
    (EX r. routePrefix r = p & knows r) &
    (ALL p' . (EX r'. routePrefix r'= p' &
                      addressInPrefix(a,p') &
                      knows r')
               --> p <$= p')
```

Model: axioms

```
axioms
ax_finite      : finite announce
ax_prefix_ord  : (  (P <= AllPrefixes) &
                    (P ~= {}) &
                    (ALL p:P . addressInPrefix (a,p))
                 )
                 ==> (EX p:P . ALL p':P . p <$= p')
ax_link        : announce(asn,(s,p,l))
                 ==> (linkTo l = asn)
ax_ann_pre     : announcedPrefixes <= AllPrefixes
```

Model: Comparing routes

```
fun better :: route => route => bool
where
better (s,p,l) (s',p',l') =
 (s=s') &
 ( (rank (s,p,l) > rank(s',p',l')) |
   (rank (s,p,l) = rank(s',p',l')
     & length p < length p') |
   (rank (s,p,l) = rank (s',p',l')
     & length p = length p'
     & comparePaths p p') |
   (rank (s,p,l) = rank (s',p',l')
     & length p = length p'
     & p = p'
     & linkID l < linkID l')
 )
```

Model: best routes & overall behavior

```
definition best :: route => bool
where
best r ==
 knows r &
 (let (s,p,l) = r
  in ALL p' l'. ((p ~= p'| l~=l') & knows(s,p',l'))
                --> better r (s,p',l'))

definition egress where
 egress a l == EX s p. mostSpecific a s & best (s,p,l)
```

Model: Concrete Examples

Announcements

Neighbor	Prefix	Path	Link
100	0.0.0.0/0	[100]	*oc3*
100	1.1.0.0/16	[100,500]	*oc3*
200	1.1.0.0/16	[200,400,500]	*ds3*
200	4.2.2.0/24	[200,800,900]	*ds3*

Knows, *Best*

Prefix	Path	Link	is Best?
0.0.0.0/0	[100]	*oc3*	yes
1.1.0.0/16	[100,500]	*oc3*	yes
1.1.0.0/16	[200,400,500]	*ds3*	no
4.2.2.0/24	[200,800,900]	*ds3*	yes

MatchingPrefixes, *MostSpecific*, *Egress*

Address a	$MatchingPrefixes(a)$	$MostSpecific(a)$	$Egress(a)$
4.2.2.103	$\{4.2.2.0/24, 0.0.0.0/0\}$	4.2.2.0/24	*ds3*
1.1.1.25	$\{1.1.0.0/16, 0.0.0.0/0\}$	1.1.0.0/16	*oc3*
170.2.3.100	$\{0.0.0.0/0\}$	0.0.0.0/0	*oc3*

Model: customers, neighbor behavior

```
consts cust      :: (address * asnumber) set

definition defaultAdvertised where
defaultAdvertised asn ==
 EX p l. announce(asn, (defaultPrefix,p,l))

definition custAdvertised where
custAdvertised asn ==
 ALL a.
 cust(a,asn) = (EX s p l. announce(asn,(s,p,l)) &
               addressInPrefix(a,s) &
               s <$ defaultPrefix)

definition defaultAndCust where
defaultAndCust asn == custAdvertised asn &
                      defaultAdvertised asn
```

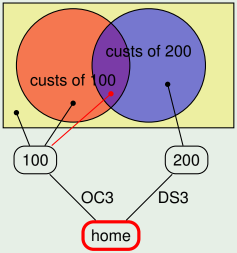
Formal Specification

Specification

1. $[Cust(a, 100) \wedge Cust(a, 200)] \rightarrow Egress(a, oc3)$
2. $[\neg Cust(a, 100) \wedge Cust(a, 200)] \rightarrow Egress(a, ds3)$
3. $\neg Cust(a, 200) \rightarrow Egress(a, oc3)$

$Cust(a, 100)$	$Cust(a, 200)$	$Egress(a)$
yes	yes	*oc3*
yes	no	*oc3*
no	yes	*ds3*
no	no	*oc3*

Multi-homed network



Verification

We can prove some parts of the specification, for example:

Lemma

$Cust(a, asn) \Longrightarrow \exists l\,.\,(Egress(a, l) \wedge Cust(a, linkTo(l)))$

Theorem

$(Cust(a, 200) \wedge \neg Cust(a, 100)) \Longrightarrow Egress(a, ds3)$

Proof.

1. Assume a is an address such that $Cust(a, 200)$ and not $Cust(a, 100)$.
2. There is a link l such that $Egress(a, l)$ and $Cust(a, linkTo(l))$ (by above).
3. Either $l = oc3$ or $l = ds3$.
4. If $l = oc3$, then $linkTo(l) = 100$, hence $Cust(a, 100)$, $\perp$.
5. Thus, $l = ds3$. □

Verification

We can prove some parts of the specification, for example:

Lemma

$Cust(a, asn) \Longrightarrow \exists l\,.\,(Egress(a, l) \wedge Cust(a, linkTo(l)))$

Theorem

$(Cust(a, 200) \wedge \neg Cust(a, 100)) \Longrightarrow Egress(a, ds3)$

Proof.

1. Assume a is an address such that $Cust(a, 200)$ and not $Cust(a, 100)$.
2. There is a link l such that $Egress(a, l)$ and $Cust(a, linkTo(l))$ (by above).
3. Either $l = oc3$ or $l = ds3$.
4. If $l = oc3$, then $linkTo(l) = 100$, hence $Cust(a, 100)$, $\perp$.
5. Thus, $l = ds3$.

□

Verification

We can prove some parts of the specification, for example:

Lemma

$Cust(a, asn) \Longrightarrow \exists l\,.\,(Egress(a, l) \wedge Cust(a, linkTo(l)))$

Theorem

$(Cust(a, 200) \wedge \neg Cust(a, 100)) \Longrightarrow Egress(a, ds3)$

Proof.

1. Assume a is an address such that $Cust(a, 200)$ and not $Cust(a, 100)$.
2. There is a link l such that $Egress(a, l)$ and $Cust(a, linkTo(l))$ (by above).
3. Either $l = oc3$ or $l = ds3$.
4. If $l = oc3$, then $linkTo(l) = 100$, hence $Cust(a, 100)$, $\perp$.
5. Thus, $l = ds3$.

$\square$

Verification

We can prove some parts of the specification, for example:

Lemma

$$Cust(a, asn) \Longrightarrow \exists l\,.\,(Egress(a, l) \land Cust(a, linkTo(l)))$$

Theorem

$$(Cust(a, 200) \land \neg Cust(a, 100)) \Longrightarrow Egress(a, ds3)$$

Proof.

1. Assume a is an address such that $Cust(a, 200)$ and not $Cust(a, 100)$.
2. There is a link l such that $Egress(a, l)$ and $Cust(a, linkTo(l))$ (by above).
3. Either $l = oc3$ or $l = ds3$.
4. If $l = oc3$, then $linkTo(l) = 100$, hence $Cust(a, 100)$, $\perp$.
5. Thus, $l = ds3$.

□

Verification

We can prove some parts of the specification, for example:

Lemma

$$Cust(a, asn) \implies \exists l\,.\,(Egress(a, l) \wedge Cust(a, linkTo(l)))$$

Theorem

$$(Cust(a, 200) \wedge \neg Cust(a, 100)) \implies Egress(a, ds3)$$

Proof.

1. Assume a is an address such that $Cust(a, 200)$ and not $Cust(a, 100)$.
2. There is a link l such that $Egress(a, l)$ and $Cust(a, linkTo(l))$ (by above).
3. Either $l = oc3$ or $l = ds3$.
4. If $l = oc3$, then $linkTo(l) = 100$, hence $Cust(a, 100)$, $\perp$.
5. Thus, $l = ds3$.

$\square$

Verification

We can prove some parts of the specification, for example:

Lemma

$Cust(a, asn) \implies \exists l\,.\,(Egress(a, l) \wedge Cust(a, linkTo(l)))$

Theorem

$(Cust(a, 200) \wedge \neg Cust(a, 100)) \implies Egress(a, ds3)$

Proof.

1. Assume a is an address such that $Cust(a, 200)$ and not $Cust(a, 100)$.
2. There is a link l such that $Egress(a, l)$ and $Cust(a, linkTo(l))$ (by above).
3. Either $l = oc3$ or $l = ds3$.
4. If $l = oc3$, then $linkTo(l) = 100$, hence $Cust(a, 100)$, $\perp$.
5. Thus, $l = ds3$.

$\square$

Verification... Fails!

On the other hand, some parts fail! In particular, this fails:

$$[Cust(a, 100) \wedge Cust(a, 200)] \to Egress(a, oc3)$$

Here is a counterexample:

Announcements

Neighbor	Prefix	Path	Link	is best?
100	0.0.0.0/0	[100]	*oc3*	yes
100	1.0.0.0/8	[100,300]	*oc3*	yes
200	0.0.0.0/0	[200]	*ds3*	no
200	1.0.0.0/16	[200,300]	*ds3*	yes

For address $a = 1.0.0.0$:

- $MatchingPrefixes(a) = \{1.0.0.0/16, 1.0.0.0/8, 0.0.0.0/0\}$
- $Cust(a, 100)$, $Cust(a, 200)$
- $MostSpecific(a) = 1.0.0.0/16$
- $Egress(a) = ds3$

Verification... Fails!

On the other hand, some parts fail! In particular, this fails:

$$[Cust(a, 100) \wedge Cust(a, 200)] \rightarrow Egress(a, oc3)$$

Here is a counterexample:

Announcements

Neighbor	Prefix	Path	Link	is best?
100	0.0.0.0/0	[100]	*oc3*	yes
100	1.0.0.0/8	[100,300]	*oc3*	yes
200	0.0.0.0/0	[200]	*ds3*	no
200	1.0.0.0/16	[200,300]	*ds3*	yes

For address $a = 1.0.0.0$:

- $MatchingPrefixes(a) = \{1.0.0.0/16, 1.0.0.0/8, 0.0.0.0/0\}$
- $Cust(a, 100)$, $Cust(a, 200)$
- $MostSpecific(a) = 1.0.0.0/16$
- $Egress(a) = ds3$

Verification

Two solutions:

- Change the specification so that it makes no requirement on the case $\mathit{Cust}(a, 100) \wedge \mathit{Cust}(a, 200)$;
- Add an assumption; for example the assumption that both providers will announce their common customers' prefixes identically.

Either option allows us to verify that the policy meets the specification.

Conclusions

We see that:

- there is a gap between operator intent and BGP policy, and
- that gap is nontrivial, even when using a toy model of BGP.

Questions:

- Are there better ways to characterize operator intentions? In this example, maybe we should have expressed the real intent as “traffic balance”.

Cryptographic protocols

A *cryptographic protocol* lets agents communicate securely in insecure networks.

In Chapter 10 of Isabelle/HOL tutorial, L. Paulson analyzes several cryptographic protocols using Isabelle/HOL and verifies numerous secrecy and authenticity properties.

This is a good example, because the correctness of the protocol is important, and the protocol will likely be used for a long time, making the verification effort worthwhile.

It is also a nice illustration of verification techniques, in particular inductively defined sets and induction.

References & further reading

1. Isabelle
    1. http://isabelle.in.tum.de
    2. Tutorial on Isabelle/HOL
    3. Tutorial on Isar
    4. C. Ballarin and G. Klein, http://isabelle.in.tum.de/coursematerial/IJCAR04/index.html
2. Background on HOL, theorem provers
    1. W. Farmer, “Seven Virtues of Simple Type Theory”
    2. F. Wiedijk, “Formal Proof - Getting Started”
3. Verification
    1. L. Paulson, “The Inductive Approach to Verifying Cryptographic Protocols”
    2. A. Biltcliffe et al, “Rigorous Protocol Design in Practice: An Optical Packet-Switch MAC in HOL”.