Prototype Verification System

Maria Schett
703641 Interaktives Beweisen

Overview

- Facts
- Applications
- Telephone Book
- Subset Types
- Type Checking
- Abstract Data Types
- Dependent Types
- Propositional Proof
- Impressions

Facts

- Implemented in Common Lisp
  - Functions in C, Tcl/TK and LaTex
  - Interface: GNU Emacs
- Prototype Verification System
  - specification language: strongly typed & based on classical higher-order logic
  - interactive theorem prover (or proof checker)
- current version: PVS 6.0 (start 1993)
  - open-source & GPL

Facts

- developed by SRI
  - Jovial Verification System, Hierarchial Development Model, STP, EHDM
  - Yices

- current work: develop methodologies for
  - highly automated hardware verification
  - integration with model checkers
  - for concurrent and real-time systems

Applications

"PVS has been installed at hundreds of sites in North America, Europe and Asia" [PVS]

- DB with reference to history, technology and applications of PVS: ~300 citations

- **NASA PVS Library**
  - "The major goals of our research program are to advance the state-of-the-art in formal methods, making it practical for use on life-critical systems developed by the aerospace industry in the United States." [LFM]

Motivation

- **specification**: phone book [TUT]
  - A phone book shall store the phone numbers of a city.
  - It shall be possible to retrieve a phone number, given a name.
  - It shall be possible to add and delete entries from a phone book.

`FindPhone AddPhone DelPhone`

Types

- represent entities as distinguishable types

```
N: TYPE     % names
P: TYPE     % phone number
```

- uninterpreted types
  - distinguishable, nothing about members is known & equality predicate

```
B: TYPE = [N -> P]    % phone book
```

  - total function?

Types

```
B: TYPE = [N -> P]    % phone book
```

- For names without phone number introduce “n0”

```
n0: P
emptybook: B
emptyax: AXIOM
  FORALL (nm:N):emptybook(nm)= n0
```

FindPhone & AddPhone

- find a phone number

```
FindPhone: [B,N -> P]
Findax: AXIOM FORALL (bk:B), (nm:N):
  FindPhone(bk,nm) = bk(nm)
```

- add a phone number

```
AddPhone: [B,N,P -> B]
Addax: AXIOM FORALL (bk:B), (nm:N), (pn:P):
  AddPhone(bk,nm,pn) = bk WITH [(nm):=pn]
```

Challenge

- testing vs. challenging

```
FindAdd: CONJECTURE
FORALL(bk:B),(nm:N),(pn:P):
  FindPhone(AddPhone(bk,nm,pn),nm) = pn
```

- Prove!

Short Reminder

```
phone-1 : THEORY

N: TYPE               % names
P: TYPE               % phone number
B: TYPE = [N -> P]    % phone book

n0: P
emptybook: B
emptyax: AXIOM FORALL (nm:N):emptybook(nm)= n0

FindPhone: [B,N -> P]
Findax: AXIOM FORALL (bk:B), (nm:N): FindPhone(bk,nm) = bk(nm)

AddPhone: [B,N,P -> B]
Addax: AXIOM FORALL (bk:B), (nm:N), (pn:P):
  AddPhone(bk,nm,pn) = bk WITH [(nm):=pn]

FindAdd: CONJECTURE
FORALL(bk:B),(nm:N),(pn:P): FindPhone(AddPhone(bk,nm,pn),nm) = pn
```

Prove FindAdd Conjecture

- start prover

```
FindAdd :
|------
{1} FORALL(bk:B),(nm:N),(pn:P):
  FindPhone(AddPhone(bk,nm,pn),nm) = pn
Rule?
```

- type

```
(grind :theories(“phone-1”))
```

Grind?

- ~ 20 basic commands &
- ~ 20 higher-level commands (strategies)
  - `assert, inst?, induct-and-simplify, induct-and-rewrite, skosimp*`
- highest level: `grind [args]`
  - skolemization
  - heuristic instantiation
  - propositional simplification (BDD)
  - if-lifting
  - rewriting
  - decision procedure for linear arithmetic & equality

Output

```
Rule? (grind :theories(phone-1))
Addax rewrites AddPhone(bk, nm, pn)
   to bk WITH [(nm) := pn]
Findax rewrites
  FindPhone(bk WITH [(nm) := pn], nm)
  to pn
…
Q.E.D. ☺
```

DelPhone

```
DelPhone: [B,N -> B]
Delax: AXIOM FORALL (bk:B), (nm:N):
  DelPhone(bk,nm)=bk WITH [(nm) := n0]
```

- challenge

```
DelAdd: CONJECTURE
  FORALL (bk:B), (nm:N), (pn:P)
  DelPhone(AddPhone(bk,nm,pn),nm) = bk
```

Prove DelAdd Conjecture

```
|----
FORALL (bk:B), (nm:N), (pn:P):
DelPhone(AddPhone(bk,nm,pn),nm) = bk
Rule? (grind :theories(phone-1))
…
|----
{1} bk!1  WITH[(nm!1) := pn!1]
          WITH[(nm!1) := n0] = bk!1
Rule?
```

- **bk!1 …** representatives for quantified variables

Prove DelAdd Conjecture

```
|----
{1} bk!1  WITH[(nm!1) := pn!1]
          WITH[(nm!1) := n0] = bk!1
Rule? (apply-extensionality)
```

- to prove that bk!1 = bk!1 (functions are the same)

```
...
|----
{1} bk!1  WITH[(nm!1) := pn!1]
          WITH[(nm!1) := n0](x!1)
  = bk!1(x!1)
```

Prove DelAdd Conjecture

```
...
Rule? (lift-if)
...
Rule? (ground)
{-1} nm!1 = (x!1)
|----
{1} n0 = bk!1(x!1)
```

- We have to show, that in the original phone book, the phone number was n0.

Observation

- PVS is designed to be the “rigorous skeptic”
  - detect errors in reasoning (or specification?), but also
  - helps you to find the error by providing feedback

(Subset) Types

- Base types: **`bool, int, real`**
- Function types: **`[bool,int -> int]`**
- Subset types: `nat: TYPE = {i:int | i >= 0}`

- `/ : [int, { n: int | n /= 0 } -> int]`

- `average = sum / numbers : int`

- only well-typed if `numbers` is not 0
- type checking conditions (TCCs)

Type Checking

- type checking conditions (TCCs)

- type checking in PVS is undecidable
- proof obligations
- consistency check

- most obligations can be discharged automatically

Recursion

- must be shown to terminate

```
factorial(x:nat): RECURSIVE nat =
  IF x = 0 THEN 1
  ELSE x * factorial (x-1) ENDIF
  MEASURE (LAMBDA (x:nat):x)
```

- generates proof obligation

```
Factorial_TCC2: OBLIGATION
(FORALL (x:nat): NOT x = 0 IMPLIES x-1 < x)
```

Abstract Data Types

- automatic generation of complete axiomatization

```
stack [t: TYPE]: DATATYPE
BEGIN
  empty: emptystack?
  push(top: t, pop: stack) : nonemptystack?
END stack
```

empty, push: constructors,

top, pop: accessors,

emptystack?, nonemptystack: recognizers

Abstract Data Types

- automatic generation of complete axiomatization:

- extensionality axioms for constructors
- eta axiom
- accessor/constructor axioms
- induction scheme
- functions distributing predicates over the stack base type
- subterm function
- well-foundedness axiom
- recursive combinator

Dependent Types

```
date: TYPE = [ yr: year, mon: month,
  {d: nat | d <= days(mon, yr)} ]
```

- for function arguments

```
ratio(x, y: real, z: {z: real | z /= x}:real
  = (x - y) / (x - z)
```

- generates TCCs

Propositional Proof

```
|--------
{1} P => Q
```

Rule: `flatten`

```
[-1] P
|--------
{1} Q
```

```
|--------
{1} P OR Q
```

```
|--------
{1} P
{2} Q
```

```
[-1] P & Q
|-------
```

```
[-1] P
[-2] Q
|--------
```

Propositional Proof

```
[-1] P OR Q
|--------
```

Rule: `split`

```
[-1] P          [-1] Q
|-------        |--------
```

Comparison

[17P]	HOL	Mizar	PVS
small proof kernel	+	-	-
extensible/programmable	+	-	+
powerful automation	+	-	+
readable	-	+	-
constructive logic	-	-	-
decidable types	+	+	-
dependent types	-	+	+
based on HOL	+	-	+
large mathematical stnd. lib.	+	+	+

My Impressions

- overwhelming
- good documentation available
  - especially for using the system
- on-going development since <1993

Thank you for your attention!

Bibliography

- [LFM]: NASA Langley Formal Methods Site
  - http://shemesh.larc.nasa.gov/fm/index.html

[PVS]: PVS Specification and Verification System
  - http://pvs.csl.sri.com/index.shtml

[TUT]: A Tutorial Introduction to PVS, Boca Raton, ‘95
  - available at [PVS]

[17P]: The Seventeen Provers of the World,
Freek Wiedijk, ‘06