Propositional Translation of PVS Specifications

Carsten Sinz

Wilhelm-Schickard-Institute
University of Tübingen, Germany

09/03/04

Overview

1. Motivation
2. Challenges: finiteness, propositional encoding
3. Solution: translation method
4. Application: SPIDER IC protocol

Motivation for Propositional Translation

- Helps generating PVS proofs. Automatic checking of finite models allows to
  - debug PVS specifications
  - avoid futile proof attempts
- Delivers decision procedure for finite theories by enabling the use of SAT solvers
- Extends scope of (bounded) model checking beyond (finite state) transition systems.

Part I: Challenges, Notions and First Thoughts

1st Challenge: Finiteness

Finiteness is a necessary precondition for propositional translation. This includes:

1. Finite base domains: no infinite base types (e.g. `real`)
2. Finite computations: no recursion of unlimited depth
3. Finite data types: no recursive data types (as used, e.g., for lists, trees)

How to Achieve Finiteness I: Base Domains

Replace infinite types by finite counterparts, use e.g.

- Value restrictions, e.g. initial segment $\{0,\ldots,k\}$ of integers instead of $\mathbb{N}$ (note: overflow has to be considered)
- Abstractions, e.g. replace $\mathbb{R}$ by finite set of intervals $\{(-\infty,a_1),\ldots,[a_j,a_{j+1}),\ldots,[a_k,\infty)\}$, represented as a finite set $\{P_i \mid 0 \leq i \leq k\}$ of Boolean predicates
- Finite structure with same properties, e.g. finite field $\mathbb{F}_p$ instead of infinite number field

Finiteness II: Recursion (Example)

```
% 'standard version' of factorial function
fac(x: nat): RECURSIVE nat =
  IF x <= 1 THEN 1 ELSE x * fac(x-1) ENDIF
  MEASURE x

% with limited recursion depth
fac_lr(x: nat, lim: below(N)): RECURSIVE lift[nat] =
  IF x <= 1 THEN up(1)
  ELSE  IF lim = 0 THEN bottom
        ELSE LET rec = fac_lr(x-1, lim-1)
             IN IF bottom?(rec) THEN bottom
                ELSE up(x * down(rec))
  ENDIF ENDIF   ENDIF
  MEASURE lim
```

Finiteness II: Recursion

Methods to achieve finite computation sequences:

- Limit recursion depth as shown in previous example
- Pre-compute function result by (partial) evaluation
- Known sufficient upper bound for number of nested recursion steps (a priori), e.g. due to `measure`-annotation

Finiteness III: Recursive Data Types

User has to provide finite replacement, e.g.:

```
% finite replacement of prelude's 'list' data type
finite_list[T: TYPE+, N: nat]: THEORY
BEGIN
  finite_list: TYPE = [# len:upto(N), lmns:[below(N)->T] #]
  null: finite_list = (# len := 0, lmns :=
      (LAMBDA (x: below(N)): epsilon!(t: T): TRUE) #)
  cons(e: T, l: finite_list): finite_list =
    IF l`len = N THEN l
    ELSE (# len:=l`len+1, lmns:=l`lmns WITH [(l`len):=e] #)
    ENDIF
END finite_list
```

2^{nd} Challenge: Translation

Assume finiteness preconditions fulfilled. How can we translate a PVS specifiction to propositional logic?

Different methods conceivable:

1. Finite automata for a restricted class of PVS specifications (cf. PVS commands `abstract` and `model-check`)

2. Ground evaluation of explicitly defined functions

3. Propositional encoding of arbitrary functions (including implicitly defined functions)

Excursus: Implicit vs. Explicit Definition

Definition of a predicate $P(\vec{x})$:

Implicit: By a set S of formulae, such that predicate P has the desired properties in all models of S.

Explicit: By giving a defining sentence for P, i.e. a sentence of the form $(\forall \vec{x})(P(\vec{x}) \Leftrightarrow \phi_P(\vec{x}))$ for some formula ϕ_P.

Similar for function definitions.

Rough analogy: implicit $\approx$ axiomatic method, explicit $\approx$ functional programming language

(Note: cf. Beth's definability theorem)

Translation I: Ground Evaluation

1. Expand function definitions: Given: $f(\vec{x}) := \phi(\vec{x})$, $g(\vec{y}) := \psi(f(\vec{y}))$, replace g by $g(\vec{y}) = \psi(\phi(\vec{y}))$

2. Expand quantifiers: Assume $D = \{d_1, \ldots, d_k\}$.

$$\begin{array}{lcl} (\forall x \in D)\phi(x) & \dashrightarrow & \phi(d_1) \wedge \cdots \wedge \phi(d_k) \\ (\exists x \in D)\phi(x) & \dashrightarrow & \phi(d_1) \vee \cdots \vee \phi(d_k) \end{array}$$

If all function definitions are explicit, this process results in variable-free (ground) terms.

Moreover, if all occurrences of f are replaced, the definition of f may be discarded.

Translation I: Ground Evaluation (cont'd)

(+) Fully automatic term rewriting, no user interaction

(−) Works only for explicitly defined functions

(−) Expansion may blow up memory space

Translation II: Propositional Encoding

Given set $A = \{A_i \mid i \leq k\}$ of finite types $A_i = \{0, \ldots, \alpha_i - 1\}$.

Encode functions as sets of characterizing predicates:

$$f : A_0 \times \cdots \times A_{k-1} \to A_k \quad \dashrightarrow \quad \{P^f_{x_0,\ldots,x_k} \mid 0 \leq x_i < \alpha_i\}$$

Intention: $P^f_{x_0,\ldots,x_k}$ holds iff $f(x_0, \ldots, x_{k-1}) = x_k$.

Generates $\prod_{i=0}^{k} \alpha_i$ predicates characterizing function f.

Additional sentence required to express functionality of f:

$$\bigwedge_{\substack{0 \leq i < k \\ x_i \in A_i}} \left(\bigwedge_{\substack{y_0, y_1 \in A_k \\ y_0 \neq y_1}} \left(P^f_{x_0,\ldots,x_{k-1},y_0} \Rightarrow \neg P^f_{x_0,\ldots,x_{k-1},y_1}\right) \wedge \bigvee_{y \in A_i} P^f_{x_0,\ldots,x_{k-1},y} \right)$$

Propositional Encoding: Example

Addition on $\mathbb{F}_3 = \{0, 1, 2\}$:

$$+ : \mathbb{F}_3 \times \mathbb{F}_3 \longrightarrow \mathbb{F}_3 \quad \dashrightarrow \quad \underbrace{\{P^+_{0,0,0}, \ldots, P^+_{2,2,2}\}}_{\text{27 predicates}}$$

Properties of $+$:

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

(or axiomatic)

Propositional encoding:

$P^+_{0,0,0} \wedge P^+_{0,1,1} \wedge P^+_{0,2,2} \wedge P^+_{1,0,1} \wedge \cdots \wedge P^+_{2,2,1}$
with additional (functionality) restriction

$$\bigwedge_{\substack{a,b,x,y \in \{0,1,2\} \\ x \neq y}} (P^+_{a,b,x} \Rightarrow \neg P^+_{a,b,y})$$

Prop. Encoding: Higher Order Functions (I)

New: Functions as arguments, e.g. $f : (A \to B) \to C$.

For an encoding of higher-order function f we need:

1. An enumeration $g_0, g_1, \ldots$ of all possible arguments of f to build the set of predicates $P^f_{x,y}$.

2. A bijection between enumerated functions (g_i) and functions described by a set of predicates $(\{P^h_{x,y}\})$.

Prop. Encoding: Higher Order Functions (II)

Enumeration of function types: For $A = \{0, \ldots, \alpha - 1\}$, $B = \{0, \ldots, \beta - 1\}$ we enumerate elements of $(A \to B) = B^A$ as $\{f_0, \ldots, f_{\beta^\alpha - 1}\}$ where each f_i is defined by

$$\sum_{j=0}^{\alpha-1} f_i(j) \cdot \beta^j = i \ .$$

For $0 \leq i < \beta^\alpha$ and $0 \leq j < \alpha$ we thus have

$$\begin{aligned} f_i(j) &= \text{"}j\text{-th digit in the } \beta\text{-adic expansion of } i\text{"} \\ &= \lfloor i/\beta^j \rfloor \bmod \beta \end{aligned}$$

Prop. Enc.: Higher Order Functions (III)

Assume $f : A \longrightarrow B$ with A, B as above.

f can be described

(i) by a set of predicates $\{P^f_{x,y} \mid x < \alpha, y < \beta\}$ or

(ii) as an enumeration index i with $0 \leq i < \beta^\alpha$.

To identify equivalent functions with different representations we use the equivalence predicate $\epsilon(f, i)$:

$$\epsilon(f, i) = \bigwedge_{0 \leq j < \alpha} P^f_{j, \lfloor i/\beta^j \rfloor \bmod \beta}$$

Higher Order Functions: Example

Example: app $: (A \rightarrow B) \times A \rightarrow B$
with $A = \{0, 1\}$, $B = \{0, 1, 2\}$, i.e. $\alpha = 2, \beta = 3$

Domain size of type $(A \rightarrow B)$: $\beta^\alpha = 3^2 = 9$

Predicates to encode function app: $\beta^\alpha \alpha \beta = 54$

Example predicates for function app:

$P^{\mathsf{app}}_{7,1,2}$: function app applied to $\{0 \mapsto 1, 1 \mapsto 2\}$ (as $7 = (2, 1)_3$) and 1 yields 2

$P^{\mathsf{app}}_{2,0,1}$: function app applied to $\{0 \mapsto 2, 1 \mapsto 0\}$ (as $2 = (0, 2)_3$) and 0 yields 1

Part II: Translation Method

Method Outline

Step 1: Translate PVS specification into a finite domain version (manually)

Step 2: Rewrite theory using explicit function definitions (definitions may then be discarded)

Step 3: Perform propositional encoding:

A. Decomposition of nested functions

B. Expansion of quantifiers

C. Replacement of atomic formulas

Step 3A: Decomposition

Decomposition of nested functions:

$$f(g(x)) = y \quad \dashrightarrow \quad (\exists u)\big(f(u) = y \land g(x) = u\big)$$

Encoding of higher-order functions' arguments:

(i_g is a new index variable representing g's enumeration index)

$$f(g) = x \quad \dashrightarrow \quad f(i_g) = x$$

Decomposition of complex equations:

$$f(x) = g(y) \quad \dashrightarrow \quad (\exists u)\big(f(x) = u \land g(y) = u\big)$$

Step 3A: Decomposition (cont'd)

Decomp. of partially evaluated functions:

(i_h is a new variable representing $g(x)$'s enumeration index)

$$f(g(x)) = y \quad \dashrightarrow \quad (\exists h)\big(f(i_h) = y \wedge g(x) = i_h\big)$$

where $f : (A \to B) \to C$,
$g : D \to (A \to B)$ and
$h : (A \to B)$.

Result of decomposition:

Atomic formulae of the form $f(x) = y$, where x and y are variables (besides atoms of Boolean type).

Step 3B: Expansion of Quantifiers

Quantification over (non-fctl.) base type $A = \{0, \ldots, \alpha - 1\}$:

$$(\forall x : A)\phi(x) \quad \dashrightarrow \quad \bigwedge_{0 \le i < \alpha} \phi(i)$$

Quantification over function type (with $B = \{0, \ldots, \beta - 1\}$):

(results in QBF formula; $\hat{f}$ is new function symbol with f's type)

$$(\forall f : A \to B)\phi(f, i_f) \quad \dashrightarrow \quad \forall \{P^{\hat{f}}_{j,k} \mid j < \alpha, k < \beta\} \bigwedge_{0 \le i < \beta^\alpha} \big(\epsilon(\hat{f}, i) \Rightarrow \phi(\hat{f}, i)\big)$$

Step 3B: Expansion of Quantifiers (cont'd)

Simpler transformations possible if f does not simultaneously occur in both representations in ϕ:

$$(\forall f : A \to B)\phi(f) \quad \dashrightarrow \quad \forall\{P^{\hat{f}}_{j,k} \mid j < \alpha, k < \beta\}\ \phi(\hat{f})$$

(no enumeration index representation; $\hat{f}$ new function symbol)

$$(\forall f : A \to B)\phi(i_f) \quad \dashrightarrow \quad \bigwedge_{0 \le i < \beta^\alpha} \phi(i)$$

(only enumeration index representation)

Step 3C: Replacement of Atomic Formulas

Replace atoms by propositional variants:

$$f(i) = j \quad \dashrightarrow \quad P^f_{i,j}$$

Result of transformation:

Purely propostional, equivalent formula

Example for Higher-Order Transformation

Input: $app(f, x) = f(x)$

(where $app: (A \to B) \times A \to B,\ f: A \to B, A = \{0, 1\}, B = \{0, 1, 2\}$)

Transformation:

$$
\begin{array}{ll}
 & (\forall f, x)\big(app(f, x) = f(x)\big) \\
\dashrightarrow & (\forall f, x)\big(app(i_f, x) = f(x)\big) \\
\dashrightarrow & (\forall f, x)(\exists y)\big(app(i_f, x) = y \wedge f(x) = y\big) \\
\dashrightarrow & (\forall f) \bigwedge_{j \in \{0,1\}} \bigvee_{k \in \{0,1,2\}} \big(app(i_f, j) = k \wedge f(j) = k\big)
\end{array}
$$

Transformation Example (cont'd)

$$(\forall f) \bigwedge_{j \in \{0,1\}} \bigvee_{k \in \{0,1,2\}} \big(app(i_f, j) = k \wedge f(j) = k\big)$$

$$\dashrightarrow \quad \forall\{P^{\hat{f}}_{s,t} \mid s < 2, t < 3\} \bigwedge_{0 \le i < 9} \Bigg(\epsilon(\hat{f}, i) \Rightarrow$$

$$\bigwedge_{j \in \{0,1\}} \bigvee_{k \in \{0,1,2\}} \big(app(i, j) = k \wedge \hat{f}(j) = k\big)\Bigg)$$

$$\equiv \quad \forall\{P^{\hat{f}}_{s,t}\} \bigwedge_{0 \le i < 9} \Bigg(\bigwedge_{0 \le j < 2} P^{\hat{f}}_{j, \lfloor i/3^j \rfloor \bmod 3} \Rightarrow \bigwedge_{j \in \{0,1\}} \bigvee_{k \in \{0,1,2\}} \big(P^{app}_{i,j,k} \wedge P^{\hat{f}}_{j,k}\big)\Bigg)$$

Note: Transformation can be avoided when use of $app(f, x) = f(x)$ as a rewrite rule removes all occurrences of app.

“Global” Picture

Initial problem: Prove, that ϕ holds under assumption T, i.e.

$$T \models \phi \quad \text{where } T = \{\psi_1, \ldots, \psi_n\}$$

T: assumptions, theory (PVS axioms, function definitions)

ϕ: proof goal (PVS lemma, conjecture)

Deduction theorem: $T \models \phi$ iff $\models T \Rightarrow \phi$

iff $\models \psi_1 \wedge \cdots \wedge \psi_n \Rightarrow \phi$

Propositional translation: $\models \psi_1^* \wedge \cdots \wedge \psi_n^* \Rightarrow \phi^*$

Add functionality restriction P: $\models P \wedge \psi_1^* \wedge \cdots \wedge \psi_n^* \Rightarrow \phi^*$

SAT-solver checks consistency, thus negate:

Show unsatisfiability of $P \wedge \psi_1^* \wedge \cdots \wedge \psi_n^* \wedge \neg\phi^*$

Part III: Spider IC Protocol Translation

Spider IC Protocol

Finiteness: number of RMUs, BIUs and messages unlimited

Recursive functions and data types: not used

Unhandy: implicit function definitions (`choose`/`epsilon`/`card`), as used by IC's majority computation

Solutions:

- Fix number of RMUs, BIUs and messages (theory parameters)
- Provide explicit definitions for `card` and `majority`

Outline of Transformation Proceeding

1. Instantiate SPIDER IC theory parameters to obtain finite types
2. Rewrite function definitions (of main lemma and dependent functions)
3. Convert remaining functions and lemma to propositional logic

Interactive Consistency Protocol: Validity

Validity lemma after skolemization:

```
{-1}  src_msgs!1 = send(encode(sent!1), src_status!1)
{-2}  src_filter!1 = msg_filter(src_msgs!1)
{-3}  src_filtered_eligible!1 =
       conforming_eligible(source_eligible_restrict(src_eligible!1, s!1),
                           src_filter!1)
{-4}  rly_msgs!1 =
       send(vote(src_filtered_eligible!1, src_msgs!1), rly_status!1)
{-5}  rly_filter!1 = msg_filter(valid_or_source_error, rly_msgs!1)
{-6}  rly_filtered_eligible!1 =
       conforming_eligible(rly_eligible!1, rly_filter!1)
{-7}  good(src_status!1)(s!1)
{-8}  source_eligible(src_eligible!1, s!1)
{-9}  hybrid_majority_good?(rly_status!1, rly_eligible!1)
  |-------
{1}   vote(rly_filtered_eligible!1, rly_msgs!1)(d!1) =
       valid_encode[S, T](sent!1(s!1))
```

IC Protocol: Validity (II)

With definitions (e.g. `src_msgs!1`) expanded:

```
[-1]  trustworthy?(src_status!1(s!1)) OR recovering?(src_status!1(s!1))
[-2]  FORALL (d: below(R)): src_eligible!1(d)(s!1)
[-3]  FORALL (d: below(D)):
        card(x: below(R) | rly_eligible!1(d)(x) AND
          (trustworthy?(rly_status!1(x)) OR recovering?(rly_status!1(x)))) >
        card(x: below(R) | rly_eligible!1(d)(x) AND asymmetric(rly_status!1)(x)) +
        card(x: below(R) | rly_eligible!1(d)(x) AND symmetric(rly_status!1)(x))
  |-------
{1}   vote(conforming_eligible(rly_eligible!1, msg_filter(valid_or_source_error,
        send(vote(conforming_eligible(source_eligible_restrict(src_eligible!1, s!1),
                                      msg_filter(send(encode(sent!1),src_status!1))),
                  send(encode(sent!1), src_status!1)), rly_status!1))),
        send(vote(conforming_eligible(source_eligible_restrict(src_eligible!1, s!1),
                                      msg_filter(send(encode(sent!1), src_status!1))),
                  send(encode(sent!1), src_status!1)), rly_status!1))(d!1)
       = valid(sent!1(s!1))
```

IC Protocol: Validity (III)

Nested functions' definitions expanded:

```
(1)  vote(f, m)(d) = IF (EXISTS (x: below(S)): f(d)(x))
                     THEN majority(f(d), m(d)) ELSE source_error ENDIF
(2)  conforming_eligible(e,f)(d)(x) = ({x: below[R] | f(d)(x) AND e(d)(x)})
(3)  msg_filter(m)(d)(s) = valid?(m(d)(s))
(4)  msg_filter(valid_or_source_error, m)(d)(s) =
           valid?(m(d)(s)) OR source_error?(m(d)(s))
(5)  send(m, st)(d)(s) = CASES st(s) OF trustworthy: m(s),
                                        recovering: m(s),
                                        benign: receive_error,
                                        symmetric: sym_send(m, s),
                                        asymmetric: asym_send(m, s, d) ENDCASES
(6)  encode(msgs)(s) = valid(msgs(s))
(7)  source_eligible_restrict(se, s)(d) = {x: below(S) | se(d)(x) AND x = s}
```

Remaining complex functions: `card, majority`

IC Protocol: Validity (IV)

Alternatives for further proceeding:

(a) Provide PVS implementations of remaining functions (prototypically accomplished for `card` and `majority`)

or

(b) Encode complex functions and dependents in propositional logic (requiring quantification over functions)

Summary

Accomplished:

- Presented method to transform higher-order terms to the propositional level
- Generalized applicability of SAT-solvers for PVS
- Started feasibility study on SPIDER IC protocol

To do:

- Elaborate transformation for concrete PVS syntax
- Implement transformation (within PVS?)