Formální metody - PVS

Radek Mařík

ČVUT FEL, K13132

October 2, 2014

Obsah

PVS [COR+95]

- "Prototype Verification System"
- SRI Computer Science Laboratory,
- Specifikační jazyk integrovaný s podpůrnými nástroji a dokazovačem vět.
  - Návrh jazyka podporuje efektivní mechanizované dokazovaní vět.
- Je dán důraz na záznam funkcionality, ne na efektivní implementaci.
- PVS je implementován v Common Lisp.
- PVS používá GNU Emacs jako uživatelské rozhraní.
- SunOS 4.1.3, ReadHat (Debian, SlackWare) Linux, AIX (IBM Risc 6000), Ultrix (DECSystem 5000).
- 50 megabytů diskového prostoru,
- minimálně 100 megabytů swap prostoru,
- minimálně 48 megabytů fyzické paměti.

PVS specifikační jazyk [COR+95]

- logika vyššího řádu s rozšířeným typovým systémem,
  - predikátové podtypy a závislé typy,
  - parametrizované teorie,
  - mechanismus definování abstraktních datových typů jako seznamy a stromy,
- **PVS specifikace** obsahuje několik souborů, každý s jednou či více teoriemi.
- **Teorie** může importovat jiné teorie.
- Specifikační soubory v PVS se identifikují pomocí *.pvs*
- Důkazy jsou drženy v souborech s koncovkou *.prf*
- Specifikace a soubory s důkazy v daném adresáři tvoří PVS *kontext*: udržovaný stav specifikace mezi sessions.
- PVS používá **funkcionálního** specifikačního stylu: "stav" systému je předáván jako argument funkcí.
- **Procedurální** styl specifikací (např. v Z) má vestavěné rozpoznání stavu.

PVS verifikátor důkazů [COR+95]

- Dokazovač je interaktivní a vysoce mechanizován.
- Snaha dokázat vlastnosti specifikací je nejefektivnější cesta k porozumění jejich obsahu a k nalezení chyb:
  - pokusy dokázat "reálnou" větu prokazující, že algoritmus splňuje daný účel.
  - proces přezkušování specifikací (výzvy) jako část validačního procesu.
  - **[** "animace" specifikace: běh testů - má smysl pouze pokud specifikace mají konstrukční charakter, programy na vysoké úrovni **]**.
- **Výzva** má tvar "je-li tato specifikace správná, potom by mělo platit i následující".
  - Testovací případ ve tvaru *domnělé* věty: obecné vyjádření o něčem, o čem si myslíme, že by mělo platit, za předpokladu platné specifikace.
  - Specifikace je "provedena" dokázáním vět.

Praktické kroky [COR+95]

- **Vytvoření specifikace**.
- **Syntaktická analýza** vytvoří interní abstraktní reprezentaci teorie specifikace.
- **Typová kontrola**:
  - ověření sémantických chyb, např. nedeklarovaná jména nebo nejasné typy,
  - může vygenerovat podmínky typové správnosti TCCs (Type-Correctness Condition) představující *dokazovací povinnosti*, které musí být splněny k tomu, aby teorie mohla být považována na typově platnou.
- **Dokazování**:
  - Základní úlohou důkazu je generovat **strom důkazu**, jehož listy jsou triviálně pravdivé.
  - Založeno na **sequentech**; formule nad čárkovanou čarou jsou nazývány **předchůdci** a ty pod čarou **následníky**.
  - Interpretace sequentu je, že konjukce předchůdců implikuje disjunkci následníků.
  - Prázdný předcůdce je ekvivalentní *true*, a prázdný následník je ekvivalentní *false*.

PVS v praxi [COR+95]

- verifikace mikroprocesoru, Collins Commercial Avionics. AAMP5 procesor je založen na mikrokódu a technologii pipeline; obsahuje okolo 500,000 tranzistorů.
- specifikace protokolu, LSI Logic, fiber channel protocol,
- aritmetika pohyblivé čárky, NASA Langley Research Center, IEEE 854 Floating Point standard
- bezpečnost dynamického spojování v Java, Princeton University,
- protokoly koherentnosti vyrovnávací paměti a paměťové modely, Stanford University,
- zobecněný problém křížení na železnicích, University of Namur, Belgium
- odvození číslicových obvodů, Indiana University,
- analýza požadavků na kritický software vesmírných lodí, Jet Propulsion Laboratory, sonda Cassini vyslaná k Saturnu, 1997

PVS základní syntaktické struktury [COR+95]

- % znak uvozuje komentář,
- entity a operace,

PVS typy [COR+95]

- rozlišitelné typy entit,
- *neinterpretované typy*
  - nevíme nic o jejich členech
  - elementy typu N liší od elementů typu P,

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

- *podtypy*

```
nat_to_10: TYPE = {x:nat | x <= 10}
```

- *enumerované typy*

```
color: TYPE = {red, green blue}
```

- *typy n-tic*

```
tuptype: TYPE = [int, bool, [int -> int]]
```

PVS asociace [COR+95]

páry, n-tice

```
(name, phone_number)
```

***funkce*: PVS je velmi efektivní v manipulaci s funkcemi**

```
B: TYPE = [N -> P]
```

PVS funkce [COR+95]

- totální funkce
  - axiomatické řešení

```
n0: P
emptyBook: B
emptyax: AXIOM
   FORALL (nm: n): emptyBook(nm) = n0
```

  - definiční řešení - predikátový podtyp

```
GP: TYPE =pn:P | pn /= n0
```

Gentzenův systém [Gal86]

Efektivní způsob testování, zda výrok A je tautologie, spočívá ve snaze najít ohodnocení, při kterém A neplatí.

- strom, jehož uzly jsou označeny páry konečných seznamů výroků,
- ohodnocení, při kterém všechny výroky první komponenty páru jsou pravdivé, a všechny výroky druhé komponenty jsou nepravdivé,
- **Sequent** je pár (Γ, Δ) konečných (nebo prázdných) sekvencí $\Gamma =< A_1, \cdots, A_m >, \Delta =< B_1, \cdots, B_n >$ výroků.
- Γ je nazýván předchůdce.
- Δ je nazýván následník.
- Sequent se obvykle označuje $\Gamma \rightarrow \Delta$.

Gentzenův systém - ohodnocení [Gal86]

- Ohodnocení v, za kterého je pravdivý sequent $A_1, \cdots, A_m \to B_1, \cdots, B_n$, jestliže

$$v \models (A_1 \wedge \cdots \wedge A_m) \supset (B_1 \vee \cdots \vee B_n).$$

- Ekvivaletně, pro v je sequent nepravdivý, jestliže pro v jsou všechny $A_1, \cdots A_m$ pravdivé a všechny $B_1, \cdots, B_n$ nepravdivé.
- Sequent je pravdivý, jestliže
  - jakýkoliv předchůdce je shodný s nějakým následníkem,
  - jestliže jakýkoliv předchůdce je nepravdivý
  - nebo jestliže kterýkoliv následník je pravdivý [?].

Inferenční pravidla sequentového kalkulu [Gal86]

- Γ, Δ, Λ ... libovolné sekvence výroků,
- A, B ... výroky

$$\frac{\Gamma, A, B, \Delta \to \Lambda}{\Gamma, A \wedge B, \Delta \to \Lambda} \quad [\wedge : \textit{left}]$$

$$\frac{\Gamma \to \Delta, A, \Lambda \quad \Gamma \to \Delta, B, \Lambda}{\Gamma \to \Delta, A \wedge B, \Lambda} \quad [\wedge : \textit{right}]$$

$$\frac{\Gamma, \Delta \to A, \Lambda \quad B, \Gamma, \Delta \to \Lambda}{\Gamma, A \supset B, \Delta \to \Lambda} \quad [\supset : \textit{left}]$$

$$\frac{A, \Gamma \to B, \Delta, \Lambda}{\Gamma \to \Delta, A \supset B, \Lambda} \quad [\supset : \textit{right}]$$

Jednoduchý příklad v PVS

```
simpleOne: THEORY
BEGIN

  P, Q, R: VAR bool
  logicTh: THEOREM
    P and (Q and R) IMPLIES (P and Q) and R

END simpleOne
```

Příklad: součet přirozených čísel

```
sum: THEORY
 BEGIN

 n: VAR nat

 sum(n): RECURSIVE nat =
  (IF n = 0 THEN 0 ELSE n + sum(n - 1) ENDIF)
  MEASURE id

 closed_form: THEOREM sum(n) = (n * (n + 1))/2

 END sum
```

Příklad: součet funkcí

```
sum2: THEORY
BEGIN

  n : VAR nat
  f,g : VAR [nat -> nat]

  sum(f,n) : RECURSIVE nat =
    IF n = 0 THEN
      0
    ELSE
      f(n-1) + sum(f, n - 1)
    ENDIF
  MEASURE n

  sum_plus : LEMMA
    sum((lambda n : f(n) + g(n)), n)
   = sum(f,n) + sum(g,n)

  square(n) : nat = n * n
  sum_of_squares : LEMMA
    6 * sum(square, n+1) = n * (n+1) * (2*n + 1)

  cube(n) : nat = n * n * n

  sum_of_cubes : LEMMA
    4 * sum(cube, n+1) = n*n*(n+1)*(n+1)

END sum2
```

Příklad: telefonní seznam - axiomy [COR+95]

```
phone_1: THEORY
BEGIN
  N: TYPE                 % names
  P: TYPE                 % phone numbers
  B: TYPE = [N -> P]      % phone books

  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
END phone_1
```

Příklad: definiční telefonní seznam [COR+95]

```
phone_3: THEORY
BEGIN
  N: TYPE                      % names
  P: TYPE                      % phone numbers
  B: TYPE = [N -> setof[P]]    % phone books
  nm, x: VAR N
  pn:    VAR P
  bk:    VAR B

  emptyBook(nm): setof[P] = emptyset[P]
  FindPhone(bk, nm): setof[P] = bk(nm)
  AddPhone(bk, nm, pn): B
    = bk WITH [(nm) := add(pn, bk(nm))]
  DelPhone(bk, nm): B
    = bk WITH [(nm) := emptyset[P]]
  DelPhoneNum(bk, nm, pn): B
    = bk WITH [(nm) := remove(pn, bk(nm))]
  FindAdd: CONJECTURE
    member(pn, FindPhone(AddPhone(bk,nm,pn),nm))
  DelAdd: CONJECTURE
    DelPhoneNum(AddPhone(bk, nm, pn), nm, pn)
      = DelPhoneNum(bk, nm, pn)
END phone_3
```

Příklad: přístup k dokumentům

```
cvs: THEORY
BEGIN

  Person: TYPE+
  Document: TYPE
  CheckedOut: TYPE = [Document -> Person]
  Permissions: TYPE = [Document -> setof[Person]]
  nobody: Person

  RealPerson: TYPE = {p: Person | p /= nobody}

  co: VAR CheckedOut
  p:  VAR RealPerson
  d:  VAR Document
  db:  VAR Permissions

  Locked?(co, d): bool = co(d) /= nobody
  Permitted?(db,d,p): bool = member(p,db(d))

  CheckOut(db, co, p, d): CheckedOut =
    IF Locked?(co,d) OR NOT Permitted?(db, d, p) THEN co ELSE co WITH [(d) := p] ENDIF
  FindPerson(co, d): Person = co(d)

  FindAddPT: CONJECTURE
    NOT Locked?(co, d) AND Permitted?(db, d, p) => FindPerson(CheckOut(db,co,p,d),d)=p

END cvs
```

Literatura I


Judy Crow, Sam Owre, John Rushby, Natarajan Shankar, and Mandayam Srivas.
A tutorial introduction to PVS.
In *Workshop on Industrial-Strength Formal Specification Techniques (Boca Raton, Florida)*, April 1995.


Jean H. Gallier.
Logic for Computer Science, Foundations of Automatic Theorem Proving.
Harper & Row, Publishers, New York, 1986.

PVS System demonstration

1. run pvs
2. context yes
3. Esc x ff
4. Enter cvs
5. Select the simpleOne theory
   1. Esc x tc
   2. Select the logicTh
   3. Esc x pr
   4. try again:yes
   5. rerun: no
   6. (skolem!)
   7. (flatten)
   8. (split 1)
   9. repeat with Esc x xpr to show the prove tree
6. (grind :theories ("cvs"))

PVS System demonstration

1. run pvs
2. context yes
3. Esc x ff
4. Enter cvs
5. Select the cvs theory
   1. Select the FindAddPT
   2. Esc x pr
   3. try again:yes
   4. rerun: no
   5. (skolem!)
   6. (flatten)
   7. (rewrite "FindPerson") ... (rewrite "CheckOut") ... (lift-if)
   8. (split) ... (flatten) ... (split) ... (flatten) ... (simplify)
6. (grind :theories ("cvs"))