Process Algebra in PVS

Twan Basten & Jozef Hooman

Eindhoven University of Technology
&
University of Nijmegen

Overview

Aim: investigate mechanical support for equational reasoning in process algebra, both for concrete applications and for theoretical properties.

1. Introduction
   - Process Algebra
   - PVS
2. Process terms as uninterpreted type
3. Process terms as abstract datatype
4. Example Alternating Bit Protocol
5. Concluding Remarks

Process Algebra

ACP-style process algebras (Baeten & Bergstra).

$$
\begin{array}{l}
\_\text{PA(A)}\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \\
P;\ \ \mathsf{A} \subseteq P \\
\hline
\_+\_,\ \ \_\cdot\_,\ \ \_\parallel\_,\ \ \_\llfloor\_ :\ \ P \times P \to P \\
\hline
a:\ \ \mathsf{A};\ \ x,y,z:\ \ P;
\end{array}
$$

$$
\begin{array}{ll}
x + y = y + x & A1 \\
(x + y) + z = x + (y + z) & A2 \\
x + x = x & A3 \\
(x + y) \cdot z = x \cdot z + y \cdot z & A4 \\
(x \cdot y) \cdot z = x \cdot (y \cdot z) & A5 \\
x \parallel y = x \llfloor y + y \llfloor x & M1 \\
a \llfloor x = a \cdot x & M2 \\
a \cdot x \llfloor y = a \cdot (x \parallel y) & M3 \\
(x + y) \llfloor z = x \llfloor z + y \llfloor z & M4
\end{array}
$$

Plus general substitution and context rules.

- Model (showing consistency) is, e.g., bisimulation on closed terms.
- Reasoning is model independent, based on axioms.

Aim: tool support for these algebras which:

- Supports concrete applications,
  with heavy rewriting,
  dealing with large and complex terms
  (esp. in case of concurrency).

  Example: Alternating Bit Protocol.

- Supports verification of
  theoretical properties.
  Closed terms rewritten to

  basic terms (a, $a \cdot x$, $x + y$)

  by *elimination* (e.g., $a \| b = (a \cdot b) + (b \cdot a)$).

  General property reduced to basic terms
  and proved by induction
  (on structure or length of basic terms).

  Example: associativity of merge.

- Is flexible; easily adaptable to variations
  of the algebra.

Not aiming at completely automatic verification.

PVS (Prototype Verification System) used to get tool support. Motivation:

- Interactive theorem prover with
  - Powerful rewriting possibilities.
  - Convenient facilities for induction, on well-founded sets and on structure of abstract datatypes.
  - Powerful decision procedures, reasonable degree of automation.
  - Convenient proof management.
- Easy to learn specification language, a typed higher-order logic.

  Typechecking might generate proof obligations (Type Check Conditions – TCCs).
- Large number of built-in theories and many convenient libraries.
- Free available, see:

  `http://www.csl.sri.com/sri-csl-pvs.html`
- Local expertise available.

Terms as Uninterpreted Types

```
PArew [Atoms: NONEMPTY_TYPE]: THEORY
BEGIN
Terms : NONEMPTY_TYPE
trm   : [Atoms -> Terms]
CONVERSION trm
+, o, //, lmrg : [Terms,Terms -> Terms]
a        : VAR Atoms
x, y, z : VAR Terms

A1 : AXIOM          x + y = y + x
A2 : AXIOM    (x + y) + z = x + (y + z)
A3 : AXIOM          x + x = x
A4 : AXIOM    (x + y) o z = x o z + y o z
A5 : AXIOM    (x o y) o z = x o (y o z)

M1 : AXIOM         x // y =
                    lmrg(x,y) + lmrg(y,x)
M2 : AXIOM      lmrg(a,x) = a o x
M3 : AXIOM  lmrg(a o x,y) = a o (x // y)
M4 : AXIOM  lmrg(x + y,z) =
                    lmrg(x,z) + lmrg(y,z)
END PArew
```

```
PArewex  : THEORY
BEGIN
Atoms : TYPE = {a,b,c,d}
IMPORTING PArew[Atoms]
expand : THEOREM (a+b) o (a+b) // (c+d) =
 a o (a o (c + d) + b o (c + d) +
     (c o (a + b) + d o (a + b))) +
 b o (a o (c + d) + b o (c + d) +
     (c o (a + b) + d o (a + b))) +
 c o (a o (a + b) + b o (a + b)) +
 d o (a o (a + b) + b o (a + b))
END PArewex

(AUTO-REWRITE-THEORY "PArew[Atoms]"
                     :EXCLUDE ("A1"))
OR
(AUTO-REWRITE
     "A2" "A3" "A4" "A5" "M1" "M2" "M3" "M4")
```

Terms as Abstract Datatype

```
PA_terms [Atoms: TYPE] : DATATYPE
BEGIN
  atm(at: Atoms)                : atom?
  o(sq1, sq2: PA_terms)         : seq?
  +(ch1, ch2: PA_terms)         : choice?
  //(mrg1, mrg2: PA_terms)      : merge?
  lmrg(lmrg1, lmrg2: PA_terms) : lmerge?
END PA_terms
```

Typechecking generates `PA_terms_adt`, with e.g.:

subterm relation `x << y` with axiom:

```
PA_terms_well_founded:
            AXIOM well_founded?[PA_terms](<<)
```

induction scheme, expressing that property `p` on terms can be proved by

- showing that it holds for all atoms and
- by proving that it holds for other operators if subterms already satisfy `p`.

Attempt to use equality in this approach:

```
Algebra [Atoms: NONEMPTY_TYPE]: THEORY
BEGIN

IMPORTING PA_terms[Atoms]

x, y, z : VAR PA_terms

A1 : AXIOM         x + y = y + x
A2 : AXIOM ( x + y ) + z = x + ( y + z )
A3 : AXIOM         x + x = x

a : Atoms

big_problem : LEMMA false

END Algebra
```

Problem: “=” already means syntactic equality on abstract datatype.

Solution:

Define separate equivalence relation "==".

```
PA[Atoms: NONEMPTY_TYPE]: THEORY
BEGIN
IMPORTING PA_terms[Atoms]
== : (equivalence?[PA_terms])
a, b, c       : VAR Atoms
v, w, x, y, z : VAR PA_terms

A1: AXIOM x + y == y + x
...
M4: AXIOM lmrg(x + y, z) ==
                 lmrg(x, z) + lmrg(y, z)
```

Substitution/context properties have to be expressed explicitly. For instance,

```
ch_l:  AXIOM x == z IMPLIES  x + y == z + y
...
mrg_l: AXIOM x == z IMPLIES x // y == z // y
```

Basic Terms and Elimination

Idea of elimination:
reduce (properties of) closed process terms to (properties of) basic terms, i.e. atoms, choices, atom-prefixes.

```
basic?(x) : RECURSIVE bool =
  CASES x OF
    atm(a)     : TRUE,
    o(y, z)    : atom?(y) AND basic?(z),
    +(y, z)    : basic?(y) AND basic?(z),
    //(y, z)   : FALSE,
    lmrg(y, z): FALSE
  ENDCASES
MEASURE x BY <<;

basic_terms : TYPE = {x | basic?(x)}
```

Translating PA Terms to Basic Terms

Show existence of a function `pa2b` with

    pa2b(x) == x

Our PVS proof explicitly constructs it:

```
PA2Basic[Atoms: NONEMPTY_TYPE]: THEORY
...
pa2b(x): RECURSIVE {b: basic_terms | b <= x} =
 CASES x OF
  atm(a)     : atm(a),
  o(y, z)    : CASES pa2b(y) OF
                atm(a) : atm(a) o pa2b(z),
                o(v, w): v o pa2b(w o z),
                +(v, w): pa2b(v o z) +
                            pa2b(w o z)
              ENDCASES,
  +(y, z)    : pa2b(y) + pa2b(z),
  //(y, z)   : pa2b(lmrg(y, z)) +
                 pa2b(lmrg(z, y)),
  lmrg(y, z): ...
              ENDCASES
 ENDCASES
MEASURE x BY <;
```

Note: dependent type. Generates 26 TCCs.
Problem: definition `<` and `<=`.

Definition < and <=

Note: not length of terms, since

```
pa2b(lmrg(y, z)) + pa2b(lmrg(z, y))
                              <=   y // z
```

Based on rewrite theory, we define

```
weight(x): RECURSIVE {n: nat | n >= 2} =
CASES x OF
  atm(a)     : 2,
  o(y, z)    : weight(y) * weight(z)
                  + weight(y),
  +(y, z)    : weight(y) + weight(z) + 1,
  //(y, z)   : expt(2,
                 weight(y) + weight(z) + 2),
  lmrg(y, z): expt(2, weight(y) + weight(z))
ENDCASES
MEASURE x BY <<;

<(x, y)   : bool = weight(x) < weight(y)

<=(x, y) : bool = x < y OR x = y

pa2b_eq: THEOREM pa2b(x) == x

END PA2Basic
```

Using Elimination

Structural Induction on Basic Terms

Example:
define alphabet by axioms for basic terms.

```
Alpha [Atoms: NONEMPTY_TYPE]: THEORY
BEGIN
IMPORTING PA2Basic [Atoms]
btx, bty, btz  : VAR basic_terms

alpha : [PA_terms -> setof[Atoms]] % alphabet

AB1: AXIOM alpha(atm(a)) = singleton(a)
AB2: AXIOM alpha(atm(a) o x) = add(a,alpha(x))
AB3: AXIOM alpha(x + y) =
           union(alpha(x),alpha(y))
AB4: AXIOM x == y IMPLIES alpha(x) = alpha(y)

AB2b: LEMMA alpha(btx o bty) =
            union(alpha(btx),alpha(bty))
AB2pa: THEOREM alpha(x o y) =
               union(alpha(x),alpha(y))
END Alpha
```

Proof of lemma `AB2b` by induction on `btx`.

Leads to `AB2pa` by elimination theorem `pa2b_eq`.

Using Elimination

Induction on Length of Basic Terms

Model-independent proof of

axioms of standard concurrency.

```
PAsc [Atoms: NONEMPTY_TYPE]: THEORY
BEGIN
IMPORTING PA2Basic[Atoms]
ASC2 : THEOREM  x // y == y // x

length(x) : RECURSIVE posnat = ...

ASC46b : LEMMA FORALL (n : nat) :
(FORALL btx,bty,btz:
 n = length(btx) + length(bty) + length(btz)
 IMPLIES lmrg(lmrg(btx,bty),btz) ==
         lmrg(btx,bty//btz) )
AND
(FORALL btx,bty,btz:
 n = length(btx) + length(bty) + length(btz)
 IMPLIES (btx//bty)//btz == btx//(bty//btz) )
```

Proof by induction on `n`.

This easily leads to

```
ASC4b : LEMMA lmrg(lmrg(btx,bty),btz) ==
              lmrg(btx,bty//btz)

ASC6b : LEMMA (btx//bty)//btz ==
              btx//(bty//btz)
```

Using elimination theorem `pa2b_eq`, plus symmetry, transitivity, and properties of "==" about substitution in context,
we obtain:

```
ASC4  : THEOREM lmrg(lmrg(x,y),z) ==
                lmrg(x,y // z)

ASC6  : THEOREM (x//y)//z == x//(y//z)
END PAsc
```

Conclusion: PVS facilities for
abstract datatypes and subtyping
are suitable for proof of non-trivial theorems
in process-algebra theory.
Rewriting is cumbersome this approach.

Verifying Applications

Use first approach with uninterpreted types and equality.

Benchmark: Alternating-Bit Protocol (ABP).

Uses extension of PA, called ACP$^{\tau *}$.
Includes

- synchronous communication,
  with communication function

  ```
  CM1 : AXIOM x // y =
                 lmrg(x,y) + lmrg(y,x) + (x / y)
  ```

- Binary Kleene star represents iteration:

  ```
        x * y
  ```

- Fair Iteration Rule and
  Recursive Specification Principle.

- Encapsulation and abstraction.

Alternating-Bit Protocol:

Specification:

```
ABP_spec:  Terms = (inp(d) o outp(d))*delta
```

Correctness statement:

```
ABP_eq_spec :  THEOREM
    abs(I,enc(H,S // R)) = ABP_spec
```

- `enc` (encapsulation) enforces synchronous communication.
- `abs` (abstraction) hides internal actions.

Typical proof, for instance to show

```
once_rep : LEMMA enc(H,S // R) =
                 (X(t) o X(f)) o enc(H,S // R)
```

uses explicit rewrites on `CM1` and automatic rewrites on most other axioms.

Concluding Remarks

Two approaches to formulate ACP-like process algebras in language of PVS.

Process terms as an uninterpreted type.

Convenient for concrete applications, using automatic rewrite mechanism of PVS.

Disadvantage: lack of induction principles, which are essential for proofs of theoretical results about process algebra.

Process terms as abstract datatype.

Convenient for theoretical properties, using induction principles of PVS.

Disadvantage: rewriting by means of the equivalence relation is inconvenient.

Similar to conclusion of earlier work using HOL (Groenboom, et al));
they found performance not acceptable.

Our conclusion: mechanical support for process algebra by PVS is feasible, both for theory development & concrete applications.

Unfortunately with different frameworks.

Wanted: combination with inductive proofs and convenient term rewriting. E.g.,

- Define powerful proof strategie in PVS, incorporating general rewrite patterns for congruences.
- Extend PVS with rewriting on user-defined congruence relations.
- Use generic theorem prover Isabelle, allows also ordered rewriting.

Other directions for future work:

- show soundness of axioms
- rewrite rules for clever expansion of parallel composition
- consider algebraic extensions, e.g. with real-time