Validity checking with if-then-else expressions modulo rich first-order theories

Silvio Ranise and David Déharbe*•
LORIA/CASSIS & INRIA-Lorraine
Nancy, France
{deharbe, ranise}@loria.fr
http://www.loria.fr/~ranise
http://www.loria.fr/~deharbe
*: UFRN/DIMAp – Natal, Brazil

• Partially supported by CAPES grant BEX0006/02-5.

March 05, 2003

Background

- Decision procedures are embedded in many theorem provers to accelerate proofs (SVC, CVC, *Simplify*, Acl2, PVS, HOL, Coq, Acl2, RDL, ...)

- Problem: decision procedures are designed and/or incorporated by *ad-hoc* means (most of the times)

  design ▷ difficulties in proving correctness

  incorporation ▷ difficulties in combining with other

  decision procedures[1] and reasoning activities[2]

[1] Shostak combination schema *docet!*

[2] such as handling the boolean structure and instantiating quantifiers

What

- Eliminating trivial subgoals represented as sequents modulo a background theory T
- In many cases, this can be reduced to satisfiability checking, i.e. proving the unsatisfiability of conjunctions of ground literals modulo T, by standard techniques

Example▷ $\mathcal{E}$ is the quantifier-free theory of equality.

$$\mathcal{E} \models f^5(c) = c, f^3(c) = c \Rightarrow f(c) = c$$

can be reduced to checking the $\mathcal{E}$-unsatisfiability of

$$f^5(c) = c \wedge f^3(c) = c \wedge f(c) \neq c$$

Who & How

Approach	Propositional	First-Order	Efficiency	Flexibility
naive	DNF	anything	hopeless	
Simplify	DNF + splitting	NO. coop. DPs	+/−	++
SVC	ITE + splitting	Sh. coop. DPs	+	+
CVC, ICS	SAT + lemmas	Sh. coop. DPs	++	+
UCLID, EVC, sep	SAT	reduction	++	− −

Our goal: maintain efficiency while augmenting flexibility

Approach	Propositional	First-Order	Efficiency	Flexibility
haRVey	BDD + pruning	Superposition	++	+++

Difficulties to overcome

- Sometimes, checking the satisfiability of ground literals is not enough

  ▷ the capability of finding suitable instantiations of existentially quantified variables is required

- Handling user-defined symbols commonly used to structure specifications

  ▷ no satisfactory solution but mixture of manual unfolding and validity checking

- *Simplify* features a heuristic matching mechanism to instantiate variables but... too heuristic $\Longrightarrow$ lack of "precision" (false negatives)

Plan of the talk

- Satisfiability checking of conjunctions of ground literals (superposition)
- Extension to arbitrary boolean combinations of ground literals (BDDs)
- Pruning (efficiency)
- Extension to non-ground formulae (flexibility)
- Applications
- On-going and future work

Plan of the Talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work

A Rewriting Approach to Satisfiability Procedures [Armando, Ranise & Rusinowitch, IC'03]

- Idea: pick a superposition calculus off-the-shelf and show that it can be used as a satisfiability procedure for the theory[3] of interest
- Generality: the approach works for a wide class of theories, e.g.
  - ▷ quantifier-free theory of equality
  - ▷ (variants of) theory of lists
  - ▷ theory of arrays and sets with/without extensionality
  - ▷ combination of arrays and lists

[3] whose satisfiability problem is decidable and which is presented by a finite set of clauses

A Rewriting Approach to Satisfiability Procedures—I

- Given
  - ▷ a set S of ground literals
  - ▷ a clausal presentation $Ax(T)$ of a theory T extending $\mathcal{E}$[4]
- We want to derive
  - ▷ a procedure capable of establishing whether S is T-satisfiable[5]

[4] $Ax(T)$ is not restricted to equations!

[5] i.e. $S \cup T$ is satisfiable

A Rewriting Approach to Satisfiability Procedures—II

- The recipe
  - ▷ Flatten the input set S of ground literals to obtain S'
  - ▷ Use a refutation complete superposition inference system and prove termination for $S' \cup Ax(T)$, for all set of ground flat literals S'
- Features
  - ▷ uniform[6] and simple[7]
  - ▷ easy to implement... take a state-of-the-art saturation theorem prover

[6] it works for theories which are important in verification

[7] based on the rewriting framework, which is well-understood

A Rewriting Approach to Satisfiability Procedures—III

- flattening = add "fresh" constants to name subterms

  Example ▷ $\{f(c, c') = h(h(a)), h(h(h(a))) \neq a\}$ is flattened to

$$\{f(c, c') = c_2, c_3 \neq a\} \cup \{c_1 = h(a), c_3 = h(c_2), c_2 = h(c_1)\}$$

Fact 1. *Let S be a finite set of Σ-literals. Then there exists a finite set of flat Σ'-literals S' s.t. S' is T-satisfiable iff S is.*[8]

- Termination proofs in the second step are substantially easier[9]

[8] Σ' is obtained from Σ by adding a finite number of constants

[9] easier to reason about superpositions when depth of terms ≤ 1

A Rewriting Approach to Satisfiability Procedures—IV
The Superposition Calculus: Inference Rules

Name	Rule	Conditions
Sup.	$\dfrac{\Gamma \rightarrow \Delta, l[u'] = r \quad \Pi \rightarrow \Sigma, u = v}{\Gamma, \Pi \rightarrow \Delta, \Sigma, l[v] = r}$	$u \not\preceq v$, $l[u'] \not\preceq r$, $*$
Par.	$\dfrac{\Gamma, l[u'] = r \rightarrow \Delta \quad \Pi \rightarrow \Sigma, u = v}{l[v] = r, \Gamma, \Pi \rightarrow \Delta, \Sigma}$	$u \not\preceq v$, $l[u'] \not\preceq r$, $*$
Ref.	$\dfrac{\Gamma, u' = u \rightarrow \Delta}{\Gamma \rightarrow \Delta}$	$(u' = u) \not\prec (\Gamma \cup \Delta)$
Fac.	$\dfrac{\Gamma \rightarrow \Delta, u = t, u' = t'}{\Gamma, t = t' \rightarrow \Delta, u' = t'}$	$u \not\preceq t$, $u \not\preceq \Gamma$, $(u = t) \not\prec \{u' = t'\} \cup \Delta$

$*$ $(u = v) \not\preceq (\Pi \cup \Sigma)$, $(l[u'] = r) \not\prec (\Gamma \cup \Delta)$

$**$ $\sigma = mgu(u, u')$ implicitly applied to consequents and conditions

A Rewriting Approach to Satisfiability Procedures—V
The Superposition Calculus: Simplification Rules

Name	Rule	Conditions
Subsumption	$\frac{S \cup \{C, C'\}}{S \cup \{C\}}$	for some θ, $\theta(C) \subseteq C'$, and for no ρ, $\rho(C') = C$
Simplification	$\frac{S \cup \{C[\theta(l)], l = r\}}{S \cup \{C[\theta(r)], l = r\}}$	$\theta(l) \succ \theta(r)$, $C[\theta(l)] \succ (\theta(l) = \theta(r))$
Deletion	$\frac{S \cup \{\Gamma \rightarrow \Delta, t = t\}}{S}$	

A Rewriting Approach to Satisfiability Procedures—VI
The Superposition Calculus: Details

- Reduction Ordering $\succ$ total on ground terms and s.t. $f(c_1, \ldots, c_n) \succ c_0$, for each n-ary function symbol f and constants $c_1, ..., c_n$ $(n \geq 1)$

  ▷ $(a = b) \succ_e (c = d)$ iff $\{a, b\} \gg \{c, d\}$[10], clauses (i.e. multisets of literals) are compared by the multiset extension of $\succ_e$

- Refutation complete = any fair application of the rules to an unsatisfiable set of clauses will derive the empty clause

- Saturation of a set of clauses is the final set of clauses generated by a fair derivation using the rules with higher priority given to the simplification rules

[10] $\gg$ is the multiset extension of $\succ$ on terms

A Rewriting Approach to Satisfiability Procedures—VII
A Satisfiability Procedure for the theory $\mathcal{A}$ of Arrays

- Finite presentation

$$Ax(\mathcal{A}) := \left\{ \begin{array}{l} \mathsf{read}(\mathsf{write}(A, I, E), I) = E \\ I \neq J \Rightarrow \mathsf{read}(\mathsf{write}(A, I, E), J) = \mathsf{read}(A, J) \end{array} \right\}$$

- Intuition

$$\mathsf{read}\left(\begin{array}{c|c} i_1 : & e_1 \\ \hline i_2 : & e_2 \\ \hline i_3 : & e_3 \\ \hline i_4 : & e_4 \end{array}, i_2 \right) = e_2 \qquad \mathsf{write}\left(\begin{array}{c|c} i_1 : & e_1 \\ \hline i_2 : & e_2 \\ \hline i_3 : & e_3 \\ \hline i_4 : & e_4 \end{array}, i_2, e_2' \right) = \begin{array}{c|c} i_1 : & e_1 \\ \hline i_2 : & e_2' \\ \hline i_3 : & e_3 \\ \hline i_4 : & e_4 \end{array}$$

Lemma 1. *Let S be a finite set of flat literals. The clauses occurring in the saturations of $S \cup Ax(\mathcal{A})$ by $\mathcal{SP}$ can only be:*

i) the empty clause; *ii) axioms* *iii) ground flat literals*

iv) clauses of type $t \bowtie t' \vee c_1 = c_1' \vee \cdots \vee c_n = c_n'$
with $t \bowtie t' \in \{c \neq c', \mathsf{read}(c, i) = c', \mathsf{read}(c, i) = \mathsf{read}(c', i')\}$

v) clauses of type $\mathsf{read}(c, x) = \mathsf{read}(c', x) \vee c_1 = k_1 \vee \cdots \vee c_n = k_n$, where k_i is either x or c_i.[11]

Proof ▷ simple induction on the length of derivations $\Longrightarrow$ saturations of $S \cup Ax(\mathcal{A})$ are finite

Theorem 1. *$\mathcal{SP}$ is a satisfiability procedure for $\mathcal{A}$.*

Remark ▷ coarse grained complexity result $O(2^{n^2})$[12]

[11] $i, c, c', c_1, c_1', \ldots, c_n, c_n'$ are constants, and x is a variable

[12] the satisfiability problem for $\mathcal{A}$ is "only" $O(2^n)$ [Downey & Sethi, JACM 25(4)]

Plan of the Talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work

Arbitrary combination of ground literals—I

Starting with a naive approach:

- to show that a ground formula φ is $\mathcal{T}$-valid:
  - ▷ put $\neg\varphi$ into DNF: $\{S_i \mid i = 1, ..., n\}$
  - ▷ if there exists $j \in \{1, ..., n\}$ s.t. S_j is $\mathcal{T}$-satisfiable, then φ is not $\mathcal{T}$-valid and S_j is a "counter-example"
  - ▷ if for all $i = 1, ..., n$ S_i is $\mathcal{T}$-unsatisfiable, then φ is $\mathcal{T}$-valid
- Ideas: (1) abstract atoms to propositional vars,

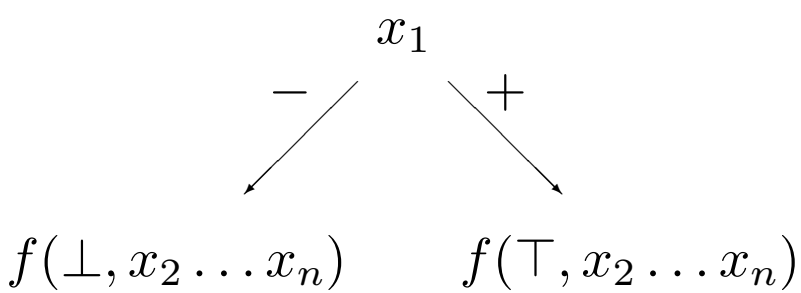
  (2) use BDDs to represent DNFs,

  (3) check $\mathcal{T}$-satisfiability by saturation

Arbitrary combination of ground literals—II

- Reduced Ordered Binary Decision Diagrams (BDDs) provide an efficient data structure for propositional reasoning.

- The BDD of a formula $f(x_1, x_2 \ldots x_n)$ can be viewed as Shannon's expansion: $[x_1 \wedge f(\top, x_2 \ldots x_n)] \vee [\neg x_1 \wedge f(\bot, x_2 \ldots x_n)]$.



Arbitrary combination of ground literals—III

- Reduced: no distinct isomorphic subtrees
- Ordered: variables appear on a given order from root to leaves
- BDDs are canonical (for propositional logic):
  - ▷ a formula is valid iff its BDD is the same as the BDD for $\top$
  - ▷ a formula is satisfiable iff it is different from the BDD for $\bot$: each branch from a root to the $\top$ leaf is a satisfiable valuation
  - ▷ no search*!*

Arbitrary combination of ground literals—IV

- BDDs are no more canonical for first-order logic

  Example ▷

$$
\begin{array}{l}
\text{ite}(x = y, \\
\quad \text{ite}(p(x), \\
\qquad\quad \text{ite}(p(y), \bot, \top), \\
\qquad\quad \top) \\
\quad p(x))
\end{array}
$$

Notice: the $\top$-branch $x = y \wedge p(x) \wedge \neg p(y)$ is not $\mathcal{E}$-satisfiable

- Search for T-satisfiable branches (= conjunctions of ground literals)
  - ▷ check branches for T-(un)satisfiability by superposition

Arbitrary combination of ground literals—V

function $search\_satisfiable$ (T: $theory$; ϕ: $formula$)
  while $\phi \neq \bot$ **do**
    $\beta \longleftarrow branch(\phi)$
    $\rho \longleftarrow saturate(\mathsf{T}, \beta)$
    if $[] \notin \rho$ **then return** (yes, β)
    $\phi \longleftarrow \phi \wedge \neg\beta$
  done
  return $(no, -)$
end

- Unfortunately, inefficient because of exhaustive search (exponentially many branches)

Plan of the talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work

Pruning—I

- Avoid repeated T-satisfiability checking of "similar" branches

  Intuition ▷ many branches in the BDD with the same prefix containing the literals which are "responsible" for the T-unsatisfiability

- S_1 and S_2 are "similar" if there exists S s.t. $S \subset S_1$, $S \subset S_2$, and the T-unsatisfiability of S entails the T-unsatisfiability of S_1 and S_2

  ▷ the problem of determining the smallest set S is NP-complete

  ▷ in practice: S is (heuristically) "small"

- modify $saturate$ to return also the proof of the empty clause

  ▷ the proof will contain S, (usually) a subset of the branch β

Pruning—II

Pruning subsumed branches

- Let π be the saturation proof showing that the branch β is $\mathcal{T}$-unsatisfiable;
- Let $hypothesis(\pi)$ be the set of all the literals in π belonging to the branch β;
- Then $hypothesis(\pi)$ is $\mathcal{T}$-unsatisfiable, and $\neg hypothesis(\pi)$ is $\mathcal{T}$-satisfiable.

Pruning—III

function $search\_satisfiable$ (T: $theory$; ϕ: $formula$)
  while $\phi \neq \bot$ **do**
    $\beta \longleftarrow branch(\phi)$
    $(\rho, \pi) \longleftarrow saturate(\mathsf{T}, \beta)$
    if $[\,] \notin \rho$ **then return** (yes, β)
    $\phi \longleftarrow \phi \wedge \neg hypothesis(\pi)$
  done
  return $(no, -)$
end

haRVey—I

These techniques have been implemented in haRVey:

- it uses:
  - ▷ CWI's ATerm library
  - ▷ Schulz E-prover
  - ▷ David Long's BDD library
- available on the Web at
  http://www.loria.fr/equipes/cassis/softwares/haRVey

haRVey—II

- Input: $\langle Ax(\mathcal{T}), \varphi \rangle$, where
  - ▷ $Ax(\mathcal{T})$ axiomatises the background theory
  - ▷ φ is the formula to be proved valid.
- Output:
  - ▷ Yes or
  - ▷ No, and a "counter-example"

Experimental results

- We compare the execution time of haRVey with and without pruning of subsumed branches

- The benchmark we use are the verification condition (valid purely equational ground formulae) for the proof of Burns' mutual exclusion protocol[13]

Proof obligation	elimination		Proof obligation	elimination	
	off	on		off	on
Burns 6	0.90	0.17	Burns 8	25.50	0.72
Burns 12	0.73	0.26	Burns 14	0.05	0.05
Burns 23	10.53	0.09	Burns 24	0.87	0.08
Burns 28	14.94	0.24	Burns 31	0.87	0.16
Burns 33	8.97	0.25	Burns 34	14.72	0.23
Burns 36	13.73	0.84	Burns 37	18.23	0.22

▷ average speed-up: 2 orders of magnitude

[13] Warning: The natural number N in Burns$N$ uniquely identifies a proof obligation. It is by no means related to the size of the proof obligations.

Plan of the talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work

Instantiating Quantifiers–I

- Problem: sometimes checking the satisfiability of ground literals is not enough

  ▷ the capability of finding suitable instantiations of existentially quantified variables is required

  for example ▷ checking the validity of formulae of the following form is required:

$$\underbrace{(\forall x_1, ..., x_n\, \phi)}_{\text{implicit existential quantif.}} \implies (\forall y_1, ..., y_m\, \psi)$$

where the only free var's of ϕ (ψ) are $x_1, ..., x_n$ ($y_1, ..., y_m$, resp.)

Instantiating Quantifiers–II

- Solution: use basic properties of the consequence relation $\models$ to pre-process the proof obligation and then invoke haRVey with an extended background theory

i.e. ▷ $\mathcal{T} \models (\forall x_1, ..., x_n\, \phi) \Longrightarrow (\forall y_1, ..., y_m\, \psi)$ can be transformed to

$$\underbrace{\mathcal{T}, \forall x_1, ..., x_n\, \phi}_{\text{extended background theory}} \models \forall y_1, ..., y_m\, \psi$$

similarly ▷ for proof obligations of the form $(\exists x_1, ..., x_n\, \phi) \Longrightarrow (\exists y_1, ..., y_m\, \psi)$ since they can be transformed into

$$(\forall x_1, ..., x_n\, \neg\psi) \Longrightarrow (\forall y_1, ..., y_m\, \neg\phi)$$

Instantiating Quantifiers–III

- Notice that we are hiding the difficulties in the background reasoner
- Fortunately, **haRVey** adopts a full-fledge first-order equational prover as background reasoner
- So, in **haRVey**: integrating

$\mathcal{T}$-satisfiability checking of ground literals

and

quantifier reasoning for $\forall x_1, ..., x_n\, \phi$

is theorem proving in the theory $\mathcal{T} \cup \{\forall x_1, ..., x_n\, \phi\}$

- Remark: fully automatic semi-decision procedure, no user-guidance!

Instantiating Quantifiers–IV

- A general approach

  ▷ replace each quantified subformula $Qx1 \ldots x_n \phi(x_1 \ldots x_n)$ with a fresh propositional constant p;

  ▷ add the clausification of formula $p \Longleftrightarrow Qx1 \ldots x_n \phi(x_1 \ldots x_n)$ to the background theory

  refinement ▷ take into account polarity, e.g. $T \models \forall x.A(x) \Rightarrow G$

  to check the T-unsatisfiability of $\forall x.A(x) \wedge \neg G$, it is sufficient

  to check the $T \cup \{p \Rightarrow \forall x.A(x)\}$-unsatisfiability of $p \wedge \neg G$

Instantiating Quantifiers–V

Example

$$\mathcal{T} \models (a = a' \wedge \exists x.a = ins(x, \emptyset)) \Longrightarrow \exists x.a' = ins(x, \emptyset)$$

quantified subformulae are replaced with constants:

$$\mathcal{T} \cup \{p \Leftrightarrow \exists x.a = ins(x, \emptyset), p' \Leftrightarrow \exists x.a' = ins(x, \emptyset)\} \models (a = a' \wedge p) \Longrightarrow p'$$

axioms are clausified:

$$\mathcal{T} \cup \{\neg p \vee a = ins(sk, \emptyset), p \vee a = ins(X, \emptyset), \ldots\} \models (a = a' \wedge p) \Longrightarrow p'$$

Plan of the talk

- Satisfiability checking of conjunctions of ground literals
- Extension to arbitrary boolean combination of ground literals
- Pruning
- Extension to non-ground formulae
- Applications
- On-going and future work

Applications—I

- Parameterized protocols (Burns) $\surd$
- Debugging of pointer-manipulating programs
- Correctness of (imperative) programs
- Push-button proofs of invariants of B machines (work in progress)

Applications—II

Debugging of pointer-manipulating programs (written in C)

- Loops are unrolled a fixed number of times (sufficient to expose bugs)
- Theory of arrays to model pointer (de-)referencing
- Theory of lists to reason about records
- This approach allows for (declarative)
  - ▷ cleanness checking of pointer-manipulating prg.[14]
  - ▷ verification of user specified annotations

[14] See the paper "*Checking Cleanness in Linked Lists*" by N. Dohr, M. Rodeh, and M. Sagiv. In Proc. of the Static Analysis Symposium 2000.

Applications—III

(a) user-defined property

	haRVey	E	*Simplify*
search^1	0.08	**0.03**	**0.03**
search^2	0.17	**0.17**	**0.10**
search^3	0.68	**0.16**	0.71
search^4	**0.16**	0.36	1.78
search^5	**0.17**	1.22	21.36
search^6	**0.23**	5.48	mem.out
remove^1	0.15	**0.02**	**0.02**
remove^2	0.19	0.39	**0.14**
remove^3	**0.21**	11.40	2.18
remove^4	**0.37**	40.55	mem.out

(b) read undefined

	haRVey	E	*Simplify*
fumble^3	0.06	1.62	**0.04**
insert^3	0.34	111.89	**0.29**
merge^2	**0.19**	time out	3.61
remove^3	**0.04**	time out	3.46
remove-all^3	0.06	**0.03**	**0.03**
reverse^3	0.26	2.64	**0.08**
rotate^3	0.48	10.80	**0.11**
search^3	**0.33**	145.40	0.79
swap^3	0.21	0.10	**0.04**

▷ The numbers appearing as exponents to the names of the proof obligations indicate how many times the loop in the corresponding C program has been unrolled.

Applications—IV

(c) null pointer dereferencing

	haRVey	E	*Simplify*
insert^6	**0.83**	time out	3.22
merge^4	**5.83**	sp. out	41.71
remove^4	**0.05**	sp. out	9.69
remove-all^3	0.06	0.05	**0.02**
reverse^8	**0.20**	time out	0.51
rotate^8	**0.39**	time out	0.85
swap^3	0.25	0.12	**0.04**

(d) memory leakage

	haRVey	E	*Simplify*
fumble^2	**0.10**	time out	38.08
insert^3	**0.07**	time out	25.93
merge^1	**0.48**	sp. out	mem. out
remove^2	**0.11**	sp. out	mem. out
remove-all^3	**0.05**	508.14	0.65
reverse^2	**1.20**	sp. out	mem. out
rotate^2	**1.68**	sp. out	mem. out
search^3	**0.05**	time out	3.64
swap^3	**0.05**	time out	1.17

Applications—V

Correctness of (imperative) algorithms

- challenge: automatically check all the proof obligations of Union-Find algorithm given in Nelson's POPL'81 paper

- Simplify (heuristic) handling of quantifier is not capable of coping with such a complex situation

  ▷ sophisticated pointer manipulations

  ▷ Nelson's formalization of linked lists is different from the previous one ⇒ flexibility of haRVey w.r.t. changes in the background theory

Applications—VI

	B1	B2	B3	B4	B5	B6
haRVey	0.03	0.06	0.03	0.03	0.06	0.14
E prover	0.03	0.03	0.10	23.16	0.03	0.04
Simplify	—	0.01	—	0.04	0.01	0.01
	B7	B8	B9	B10	B11	B12
haRVey	0.11	0.33	0.69	2.46	0.31	1.50
E prover	0.04	0.18	0.75	time out	0.13	0.65
Simplify	0.20	—	†	—	—	—

— indicates that *Simplify* gave a (false) negative answer.
† indicates that *Simplify* aborted.

From *Verifying Reachability Invariants of Linked Structures*, by G. Nelson, Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages (POPL'83), pp. 38–47, 1983.

Applications—VII

(jww A. Giorgetti and J.-F. Couchot, LIFC)

Push-button proofs of invariants of B machines

- B machines are annotated with invariants;
- B machines are written with set theory operators;
- standard methodology to generate first-order proof obligations which entails that an invariant is an inductive invariant;
- B constructs require quantifiers

Applications—VIII

```
MACHINE   scheduler
SETS      PID
VARIABLES active, ready, waiting
INVARIANT active <: PID /\ ready <: PID /\ waiting <: PID /\
          card(active) <= 1 /\
          ready /\ active = {} /\       waiting /\ ready = {} /\ active /\ waiting = {}
INITIALISATION
        active, ready, waiting := {}, {}, {}
OPERATIONS
        new(pp) = PRE pp: PID - (active \/ ready \/ waiting) THEN waiting := waiting \/ {pp} END
        swap = PRE active /= {} THEN
                 waiting := waiting \/ active ||
                 IF (ready /= {}) THEN
                   ANY pr WHERE pr: ready THEN active := {pr} || ready := ready - {pr} END
                 END
        ready = PRE pw: waiting THEN
                  waiting := waiting - {pw} ||
                  IF active /= {} THEN ready := ready \/ {pw}
                  ELSE active := {pw} END
                END
        del(dd) = PRE dd: waiting THEN waiting := waiting - {dd} END
END
```

Future Work & Advertising

- Push-button theorem prover for invariant verification of B machines
- integration of arithmetic reasoning in haRVey
  - ▷ (generalization of) Nelson and Oppen combination schema
- Benchmarks, benchmarks, benchmarks, benchmarks, benchmarks, ...
  - ▷ SMT-LIB Initiative: Satisfiability Modulo Theory Library
    http://combination.cs.uiowa.edu/smtlib
  - ▷ PDPAR Workshop (affiliated to CADE): Practical Aspects of Decision Procedures in Automated Reasoning
    http://www.loria.fr/~ranise/pdpar03

Take a look at http://www.loria.fr/equipes/cassis/softwares/haRVey

- A rewriting approach to satisfiability procedures

▷ A. Armando, S. Ranise, M. Rusinowitch. **Uniform Derivation of Decision Procedures by Superposition.** In Annual Conf. on Computer Science Logic (CSL01), 2001.

▷ A. Armando, S. Ranise, M. Rusinowitch. **The Rewriting Approach to Satisfiability Procedures.** To appear in the J. of Information and Computation, 2002.

- Combining theory and propositional reasoning

▷ D. Déharbe and S. Ranise. **BDD-Driven First-Order Satisfiability Procedures.** INRIA Technical Report 4360, November 2002.