Deciding Propositional Combinations of Equalities and Inequalities

Jean-Christophe Filliâtre
LRI, Orsay, France

Joint work with Sam Owre, Harald Rueß and Natarajan Shankar
SRI International, Menlo Park, USA

ICASE, August 22nd, 2001

Plan

1. Overview of ICS
2. A combination of proof techniques
3. The implementation of ICS

Part I

Overview of ICS

ICS (Integrated Canonizers and Solvers)

ICS is a decision procedure for the combination of

- propositional satisfiability
- equational reasoning over uninterpreted function symbols
- equational reasoning combined with given theories (arithmetic, tuples, arrays, sets, bit vectors)
- linear inequations (over rationals)

Examples

ICS can prove

- $f^3(x) = f(x) \vdash f^5(x) = f(x)$
- $\vdash 2x + 3y = 13 \;\wedge\; 3x = 2y \;\Rightarrow\; x = 2$
- $f(f(x) - f(y)) \neq f(z),\; y \leq x,\; y \geq x + z,\; z \geq 0 \vdash \mathsf{false}$
- $a + 2 = b \vdash f(A[a := 3][b - 2]) = f(b - a + 1)$
- $\vdash ((p \Rightarrow q) \Rightarrow p) \Rightarrow p$

ask/tell interface

Interface designed for an incremental use

(PVS processes $a_1, \ldots, a_n \vdash b_1, \ldots, b_m$ incrementally)

```
type state
val empty_state : state

val process : state → prop → ⊥+state

val check : state → prop → bool
```

Part II

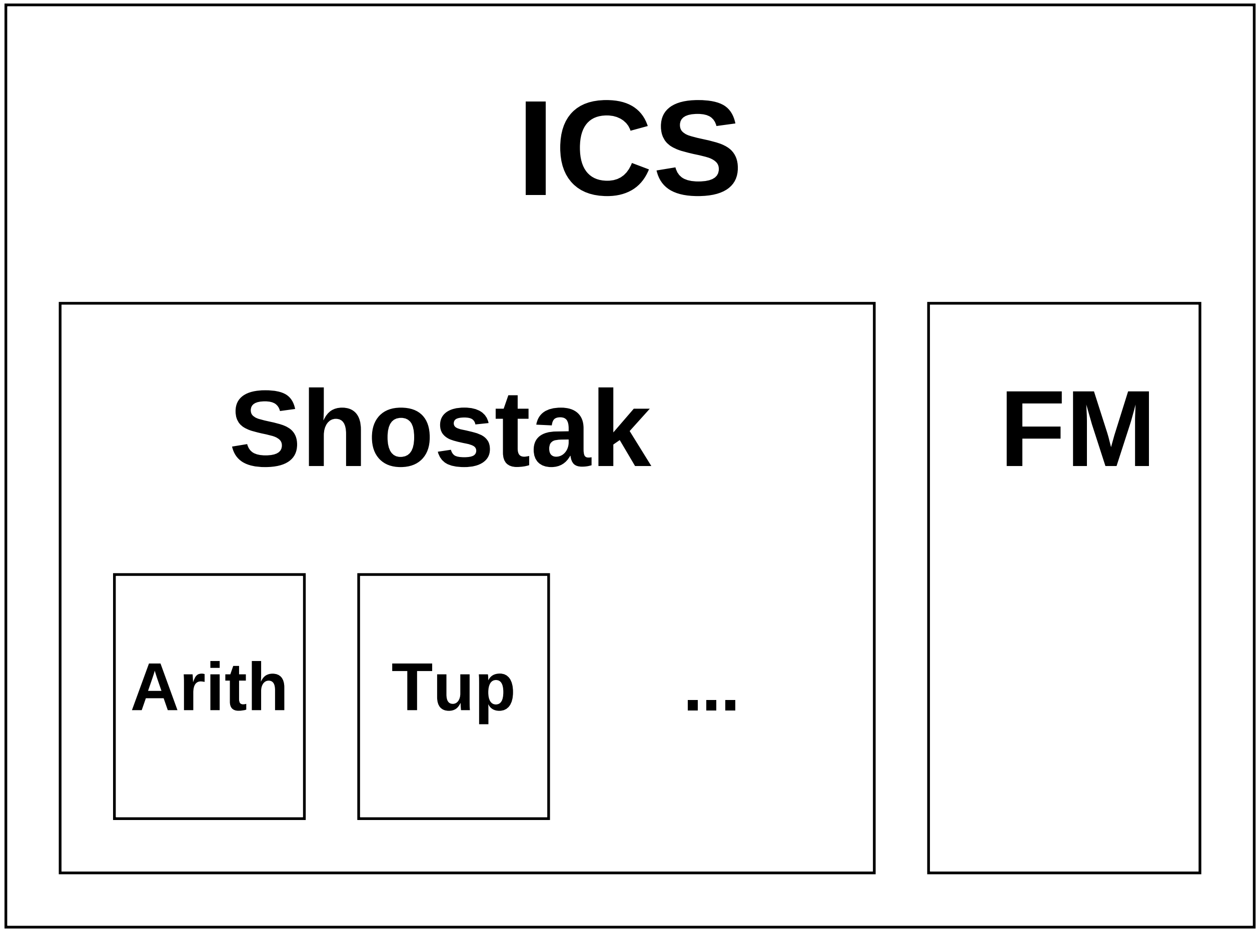
A combination of proof techniques

A combination of proof techniques

ICS is made of

- Shostak algorithm for equality reasoning
- Fourier-Motzkin elimination for inequality reasoning
- An original combination with a satisfiability checker

ICS modular design



ICS

Shostak

FM

Arith

Tup

...

Terms

terms $t ::= x \mid f(t, \ldots, t)$

atoms $a ::= t = t \mid t \neq t \mid t < t \mid t \leq t \mid \textsf{true} \mid \textsf{false}$

propositions $p ::= a \mid \textsf{if } a \textsf{ then } p \textsf{ else } p$

Equality reasoning: Shostak algorithm

Shostak's original algorithm actually non-terminating and incomplete. Recently fixed in

Deconstructing Shostak (Shankar and Rueß, LICS 2001)

```
type estate (* a set of equations *)

val eprocess : estate → term = term → ⊥+estate

val canon : estate → term → term
```

Inequality reasoning

Fourier-Motzkin elimination

```
type istate (* a set of inequations *)

val iprocess : istate → ⊥+equations
```

The state of ICS

The state of ICS is a quadruple (E, D, I, P) where

- E is a set of equations (a state for Shostak)
- D is a set of disequations
- I is a set of inequations (a state for Fourier-Motzkin)
- P is a set of propositions

The process function

$$
\begin{array}{lll}
\multicolumn{3}{l}{process(E,D,I,P)(p) = \textsf{match } canon(E)(p) \textsf{ with}} \\
\mid\ a = b & \Rightarrow & \textsf{let } E' = eprocess(E)(a = b) \textsf{ in} \\
 & & processmany(E', \emptyset, \emptyset, \emptyset)(D \cup I \cup P) \\
\mid\ a \neq b & \Rightarrow & \textsf{if } a \equiv b \textsf{ then } \bot \textsf{ else } (E, \{a \neq b\} \cup D, I, P) \\
\mid\ a \lhd b & \Rightarrow & \textsf{let } E' = iprocess(\{a \lhd b\} \cup I) \textsf{ in} \\
 & & processmany(E, D, \{a \lhd b\} \cup I, P)(E') \\
\mid\ \textsf{true} & \Rightarrow & (E, D, I, P) \\
\mid\ \textsf{false} & \Rightarrow & \bot \\
\mid\ p' & \Rightarrow & (E, D, I, \{p'\} \cup P)
\end{array}
$$

$$processmany(S)(\{p_1, \ldots, p_n\}) = process((\ldots process(S, p_1) \ldots), p_n)$$

The `check` function

We actually check the inconsistency of a state, since

$$check(S)(p) \quad \equiv \quad inconsistent(process(S)(\neg p))$$

The check function

```
$inconsistent(E, D, I, P) =$ match $P$ with
  | $\emptyset \Rightarrow$ false
  | $\{$if $a$ then $p$ else $q\} \cup P' \Rightarrow$
      let $S' = processmany(E, D, I, P')(\{a, p\})$ in
      let $S'' = processmany(E, D, I, P')(\{\neg a, q\})$ in
      match $S', S''$ with
        | $\bot, \bot \Rightarrow$ true
        | $\bot, \_ \Rightarrow inconsistent\ S''$
        | $\_, \bot \Rightarrow inconsistent\ S'$
        | $\_ \Rightarrow inconsistent\ S' \ \wedge\ inconsistent\ S''$
```

Results

$process$ always terminates and is sound.

$inconsistent$ always terminates, is sound and complete.

Part III

The implementation of ICS

The big picture

ICS is written in Objective Caml, a dialect of ML developed at INRIA (Xavier Leroy et al.)

The code of ICS is 5000 lines long

Uses GMP (The GNU Multi Precision library)

ICS is documented using ocamlweb, a literate programming tool for Ocaml ⇒ a 80 pages document

ICS is both an Ocaml and a C library (and even a Lisp one)

Keys to efficiency

- hash-consing: structurally equal terms are physically shared
- efficient data structures to represent polynomials, maps from terms to terms, states, etc.
- persistent (functional) data structures ⇒ no need to copy
- strongly typed language ⇒ compact memory representation

Shipping

ICS comes both as a library and as a standalone interactive program (also batch interpreter)

```
% ics

> assert y <= x; y >= x + z; z >= 0.

> check x = y.
Valid.
```

Comparison with SVC (Stanford Validity Checker)

Linear systems of N equations with integer solutions

N	ICS		SVC	
5	0.05		0.07	
10	0.43		0.44	
15	2.36		1.84	
20	8.85		8.84	
25	25.04		23.05	
30	61.95	(29 Mo)	71.18	(91 Mo)
35	130.25	(40 Mo)	> 4 mn	(> 150 Mo)

Conclusion

Positives

- A decision procedure independent of PVS
- A completeness property
- A functional implementation $\Rightarrow$ easy soundness argument (and may be a formal proof)

Negatives

- Fairly naive implementation of Fourier-Motzkin (SVC or Omega really faster here)
- Does not compete with specialized decision procedures
- Still needs heavy benchmarks (work on progress)