

CC(X): Efficiently Combining Equalities and Solvable Theories without Canonizers

Sylvain Conchon, Evelyne Contejean and Johannes Kanig

PROVAL Project

Université Paris Sud / CNRS / INRIA

SMT 2007

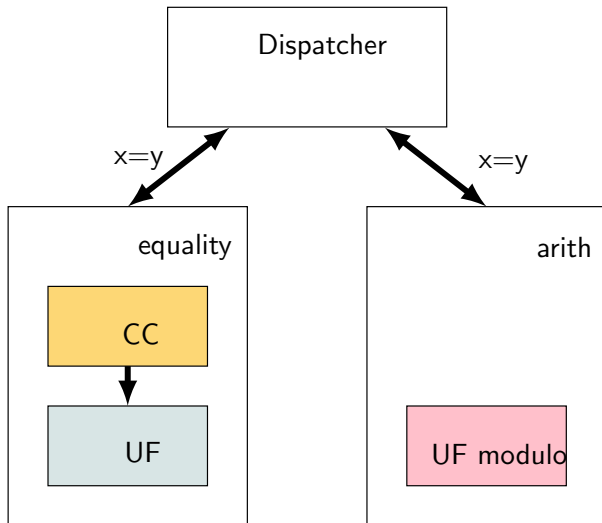


- the **Why** tool: a VCG generator based on a polymorphic language/logic
- the **Caduceus** and **Krakatoa** tools for verifying C and Java programs, respectively
- the **Ergo** automatic theorem prover

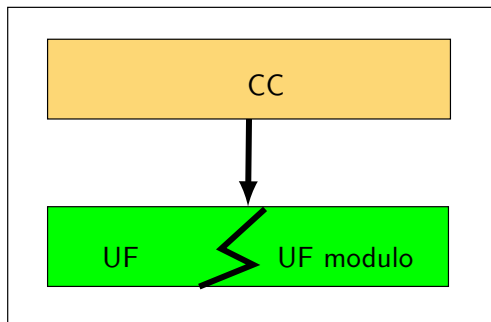
`http://{krakatoa,caduceus,why,ergo}.lri.fr`

Combining equality with a theory X

Nelson-Oppen architecture



Shostak architecture



Congruence closure modulo X

- SHOSTAK *Deciding combination of theories*, 1984
- SHANKAR-RUESS *Deconstructing Shostak*, 2001

- the algorithm maintains a partition of terms within a dictionary mapping terms to their term representatives
- the theory X must have a *canonizer* and a *solver*

given an equation $u = t$, the main steps of the algorithm are:

- 1 apply global canonization to u and t
- 2 $\text{solve}(\bar{u}, \bar{t})$ returns a substitution σ , the solution of $\bar{u} = \bar{t}$
- 3 σ is applied to all representatives
- 4 the representatives are reduced to normal forms by the canonizer
- 5 find new equations that hold by congruence

A running example

$$g(x+k) = a \wedge s=g(k) \wedge x=0 \wedge f(g(x+k))=f(a) \models_x s = a$$

terms	representatives
a	$g(x + k)$
s	$g(k)$
$g(k)$	$g(k)$
$g(x + k)$	$g(x + k)$
$x + k$	$x + k$
x	x
k	k

A running example

$$g(x+k) = a \wedge s=g(k) \wedge x=0 \wedge f(g(x+k))=f(a) \models_x s = a$$

processing $x = 0$

terms	representatives
a	$g(x + k)$
s	$g(k)$
$g(k)$	$g(k)$
$g(x + k)$	$g(x + k)$
$x + k$	$x + k$
x	x
k	k

A running example

$$g(x+k) = a \wedge s=g(k) \wedge x=0 \wedge f(g(x+k))=f(a) \models_x s = a$$

$$\text{solve}(x, 0) = [x \mapsto 0]$$

terms	representatives
a	$g(x + k)$
s	$g(k)$
$g(k)$	$g(k)$
$g(x + k)$	$g(x + k)$
$x + k$	$x + k$
x	x
k	k

A running example

$$g(x+k) = a \wedge s=g(k) \wedge x=0 \wedge f(g(x+k))=f(a) \models_x s = a$$

applying the substitution

terms	representatives
a	$g(x + k)$
s	$g(k)$
$g(k)$	$g(k)$
$g(x + k)$	$g(x + k)$
$x + k$	$0 + k$
x	0
k	k

A running example

$$g(x+k) = a \wedge s=g(k) \wedge x=0 \wedge f(g(x+k))=f(a) \models_x s = a$$

canonizing the representatives

terms	representatives
a	$g(x + k)$
s	$g(k)$
$g(k)$	$g(k)$
$g(x + k)$	$g(x + k)$
$x + k$	k
x	0
k	k

A running example

$$g(x+k) = a \wedge s=g(k) \wedge x=0 \wedge f(g(x+k))=f(a) \models_x s = a$$

finding new equations

terms	representatives
a	$g(x + k)$
s	$g(k)$
$g(k)$	$g(k)$
$g(x + k)$	$g(x + k)$
$x + k$	k
x	0
k	k

A running example

$$g(x+k) = a \wedge s=g(k) \wedge x=0 \wedge f(g(x+k))=f(a) \models_x s = a$$

terms	representatives
a	$g(k)$
s	$g(k)$
$g(k)$	$g(k)$
$g(x+k)$	$g(k)$
$x+k$	k
x	0
k	k

A running example

$$g(x+k) = a \wedge s=g(k) \wedge x=0 \wedge f(g(x+k))=f(a) \models_x s = a$$

processing $f(g(x + k)) = f(a)$

terms	representatives
a	$g(k)$
s	$g(k)$
$g(k)$	$g(k)$
$g(x + k)$	$g(k)$
$x + k$	k
x	0
k	k

A running example

$$g(x+k) = a \wedge s=g(k) \wedge x=0 \wedge f(g(x+k))=f(a) \models_x s = a$$

global canonization $f(g(k)) = f(g(k))$

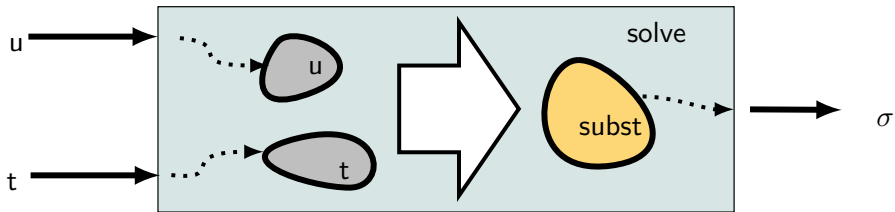
terms	representatives
a	$g(k)$
s	$g(k)$
$g(k)$	$g(k)$
$g(x+k)$	$g(k)$
$x+k$	k
x	0
k	k

Implementing Shostak's algorithm

- representatives are **terms**
- solvers take terms as arguments and return term substitutions
- global canonization traverses the term structure

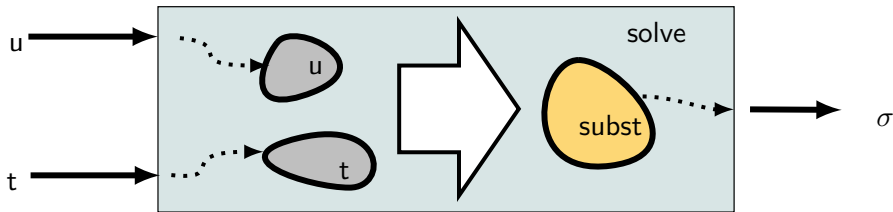
Implementing solvers

from an implementation point of view



Implementing solvers

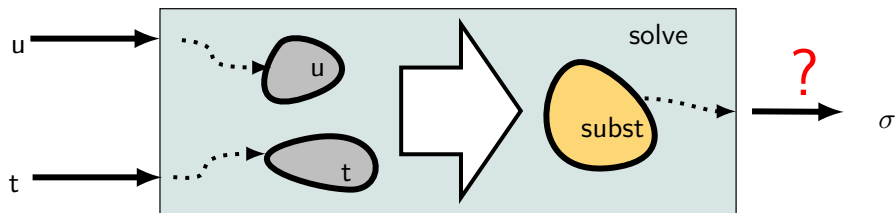
from an implementation point of view



- it is easier and more efficient to solve equations modulo X using **specific data structures**

Implementing solvers

from an implementation point of view



- it is easier and more efficient to solve equations modulo X using **specific data structures**
- returning solutions as term substitutions from these data structures is not immediate

... a Shostak like algorithm where

- representatives are **abstract values**
- solvers take abstract values as arguments and return **abstract substitutions** (from terms to abstract values)
- global canonization is skipped

furthermore,

- a special mechanism to recover incrementality
- a fine grained search of new equations

$$\bigwedge_{i \in I} u_i = t_i \models_X u \stackrel{?}{=} v$$

configurations are $\langle \Phi \mid \Delta \mid \Gamma \rangle$

- Φ is a sequence of equations or queries
- Δ is a map from terms to abstract values
- Γ is a dictionary “*is used by*” mapping abstract values to sets of terms

the algorithm starts in the configuration

$$\langle u_1 = v_1; \dots; u_n = v_n; u \stackrel{?}{=} v \mid \Delta_0 \mid \emptyset \rangle$$

with $\Delta_0(t) = [t]$, where $[t]$ is the abstract value for the term t

$$\text{REMOVE} \quad \frac{\langle \{a = b\} \uplus \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi \mid \Delta \mid \Gamma \rangle} \Delta(a) = \Delta(b)$$

$$\text{QUERY} \quad \frac{\langle a \stackrel{?}{=} b; \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi \mid \Delta \mid \Gamma \rangle} \Delta(a) = \Delta(b)$$

$$\text{CONGR} \quad \frac{\langle \{a = b\} \uplus \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi' \cup \Phi \mid \Delta' \mid \Gamma' \rangle} \quad \Delta(a) \neq \Delta(b)$$

$$\text{CONGR} \quad \frac{\langle \{a = b\} \uplus \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi' \cup \Phi \mid \Delta' \mid \Gamma' \rangle} \quad \Delta(a) \neq \Delta(b)$$

- $\sigma = \text{solve}(\Delta(a), \Delta(b))$

$$\text{CONGR} \quad \frac{\langle \{a = b\} \uplus \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi' \cup \Phi \mid \Delta' \mid \Gamma' \rangle} \quad \Delta(a) \neq \Delta(b)$$

- $\sigma = \text{solve}(\Delta(a), \Delta(b))$
- $\Delta'(t) = \Delta(t)\sigma$

$$\text{CONGR} \quad \frac{\langle \{a = b\} \uplus \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi' \cup \Phi \mid \Delta' \mid \Gamma' \rangle} \quad \Delta(a) \neq \Delta(b)$$

- $\sigma = \text{solve}(\Delta(a), \Delta(b))$
- $\Delta'(t) = \Delta(t)\sigma$
- Φ' are the new equations obtained by congruence, i.e. $f(\vec{x}) = f(\vec{y})$ such that $\Delta'(\vec{x}) = \Delta'(\vec{y})$

$$\text{CONGR} \quad \frac{\langle \{a = b\} \uplus \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi' \cup \Phi \mid \Delta' \mid \Gamma' \rangle} \quad \Delta(a) \neq \Delta(b)$$

- $\sigma = \text{solve}(\Delta(a), \Delta(b))$
- $\Delta'(t) = \Delta(t)\sigma$
- Φ' are the new equations obtained by congruence, i.e. $f(\vec{x}) = f(\vec{y})$ such that $\Delta'(\vec{x}) = \Delta'(\vec{y})$
- Γ' is a new dictionary

- to compensate the abstract nature of representatives, a function **leaves** returns the set of maximal uninterpreted terms an abstract value consists of

- to compensate the abstract nature of representatives, a function **leaves** returns the set of maximal uninterpreted terms an abstract value consists of
- dictionary invariant

$$f(t_1, \dots, t_n) \in \Gamma(\Delta(t)) \text{ if } \exists i. t \in \text{leaves}(\Delta(t_i))$$

e.g. $f(a+b, c)$ “uses” a , b and c

Variable abstraction

- to compensate the abstract nature of representatives, a function **leaves** returns the set of maximal uninterpreted terms an abstract value consists of
- dictionary invariant

$$f(t_1, \dots, t_n) \in \Gamma(\Delta(t)) \text{ if } \exists i. t \in \text{leaves}(\Delta(t_i))$$

e.g. $f(a+b, c)$ “uses” a , b and c

Γ' is the following dictionary:

$$\begin{aligned} \Gamma'(\Delta(l)) &= \Gamma(\Delta(l)) \cup \Gamma([x]) && \text{for any } (x, r) \in \sigma \text{ and } l \in \text{leaves}(r) \\ \Gamma'(r) &= \Gamma(r) && \text{otherwise} \end{aligned}$$

Finding new equations

$$g(x + k) = a \wedge s = g(k) \wedge x = 0 \models_x s = a$$

terms	abstract representatives
$g(k)$	$g(k)$
$g(x + k)$	$g(x+k)$
$x + k$	$x + k$
x	x
k	k

Finding new equations

$$g(x + k) = a \wedge s = g(k) \wedge x = 0 \models_x s = a$$

terms	abstract representatives
$g(k)$	$g(k)$
$g(x + k)$	$g(x+k)$
$x + k$	$x + k$
x	x
k	k

Finding new equations

$$g(x + k) = a \wedge s = g(k) \wedge x = 0 \models_x s = a$$

terms	abstract representatives
$g(k)$	$g(k)$
$g(x + k)$	$g(x+k)$
$x + k$	$0 + k$
x	0
k	k

Finding new equations

$$g(x + k) = a \wedge s = g(k) \wedge x = 0 \models_x s = a$$

terms	abstract representatives
$g(k)$	$g(k)$
$g(x + k)$	$g(x+k)$
$x + k$	k
x	0
k	k

Finding new equations

$$g(x + k) = a \wedge s = g(k) \wedge x = 0 \models_x s = a$$

terms	abstract representatives
$g(k)$	$g(k)$
$g(x + k)$	$g(x+k)$
$x + k$	k
x	0
k	k

Finding new equations

$$g(x + k) = a \wedge s = g(k) \wedge x = 0 \models_x s = a$$

terms	abstract representatives
$g(k)$	$g(k)$
$g(x + k)$	$g(x+k)$
$x + k$	k
x	0
k	k

Finding new equations

$$g(x + k) = a \wedge s = g(k) \wedge x = 0 \models_x s = a$$

terms	abstract representatives
$g(k)$	$g(k)$
$g(x + k)$	$g(x+k)$
$x + k$	k
x	0
k	k

$$\Phi' = \left\{ f(\vec{u}) = f(\vec{v}) \mid \begin{array}{l} f(\vec{u}) \in \Gamma([x]) \\ f(\vec{v}) \in \Gamma([x]) \cup \bigcup_{t \mid x \in \mathcal{L}(\Delta, t)} \bigcup_{l \in \mathcal{L}(\Delta', t)} \Gamma(\Delta'(l)) \\ \Delta'(\vec{u}) \equiv \Delta'(\vec{v}) \end{array} \right\}$$

Incrementality

$$a = b \wedge f(a) = u \wedge f(b) = t \models_{\emptyset} u = t$$

Incrementality

$$a = b \wedge f(a) = u \wedge f(b) = t \models_{\emptyset} u = t$$

$$\text{ADD} \quad \frac{\langle C[f(\vec{a})]; \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi'; C[f(\vec{a})]; \Phi \mid \Delta \mid \Gamma' \rangle} \quad f(\vec{a}) \notin \Delta, \vec{a} \in \Delta$$

$$a = b \wedge f(a) = u \wedge f(b) = t \models_{\emptyset} u = t$$

$$\text{ADD} \quad \frac{\langle C[f(\vec{a})]; \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi'; C[f(\vec{a})]; \Phi \mid \Delta \mid \Gamma' \rangle} \quad f(\vec{a}) \notin \Delta, \vec{a} \in \Delta$$

- Γ' is the following dictionary:

$$\Gamma'([f(\vec{a})]) = \emptyset$$

$$\Gamma'(\Delta(l)) = \Gamma(\Delta(l)) \cup \{f(\vec{a})\} \quad \text{for any } l \in \text{leaves}(\Delta(\vec{a}))$$

$$\Gamma'(r) = \Gamma(r) \quad \text{otherwise}$$

$$a = b \wedge f(a) = u \wedge f(b) = t \models_{\emptyset} u = t$$

$$\text{ADD} \quad \frac{\langle C[f(\vec{a})]; \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi'; C[f(\vec{a})]; \Phi \mid \Delta \mid \Gamma' \rangle} \quad f(\vec{a}) \notin \Delta, \vec{a} \in \Delta$$

- Γ' is the following dictionary:

$$\Gamma'([f(\vec{a})]) = \emptyset$$

$$\Gamma'(\Delta(l)) = \Gamma(\Delta(l)) \cup \{f(\vec{a})\} \quad \text{for any } l \in \text{leaves}(\Delta(\vec{a}))$$

$$\Gamma'(r) = \Gamma(r) \quad \text{otherwise}$$

- Φ' are the new equations $f(\vec{a}) = f(\vec{b})$ such that

$$\exists l \in \text{leaves}(\Delta(\vec{a})). f(\vec{b}) \in \Gamma'(\Delta(l)) \text{ and } \Delta(\vec{a}) = \Delta(\vec{b})$$

Implementation

an implementation of CC(X) is at the heart of the Ergo automated theorem prover (<http://ergo.lri.fr>)

- purely functional data structures
- directly follows the inference rules presented so far
- \sim 500 lines of Ocaml code (Ergo is \sim 3000 loc)

benchmark (1349 VC from 61 C programs, timeout after 20s, 3.20 GH processor, 2 Gb of memory)

	valid	timeout	unknown
Simplify	98%	1%	1%
Yices	95%	2%	3%
Ergo	95%	4%	1%
CVC-Lite	67%	30%	3%

- $CC(X)$ efficiently combines equality with a solvable theory
- its main novelty rests on the use of abstract values for representatives and a fine grained search for new equations by congruence

since practice often arrives before theory, formalizing:

- $CC(X_1, \dots, X_n)$ to combine several solvable theories
- the treatment of predicates
- $CC(X)$ in Coq, proving its soundness and interpreting traces from ergo