# 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



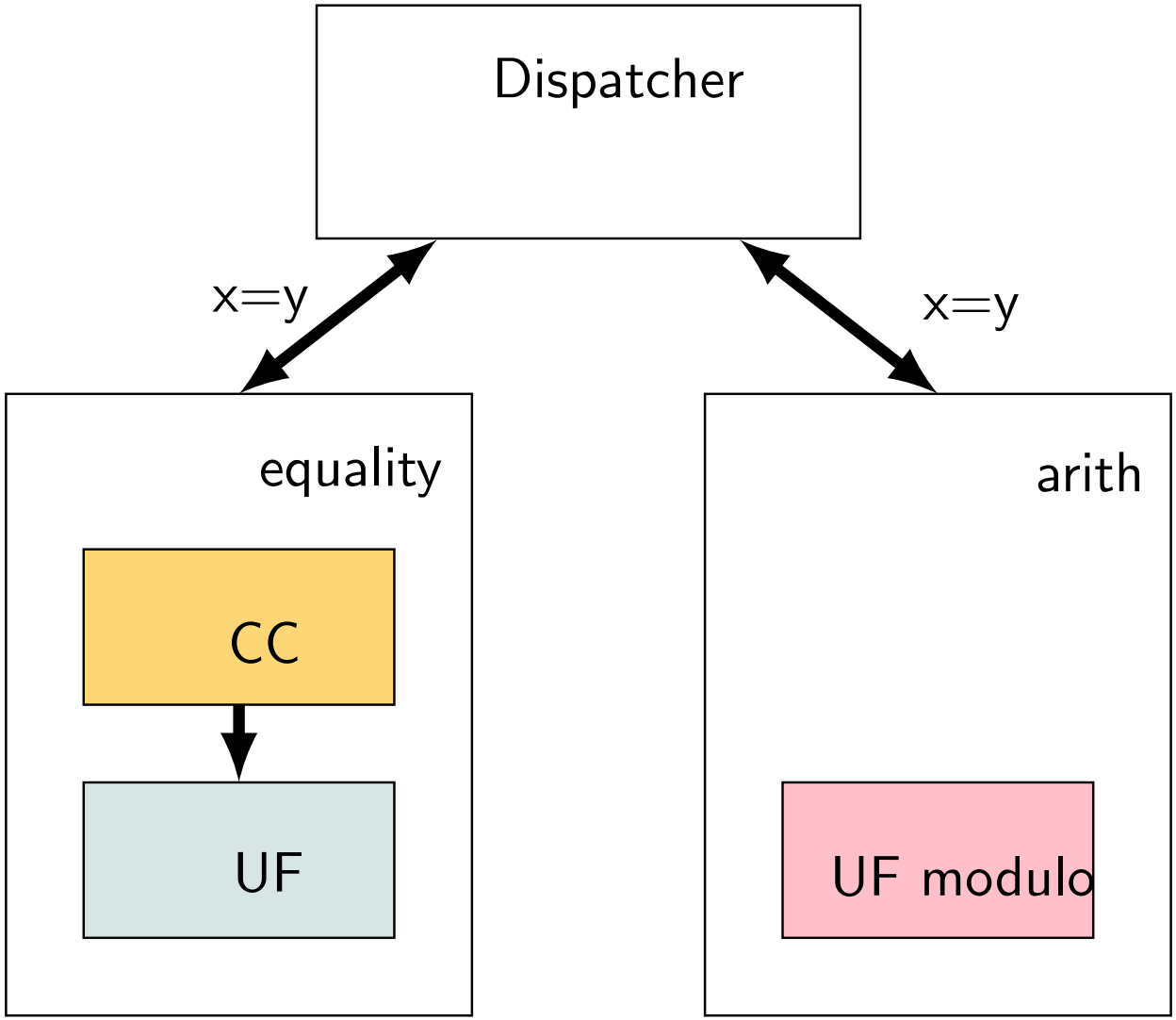
# The ProVal Project

- 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`

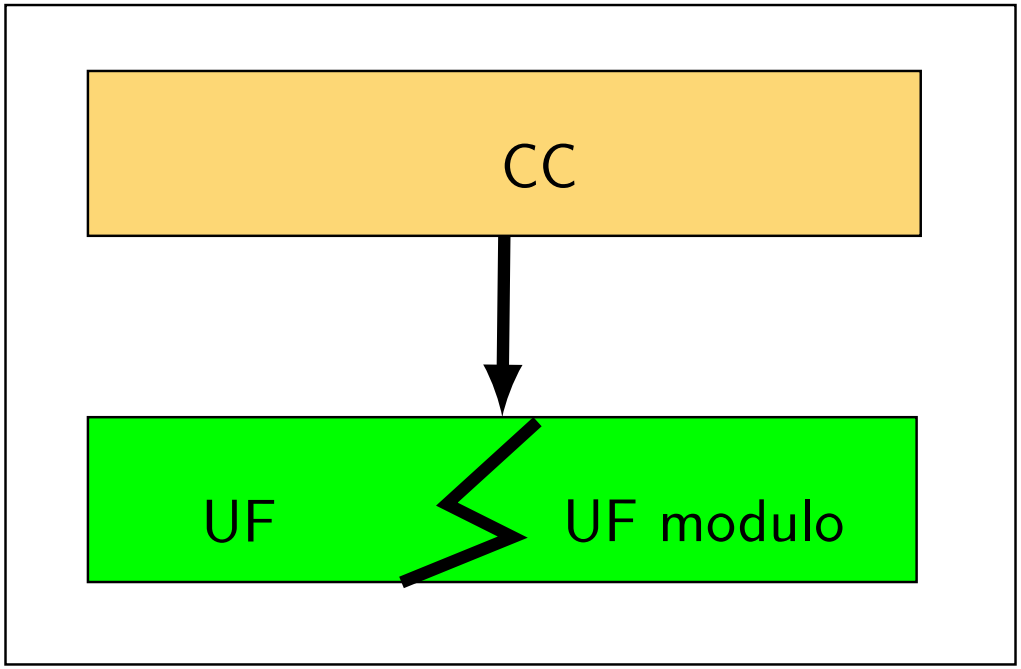
# Combining equality with a theory X

Nelson-Oppen architecture



# Combining equality with a theory X

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 $\mathtt{u} = \mathtt{t}$, the main steps of the algorithm are:

1. apply global canonization to $\mathtt{u}$ and $\mathtt{t}$
2. $\mathtt{solve}(\bar{\mathtt{u}}, \bar{\mathtt{t}})$ returns a substitution $\sigma$, the solution of $\bar{\mathtt{u}} = \bar{\mathtt{t}}$
3. $\sigma$ 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

$$\mathtt{g(x+k)} = \mathtt{a} \wedge \mathtt{s=g(k)} \wedge \mathtt{x=0} \wedge \mathtt{f(g(x+k))=f(a)} \models_X \mathtt{s} = \mathtt{a}$$

| terms | representatives |
|---|---|
| $\mathtt{a}$ | $\mathtt{g(x+k)}$ |
| $\mathtt{s}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(k)}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(x+k)}$ | $\mathtt{g(x+k)}$ |
| $\mathtt{x+k}$ | $\mathtt{x+k}$ |
| $\mathtt{x}$ | $\mathtt{x}$ |
| $\mathtt{k}$ | $\mathtt{k}$ |

# A running example

$$\texttt{g(x+k) = a} \wedge \texttt{s=g(k)} \wedge \texttt{x=0} \wedge \texttt{f(g(x+k))=f(a)} \models_X \texttt{s = a}$$

processing $\mathtt{x} = 0$

| terms | representatives |
|---|---|
| $\mathtt{a}$ | $\mathtt{g}(\mathtt{x}+\mathtt{k})$ |
| $\mathtt{s}$ | $\mathtt{g}(\mathtt{k})$ |
| $\mathtt{g}(\mathtt{k})$ | $\mathtt{g}(\mathtt{k})$ |
| $\mathtt{g}(\mathtt{x}+\mathtt{k})$ | $\mathtt{g}(\mathtt{x}+\mathtt{k})$ |
| $\mathtt{x}+\mathtt{k}$ | $\mathtt{x}+\mathtt{k}$ |
| $\mathtt{x}$ | $\mathtt{x}$ |
| $\mathtt{k}$ | $\mathtt{k}$ |

# A running example

$$\texttt{g(x+k)} = \texttt{a} \wedge \texttt{s=g(k)} \wedge \texttt{x=0} \wedge \texttt{f(g(x+k))=f(a)} \models_X \texttt{s} = \texttt{a}$$

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

| terms | representatives |
|---|---|
| $\texttt{a}$ | $\texttt{g(x + k)}$ |
| $\texttt{s}$ | $\texttt{g(k)}$ |
| $\texttt{g(k)}$ | $\texttt{g(k)}$ |
| $\texttt{g(x + k)}$ | $\texttt{g(x + k)}$ |
| $\texttt{x + k}$ | $\texttt{x + k}$ |
| $\texttt{x}$ | $\texttt{x}$ |
| $\texttt{k}$ | $\texttt{k}$ |

# A running example

$$\texttt{g(x+k)} = \texttt{a} \wedge \texttt{s=g(k)} \wedge \texttt{x=0} \wedge \texttt{f(g(x+k))=f(a)} \models_X \texttt{s} = \texttt{a}$$

applying the substitution

| terms | representatives |
|---|---|
| $\texttt{a}$ | $\texttt{g}(\texttt{x}+\texttt{k})$ |
| $\texttt{s}$ | $\texttt{g}(\texttt{k})$ |
| $\texttt{g}(\texttt{k})$ | $\texttt{g}(\texttt{k})$ |
| $\texttt{g}(\texttt{x}+\texttt{k})$ | $\texttt{g}(\texttt{x}+\texttt{k})$ |
| $\texttt{x}+\texttt{k}$ | $0+\texttt{k}$ |
| $\texttt{x}$ | $0$ |
| $\texttt{k}$ | $\texttt{k}$ |

# A running example

$$\texttt{g(x+k) = a} \wedge \texttt{s=g(k)} \wedge \texttt{x=0} \wedge \texttt{f(g(x+k))=f(a)} \models_X \texttt{s = a}$$

canonizing the representatives

| terms | representatives |
|---|---|
| $\texttt{a}$ | $\texttt{g(x + k)}$ |
| $\texttt{s}$ | $\texttt{g(k)}$ |
| $\texttt{g(k)}$ | $\texttt{g(k)}$ |
| $\texttt{g(x + k)}$ | $\texttt{g(x + k)}$ |
| $\texttt{x + k}$ | $\texttt{k}$ |
| $\texttt{x}$ | $0$ |
| $\texttt{k}$ | $\texttt{k}$ |

# A running example

$$\mathtt{g(x+k)} = \mathtt{a} \wedge \mathtt{s=g(k)} \wedge \mathtt{x=0} \wedge \mathtt{f(g(x+k))=f(a)} \models_X \mathtt{s} = \mathtt{a}$$

finding new equations

| terms | representatives |
|---|---|
| $\mathtt{a}$ | $\mathtt{g(x+k)}$ |
| $\mathtt{s}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(k)}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(x+k)}$ | $\mathtt{g(x+k)}$ |
| $\mathtt{x+k}$ | $\mathtt{k}$ |
| $\mathtt{x}$ | $0$ |
| $\mathtt{k}$ | $\mathtt{k}$ |

# A running example

$$\mathtt{g(x+k)} = \mathtt{a} \wedge \mathtt{s=g(k)} \wedge \mathtt{x=0} \wedge \mathtt{f(g(x+k))=f(a)} \models_X \mathtt{s} = \mathtt{a}$$

| terms | representatives |
|---|---|
| $\mathtt{a}$ | $\mathtt{g(k)}$ |
| $\mathtt{s}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(k)}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(x+k)}$ | $\mathtt{g(k)}$ |
| $\mathtt{x+k}$ | $\mathtt{k}$ |
| $\mathtt{x}$ | $0$ |
| $\mathtt{k}$ | $\mathtt{k}$ |

# A running example

$$\mathtt{g(x+k)} = \mathtt{a} \wedge \mathtt{s=g(k)} \wedge \mathtt{x=0} \wedge \mathtt{f(g(x+k))=f(a)} \models_X \mathtt{s} = \mathtt{a}$$

processing $\mathtt{f(g(x+k))} = \mathtt{f(a)}$

| terms | representatives |
|---|---|
| $\mathtt{a}$ | $\mathtt{g(k)}$ |
| $\mathtt{s}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(k)}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(x+k)}$ | $\mathtt{g(k)}$ |
| $\mathtt{x+k}$ | $\mathtt{k}$ |
| $\mathtt{x}$ | $0$ |
| $\mathtt{k}$ | $\mathtt{k}$ |

# A running example

$$\mathtt{g(x+k)} = \mathtt{a} \wedge \mathtt{s=g(k)} \wedge \mathtt{x=0} \wedge \mathtt{f(g(x+k))=f(a)} \models_X \mathtt{s} = \mathtt{a}$$

global canonization $\mathtt{f(g(k))} = \mathtt{f(g(k))}$

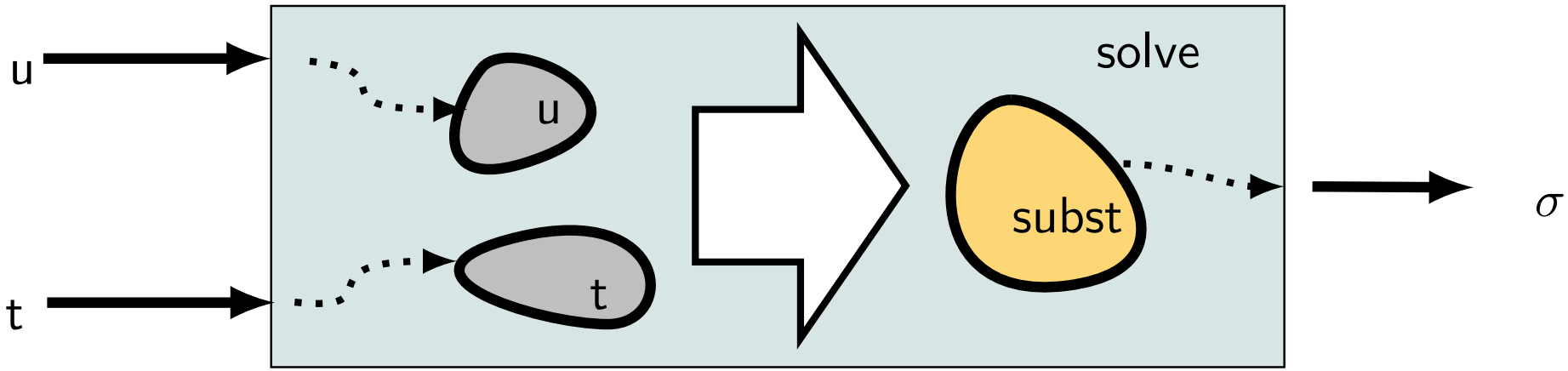
| terms | representatives |
|---|---|
| $\mathtt{a}$ | $\mathtt{g(k)}$ |
| $\mathtt{s}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(k)}$ | $\mathtt{g(k)}$ |
| $\mathtt{g(x+k)}$ | $\mathtt{g(k)}$ |
| $\mathtt{x+k}$ | $\mathtt{k}$ |
| $\mathtt{x}$ | $0$ |
| $\mathtt{k}$ | $\mathtt{k}$ |

# Implementing Shostak's algorithm

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

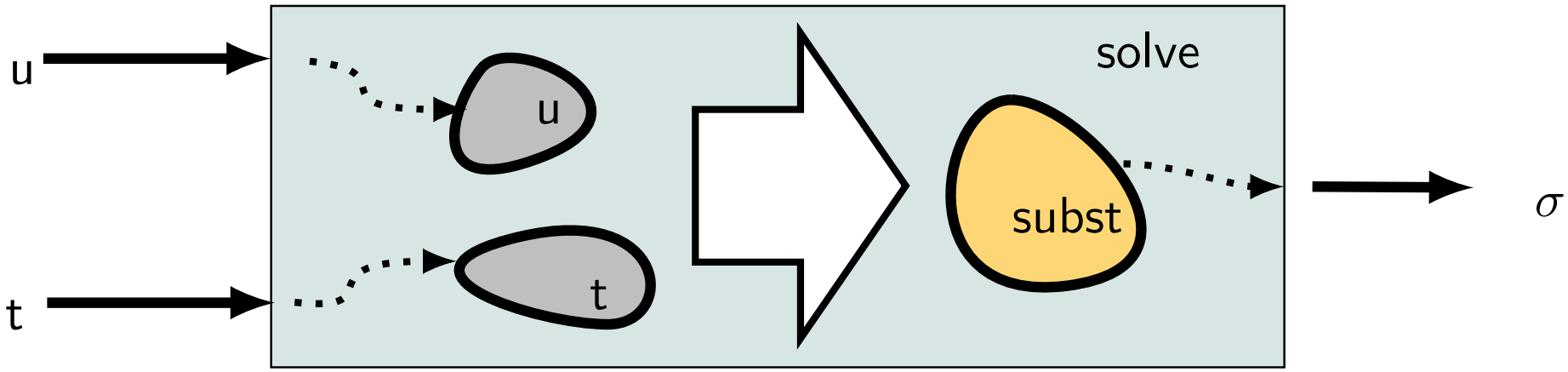
# Implementing solvers

from an implementation point of view



# Implementing solvers

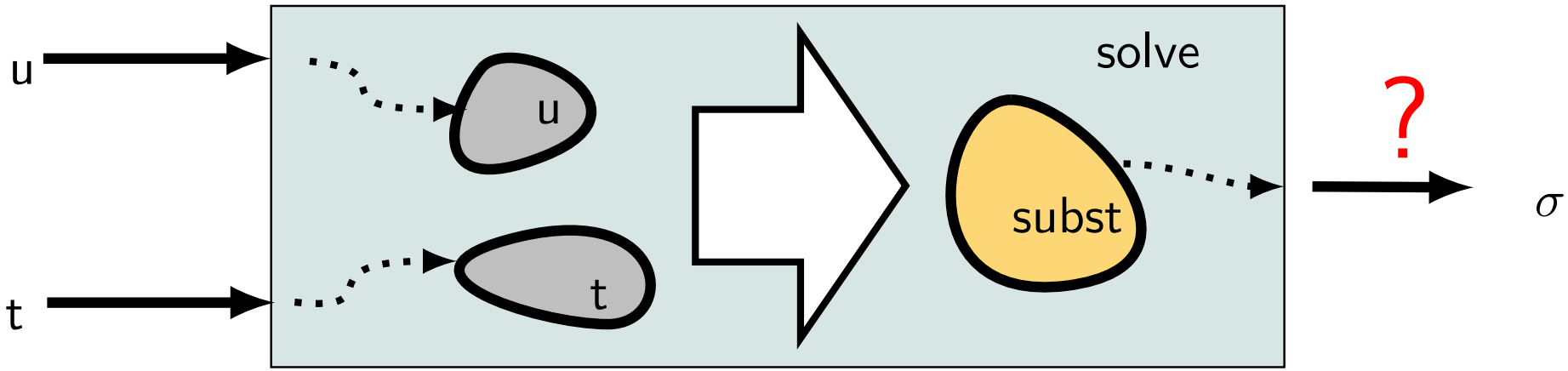
from an implementation point of view




- 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

# CC(X)

. . . 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

# Configurations

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

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

- $\Phi$ is a sequence of equations or queries
- $\Delta$ is a map from terms to abstract values
- $\Gamma$ is a dictionary *"is used by"* mapping abstract values to sets of terms

the algorithm starts in the configuration

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

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

# CC(X) inference rules

$$\textsc{Remove} \quad \frac{\langle \{a = b\} \uplus \Phi \mid \Delta \mid \Gamma \rangle}{\langle \Phi \mid \Delta \mid \Gamma \rangle} \; \Delta(a) = \Delta(b)$$

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

# CC(X) inference rules

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

# CC(X) inference rules

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

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

# CC(X) inference rules

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

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

# CC(X) inference rules

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

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

# CC(X) inference rules

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

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

# Variable abstraction

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

# 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

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

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

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

e.g. `f(a+b,c)` "uses" `a`, `b` and `c`

$\Gamma'$ is the following dictionary:

$$\Gamma'(\Delta(l)) = \Gamma(\Delta(l)) \cup \Gamma([x]) \quad \text{for any } (x, r) \in \sigma \text{ and } l \in \mathtt{leaves}(r)$$

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

# 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 |

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

# Incrementality

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

# Incrementality

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

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

# Incrementality

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

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

- $\Gamma'$ is the following dictionary:

$$\Gamma'([f(\vec{a})]) = \emptyset$$
$$\Gamma'(\Delta(l)) = \Gamma(\Delta(l)) \cup \{f(\vec{a})\} \quad \text{for any } l \in \texttt{leaves}(\Delta(\vec{a}))$$
$$\Gamma'(r) = \Gamma(r) \quad \text{otherwise}$$

# Incrementality

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

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

- $\Gamma'$ is the following dictionary:

$$\begin{aligned} &\Gamma'([f(\vec{a})]) = \emptyset \\ &\Gamma'(\Delta(l)) = \Gamma(\Delta(l)) \cup \{f(\vec{a})\} \quad \text{for any } l \in \texttt{leaves}(\Delta(\vec{a})) \\ &\Gamma'(r) = \Gamma(r) \qquad\qquad\qquad\qquad \text{otherwise} \end{aligned}$$

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

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

# 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% |

# Conclusion

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