SMT and Z3

Nikolaj Bjørner
Microsoft Research
ReRISE Winter School, Linz, Austria
February 5, 2014

Plan

Mon	An invitation to SMT with Z3
Tue	Equalities and Theory Combination
Wed	Theories: Arithmetic, Arrays, Data types
Thu	Quantifiers and Theories
Fri	Programming Z3: Interfacing and Solving

Quiz

Show: A *difference logic graph* without negative cycles has a model. Give a procedure for extracting a model.

True or false: A formula over difference logic has a model over reals **iff** it has a model over integers?

Give an *efficient* algorithm to extract models for UTVPI over integers.

Encode lambda Calculus into $map, K, read$ (without I).

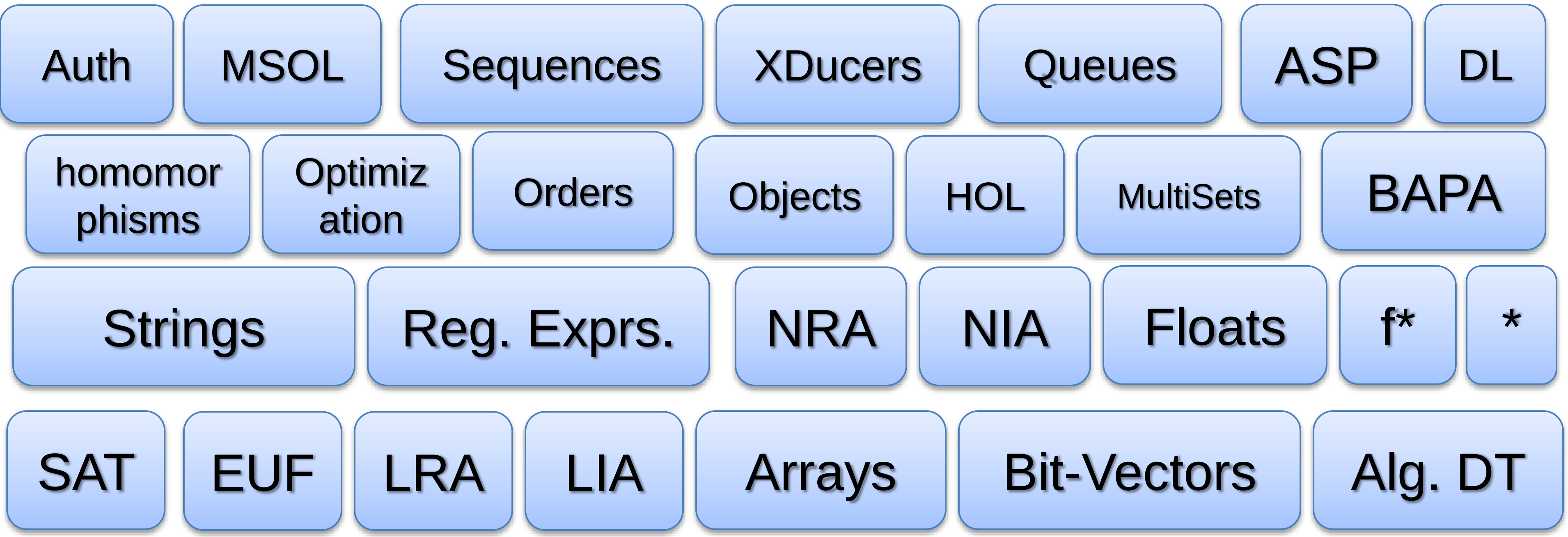
Plan

- Arithmetic
- Arrays and friends
- Data types [Introduction]

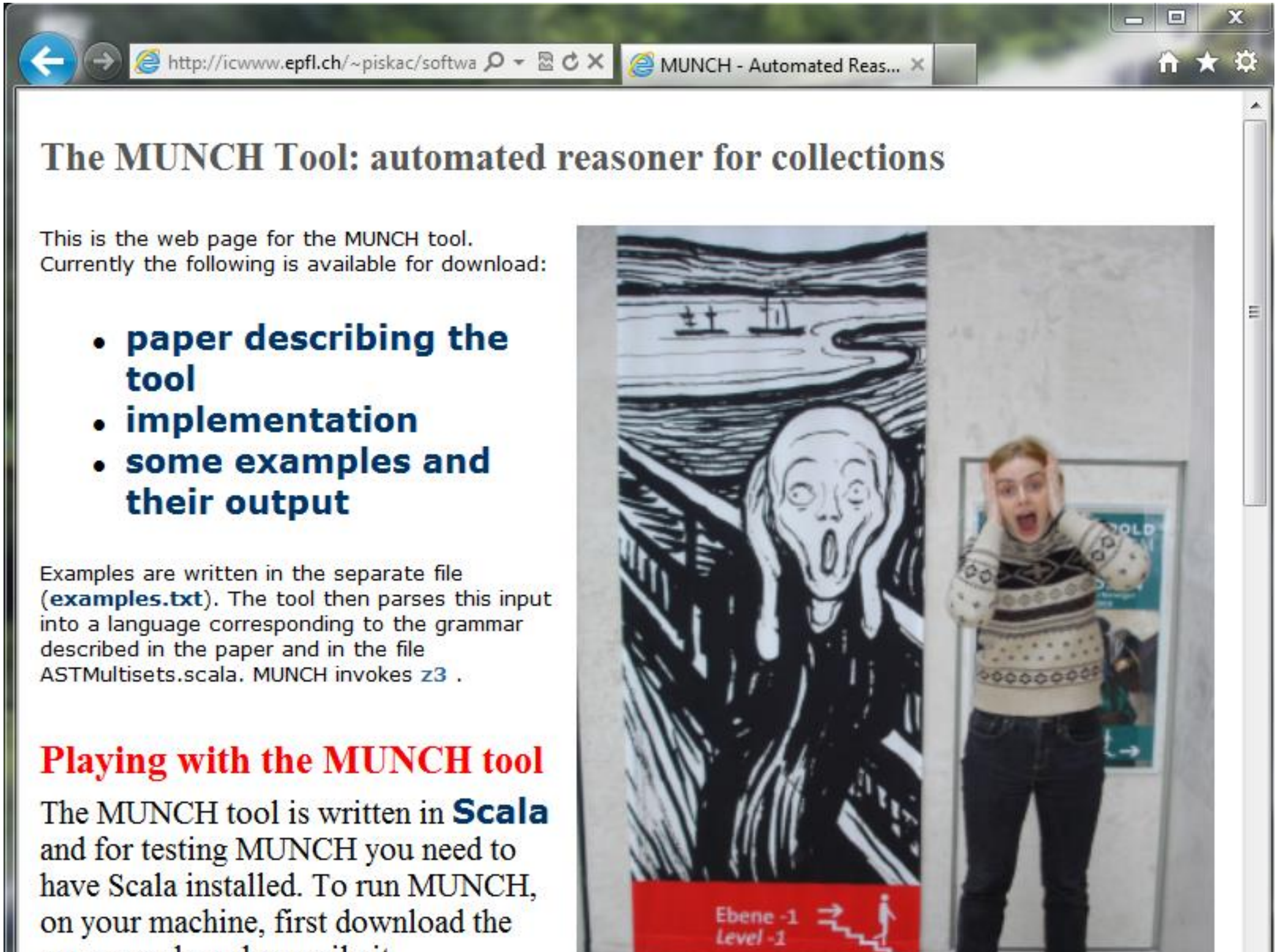
What Theories?

Overall aim:

Rich Theories *(and logics) with*
Efficient *Decision Procedures*



Be afraid!


http://icwww.epfl.ch/~piskac/softwa
MUNCH - Automated Reas...

The MUNCH Tool: automated reasoner for collections

This is the web page for the MUNCH tool. Currently the following is available for download:

- **paper describing the tool**
- **implementation**
- **some examples and their output**

Examples are written in the separate file (**examples.txt**). The tool then parses this input into a language corresponding to the grammar described in the paper and in the file ASTMultisets.scala. MUNCH invokes z3 .

Playing with the MUNCH tool

The MUNCH tool is written in **Scala** and for testing MUNCH you need to have Scala installed. To run MUNCH, on your machine, first download the

Linear Real Arithmetic

- Many approaches
  - Graph-based for difference logic: $a - b \leq 3$
  - Fourier-Motzkin elimination:

$$t_1 \leq ax,\ \ bx \leq t_2 \ \ \Rightarrow \ \ bt_1 \leq at_2$$

  - Standard Simplex
  - General Form Simplex
  - GDPLL [McMillan],
    Unate Resolution [Coton],
    Conflict Resolution [Korovin et.al.]

Difference Logic: $a - b \leq 5$

Very useful in practice!

Most arithmetical constraints in software verification/analysis are in this fragment.

$$x := x + 1$$

$$\Downarrow$$

$$x_1 = x_0 + 1$$

$$\Downarrow$$

$$x_1 - x_0 \leq 1, \; x_0 - x_1 \leq -1$$

Job shop scheduling

$d_{i,j}$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

$max = 8$

Solution

$t_{1,1} = 5,\ t_{1,2} = 7,\ t_{2,1} = 2,$
$t_{2,2} = 6,\ t_{3,1} = 0,\ t_{3,2} = 3$

Encoding

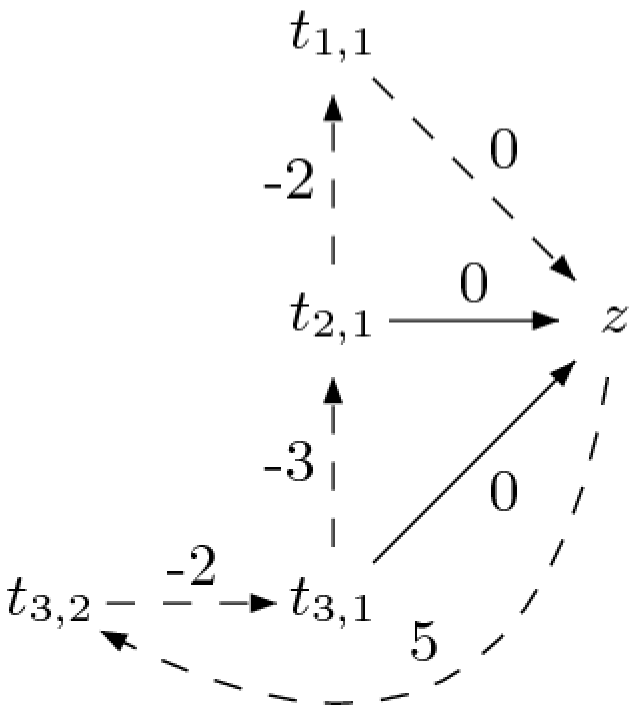
$$\begin{aligned}
&(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge \\
&(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge \\
&(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge \\
&((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2)) \wedge \\
&((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2)) \wedge \\
&((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3)) \wedge \\
&((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1)) \wedge \\
&((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1)) \wedge \\
&((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))
\end{aligned}$$

Difference Logic

Chasing negative cycles!

Algorithms based on Bellman-Ford (O(mn)).

$$\begin{array}{lcccl}
z & - & t_{1,1} & \leq & 0 \\
z & - & t_{2,1} & \leq & 0 \\
z & - & t_{3,1} & \leq & 0 \\
t_{3,2} & - & z & \leq & 5 \\
t_{3,1} & - & t_{3,2} & \leq & -2 \\
t_{2,1} & - & t_{3,1} & \leq & -3 \\
t_{1,1} & - & t_{2,1} & \leq & -2
\end{array}$$



Unit Two Variables Per Inequality

$$x + y \leq 5 \wedge -x + y \leq -4 \wedge y + y \geq 1$$

Unit Two Variables Per Inequality

$$x + y \leq 5 \land -x + y \leq -4 \land 2y \geq 1$$

$$2y \leq 1 \land 2y \geq 1$$

Unit Two Variables Per Inequality

$$x + y \leq 5 \wedge -x + y \leq -4 \wedge 2y \geq 1$$

$$2y \leq 1 \wedge 2y \geq 1$$

$$y \leq 0 \wedge y \geq 1$$

Unit Two Variables Per Inequality: UTVPI

Reduce to Difference Logic:

- For every variable x introduce fresh variables x^+, x^-
- Meaning: $2x \coloneqq x^+ - x^-$
- Rewrite constraints as follows:

- $x - y \leq k \quad \Rightarrow \begin{cases} x^+ - y^+ \leq k \\ y^- - x^- \leq k \end{cases}$

UTVPI

- $x - y \leq k \quad \Rightarrow \begin{cases} x^{+} - y^{+} \leq k \\ y^{-} - x^{-} \leq k \end{cases}$

- $x \leq k \quad \Rightarrow x^{+} - x^{-} \leq 2k$

- $x + y \leq k \quad \Rightarrow \begin{cases} x^{+} - y^{-} \leq k \\ y^{+} - x^{-} \leq k \end{cases}$
- $x + y \leq k \quad \Rightarrow$ chalkboard

UTVPI

$$x + y \leq 5 \land -x + y \leq -4 \land 2y \geq 1$$

$$x^+ - y^- \leq 5 \land y^+ - x^- \leq 5 \land$$

$$-x^+ + y^+ \leq -4 \land x^- - y^- \leq -4 \land$$

$$y^- - y^+ \leq 1$$

UTVPI

- Solve for x^+ and x^-

- $M(x) \coloneqq (M(x^+) - M(x^-))/2$

- Nothing can go wrong…

$$2y \leq 1 \land 2y \geq 1$$

UTVPI

- $M(x) \coloneqq (M(x^+) - M(x^-))/2$
- Nothing can go wrong… as if
- What if:
  - x is an integer
  - $M(x^+)$ is **odd** and
  - $M(x^-)$ is **even**

- **Thm**: Parity can be fixed **iff** there is no tight loop forcing the wrong parity

UTVPI

$$\begin{aligned} x^- - y^+ &\leq 5 \\ y^+ - z^- &\leq -6 \\ z^- - x^+ &\leq -2 \\ x^+ - v^+ &\leq 3 \\ v^+ - x^- &\leq 0 \end{aligned}$$

$$\Rightarrow \quad \begin{aligned} x^- - x^+ &\leq -3 \\ x^+ - x^- &\leq 3 \end{aligned}$$

General Form

General Form: $Ax = 0$ and $l_j \leq x_j \leq u_j$

Example:

$$x \geq 0, (x + y \leq 2 \vee x + 2y \geq 6), (x + y = 2 \vee x + 2y > 4)$$

$$\leadsto$$

$$s_1 \equiv x + y, s_2 \equiv x + 2y,$$

$$x \geq 0, (s_1 \leq 2 \vee s_2 \geq 6), (s_1 = 2 \vee s_2 > 4)$$

Only bounds (e.g., $s_1 \leq 2$) are asserted during the search.

Unconstrained variables can be eliminated before the beginning of the search.

From Definitions to a Tableau

$s_1 \equiv x + y, \quad s_2 \equiv x + 2y$

From Definitions to a Tableau

$$s_1 \equiv x + y, \quad s_2 \equiv x + 2y$$

$$\Big\downarrow$$

$$s_1 = x + y,$$
$$s_2 = x + 2y$$

From Definitions to a Tableau

$s_1 \equiv x + y, \quad s_2 \equiv x + 2y$

$$\Downarrow$$

$s_1 = x + y,$

$s_2 = x + 2y$

$$\Downarrow$$

$s_1 - x - y \quad = 0$

$s_2 - x - 2y = 0$

From Definitions to a Tableau

$s_1 \equiv x + y, \quad s_2 \equiv x + 2y$

$$\Downarrow$$

$s_1 = x + y,$

$s_2 = x + 2y$

$$\Downarrow$$

$s_1 - x - y \ \ = 0$ $\qquad$ s_1, s_2 are basic (dependent)

$s_2 - x - 2y = 0$ $\qquad$ x,y are non-basic

Pivoting

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$

Pivoting

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$
$$\Downarrow$$
$$-s_1 + x + y = 0$$
$$s_2 - x - 2y = 0$$

Pivoting

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - 2s_1 + x = 0$$

Pivoting

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - 2s_1 + x = 0$$

It is just substituting equals by equals.

Definition:
An assignment (model) is a mapping from variables to values

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_1 and y

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - x - 2y = 0$$

$$\Downarrow$$

$$-s_1 + x + y = 0$$
$$s_2 - 2s_1 + x = 0$$

It is just substituting equals by equals.

Key Property:
If an assignment satisfies the equations before a pivoting step, then it will also satisfy them after!

Definition:
An assignment (model) is a mapping from variables to values

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap s_2 and y

$s_1 - x - y = 0$
$s_2 - x - 2y = 0$

$$\Downarrow$$

$-s_1 + x + y = 0$
$s_2 - x - 2y = 0$

$$\Downarrow$$

$-s_1 + x + y = 0$
$s_2 - 2s_1 + x = 0$

It is just substituting equals by equals.

Example:
$M(x) = 1$
$M(y) = 1$
$M(s_1) = 2$
$M(s_2) = 3$

Key Property:
If an assignment satisfies the equations before a pivoting step, then it will also satisfy them after!

Equations + Bounds + Assignment

An assignment (model) is a mapping from variables to values.

We maintain an assignment that satisfies all equations and bounds.

The assignment of non dependent variables implies the assignment of dependent variables.

Equations + Bounds can be used to derive new bounds.

Example: $x = y - z,\ y \leq 2,\ z \geq 3 \rightsquigarrow x \leq -1.$

The new bound may be inconsistent with the already known bounds.

Example: $x \leq -1,\ x \geq 0.$

“Repairing Models”

If the assignment of a non-basic variable does not satisfy a bound, then fix it and propagate the change to all dependent variables.

$a = c - d$
$b = c + d$
$M(a) = 0$
$M(b) = 0$
$M(c) = 0$
$M(d) = 0$
$1 \leq c$

$\Longrightarrow$

$a = c - d$
$b = c + d$
$M(a) = 1$
$M(b) = 1$
$M(c) = 1$
$M(d) = 0$
$1 \leq c$

“Repairing Models”

If the assignment of a non-basic variable does not satisfy a bound, then fix it and propagate the change to all dependent variables. Of course, we may introduce new “problems”.

$a = c - d$		$a = c - d$
$b = c + d$		$b = c + d$
$M(a) = 0$		$M(a) = 1$
$M(b) = 0$	$\Rightarrow$	$M(b) = 1$
$M(c) = 0$		$M(c) = 1$
$M(d) = 0$		$M(d) = 0$
$1 \leq c$		$1 \leq c$
$a \leq 0$		$\mathbf{a \leq 0}$

"Repairing Models"

If the assignment of a basic variable does not satisfy a bound, then pivot it, fix it, and propagate the change to its new dependent variables.

$a = c - d$		$c = a + d$		$c = a + d$
$b = c + d$		$b = a + 2d$		$b = a + 2d$
$M(a) = 0$		$M(a) = 0$		$M(a) = 1$
$M(b) = 0$	$\Rightarrow$	$M(b) = 0$	$\Rightarrow$	$M(b) = 1$
$M(c) = 0$		$M(c) = 0$		$M(c) = 1$
$M(d) = 0$		$M(d) = 0$		$M(d) = 0$
$1 \leq a$		$1 \leq a$		$1 \leq a$

"Repairing Models"

Sometimes, a model cannot be repaired. It is pointless to pivot.

$a = b - c$

$a \leq 0$, $1 \leq b$, $c \leq 0$

$M(a) = 1$

$M(b) = 1$

$M(c) = 0$

The value of M(a) is too big. We can reduce it by:

- reducing M(b)

  not possible b is at lower bound
- increasing M(c)

  not possible c is at upper bound

"Repairing Models"

Extracting proof from failed repair attempts is easy.

$s_1 \equiv a + d,\ s_2 \equiv c + d$

$a = s_1 - s_2 + c$

$a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c$

$M(a) = 1$

$M(s_1) = 1$

$M(s_2) = 0$

$M(c) = 0$

“Repairing Models”

Extracting proof from failed repair attempts is easy.

$s_1 \equiv a + d,\ s_2 \equiv c + d$

$a = s_1 - s_2 + c$

$a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c$

$M(a) = 1$

$M(s_1) = 1$

$M(s_2) = 0$

$M(c) = 0$

$\{ a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c \}$ is inconsistent

"Repairing Models"

Extracting proof from failed repair attempts is easy.

$s_1 \equiv a + d,\ s_2 \equiv c + d$

$a = s_1 - s_2 + c$

$a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c$

$M(a) = 1$

$M(s_1) = 1$

$M(s_2) = 0$

$M(c) = 0$

$\{\ a \leq 0,\ 1 \leq s_1,\ s_2 \leq 0,\ 0 \leq c\ \}$ is inconsistent

$\{\ a \leq 0,\ 1 \leq a + d,\ c + d \leq 0,\ 0 \leq c\ \}$ is inconsistent

Arrays and Combinatory Array Logic

What are arrays?

- Applicative stores:

$$write(a,i,v)[i] = v$$

$$i \neq j \Rightarrow write(a,i,v)[j] = a[j]$$

- Or, special combinator:

$$write(a,i,v) = \lambda j.ite(i = j, v, a[j])$$

What are arrays?

- Special combinator:

$$write(a,i,v) = \lambda j.ite(i = j, v, a[j])$$

- Existential fragment is decidable by reduction to congruence closure using finite set of instances.

- Models for arrays are finite maps with default values.

What else are arrays?

- Special combinators:

$$write(a,i,v) = \lambda j.ite(i = j, v, a[j])$$

$$K(v) = \lambda j.v$$

$$map_f(a,b) = \lambda j.f(a[j], b[j])$$

- **Result**: Existential fragment is decidable and in NP by reduction to congruence closure using finite set of instances.

What else are arrays++?

- Extra special combinators:

$$write(a,i,v) = \lambda j.ite(i = j, v, a[j])$$

$$K(v) = \lambda j.v$$

$$map_f(a,b) = \lambda j.f(a[j], b[j])$$

$$I = \lambda j.j$$

- Easy to encode lambda calculus

What else are arrays++?

- Encoding lambda terms into CAL+:

$$[[\lambda x.M]] = tr(x,[[M]]) \qquad tr(x,x) = I$$

$$[[x]] = x \qquad tr(x,y) = K(y)$$

$$[[(MN)]] = map_{read}([[M]],[[N]]) \qquad tr(x, f(M,N)) = map_f(tr(x,M), tr(x,N))$$

- Where

$$M, N ::= x \mid \lambda x.M \mid (MN)$$

Exercise: encode lambda calculus without *I*

NB. Our procedure is going to assume that function passed to map is not from *read.*

Example translation

$$[[\lambda x.((\lambda y.(yx))x)]]$$
$$= tr(x,[[((\lambda y.(yx))x)]])$$
$$= tr(x, map_{read}([[\lambda y.(yx)]],[[x]]))$$
$$= tr(x, map_{read}([[\lambda y.(yx)]], x))$$
$$= tr(x, map_{read}(tr(y,[[(yx)]]), x))$$
$$= tr(x, map_{read}(tr(y, map_{read}(y, x)), x))$$
$$= tr(x, map_{read}(map_{map_{read}}(tr(y, y), tr(y, x))), x))$$

$$= tr(x, map_{read}(map_{map_{read}}(I, K(x)), x))$$
$$= map_{map_{read}}(tr(x, map_{map_{read}}(I, K(x))), tr(x, x))$$
$$= map_{map_{read}}(map_{map_{map_{read}}}(tr(x, I), tr(x, K(x)))), I)$$
$$= map_{map_{read}}(map_{map_{map_{read}}}(K(I), tr(x, K(x)))), I)$$
$$= map_{map_{read}}(map_{map_{map_{read}}}(K(I), map_K(tr(x, x))), I)$$
$$= map_{map_{read}}(map_{map_{map_{read}}}(K(I), map_K(I)), I)$$

… But there are arrays#:

- Restricted theory using *I.*

$$K(v) = \lambda j.v$$

$$map_{ite}(a,b,c) = \lambda j.ite(a[j],b[j],c[j])$$

$$map_{=}(a,b) = \lambda j.(a[j] = b[j])$$

$$I = \lambda j.j$$

- Then: $write(a,i,v) = map_{ite}(map_{=}(K(i),I),K(v),a)$
- Theory of arrays# is decidable.

Last combinator for the road…

- Can I access a *default* array value?

$\delta(a) - default$

$\delta(K(v)) = v$

$\delta(map_f(a,b)) = f(\delta(a), \delta(b))$

$\delta(write(a,i,v)) = \delta(a)$ Only sound for infinite domains

Let’s use CAL:

- Simple set and bag operations:

$\emptyset$	□	$K(\mathit{false})$	$\emptyset_{Bag}$	□	$K(0)$
$\{a\}$	□	$\mathit{write}(\emptyset, a, \mathit{true})$	$\{a\}$	□	$\mathit{write}(\emptyset, a, 1)$
$a \in A$	□	$A[a]$	$\mathit{mult}(a, A)$	□	$A[a]$
$A \cup B$	□	$\mathit{map}_{\vee}(A, B)$	$A \oplus B$	□	$\mathit{map}_{+}(A, B)$
$A \cap B$	□	$\mathit{map}_{\wedge}(A, B)$	$A \Pi B$	□	$\mathit{map}_{\min}(A, B)$
$\mathit{finite}(A)$	□	$(\delta(A) = \mathit{false})$	$\mathit{finite}_{Bag}(A)$	□	$(\delta(A) = 0)$

- But not cardinality $|A|$, power-set 2^A, …

CAL: Arrays as Combinators

- McCarthy Arrays:
  store/select

$$select(store(a,i,v),i) = v$$
$$i \neq j \Rightarrow select(store(a,i,v),j) = select(a,j)$$

- Array combinators:

$$\begin{aligned} store(a,i,v) &:= \lambda j.\, \textbf{\textit{if}}\ i = j\ \textbf{\textit{then}}\ v\ \textbf{\textit{else}}\ select(a,j) \\ const(v) &\coloneqq \lambda i.\, v \\ map_f(a,b) &\coloneqq \lambda i.\, f(select(a,i), select(b,i)) \end{aligned}$$

- Takeaway: A common procedure for Array Combinators

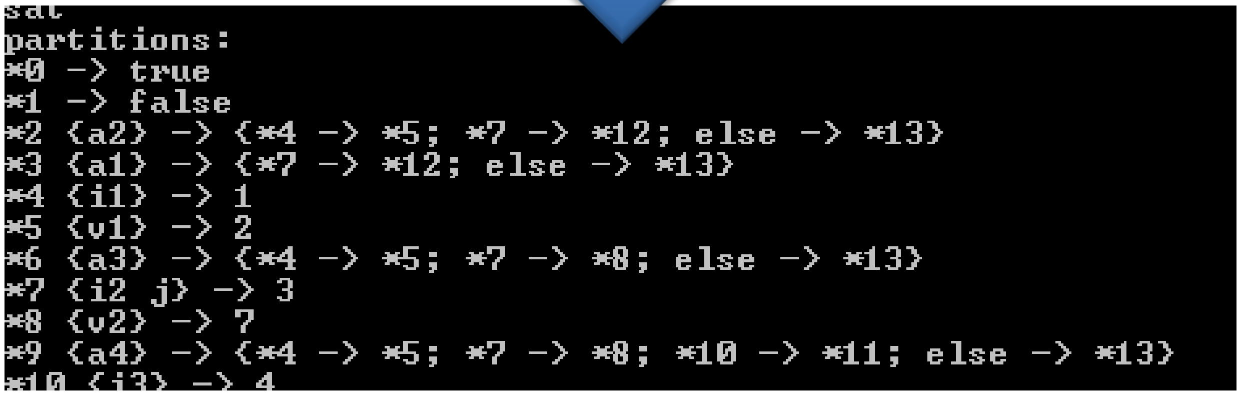
A reduction-based approach

$$Sat(T_{Array} \wedge \varphi)?$$

Use saturation rules to reduce arrays
to the theory of un-interpreted functions

$$Sat(T_{Equality} \wedge Closure_{Array}(\varphi) \wedge \varphi)?$$

Extract models for arrays as finite graphs



```
sat
partitions:
*0 -> true
*1 -> false
*2 {a2} -> {*4 -> *5; *7 -> *12; else -> *13}
*3 {a1} -> {*7 -> *12; else -> *13}
*4 {i1} -> 1
*5 {v1} -> 2
*6 {a3} -> {*4 -> *5; *7 -> *8; else -> *13}
*7 {i2 j} -> 3
*8 {v2} -> 7
*9 {a4} -> {*4 -> *5; *7 -> *8; *10 -> *11; else -> *13}
*10 {i3} -> 4
```

Deciding *store*

For every sub-term $store(a, i, v)$, every index j in φ, add equation to φ:

$$select(store(a, i, v), j) = \textbf{\textit{if}}\ i = j\ \textbf{\textit{then}}\ v\ \textbf{\textit{else}}\ select(a, j)$$

EUF model of φ => Array Model:

For each array *a* define

$$M_{array}(a) := \{\, M(i) \ \rightarrow \ M(select(a, i)), else \ \rightarrow \ \blacklozenge_{\boldsymbol{Ma}} \}$$

where select(a,i) occurs in φ.

Deciding *store*

For each array *a* in φ define

$$M_{array}(a) := \{\, M(i) \;\to\; M(select(a,i)), else \;\to\; \blacklozenge_{\boldsymbol{Ma}} \}$$

Does M satisfy axioms for *store?*

$$M(store(a,i,v)) \;= \lambda\, j.\ \boldsymbol{if}\ M(i) \;=\; j\ \boldsymbol{then}\ M(v)\ \boldsymbol{else}\ M(select(a,j))$$

Recall, we added

$$select(store(a,i,v),j) = \boldsymbol{if}\ i = j\ \boldsymbol{then}\ v\ \boldsymbol{else}\ select(a,j)$$

Thus, $M(select(store(a,i,v),j))$

$$= \;M\big(\boldsymbol{if}\ i = j\ \boldsymbol{then}\ v\ \boldsymbol{else}\ select(a,j)\big)$$

$$= \boldsymbol{if}\ M(i) \;= M(j)\ \boldsymbol{then}\ M(v)\ \boldsymbol{else}\ M(select(a,j))$$

Extesionality

$$\forall a, b\ \left(\left(\forall i\ .\ select(a, i) = select(b, i)\right) \Rightarrow a = b\right)$$

Not automatically satisfied by basic decision procedure.

Skolemized:

$$\forall a, b\ \left(\left(select(a, \delta(a, b)) = select(b, \delta(a, b))\right) \Rightarrow a = b\right)$$

Add instance for every pair *a, b.*

More Efficiently Deciding *store*

- *a~b* – *a* and *b* are equal in current context
- *a≡t* – *a* is a name for the term *t*

$$\text{idx}\ \frac{a \equiv store(b,i,v)}{a[i] \simeq v}$$

$$\Downarrow\ \frac{a \equiv store(b,i,v), \quad w \equiv a'[j], \quad a \sim a'}{i \simeq j \vee a[j] \simeq b[j]}$$

$$\Uparrow\ \frac{a \equiv store(b,i,v), \quad w \equiv b'[j], \quad b \sim b'}{i \simeq j \vee a[j] \simeq b[j]}$$

$$\text{ext}\ \frac{a\!:(\sigma \Rightarrow \tau), \quad b\!:(\sigma \Rightarrow \tau)}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

What makes it more *Efficient*?

- Axioms for *store* are only added by the model induced by EUF

Bottlenecks

$$\text{ext}\ \frac{a\colon(\sigma\Rightarrow\tau),\quad b\colon(\sigma\Rightarrow\tau)}{a\simeq b\vee a[k_{a,b}]\not\simeq b[k_{a,b}]}$$

- Extensionality axiom is instantiated on every pair of array variables.

$$\Uparrow\ \frac{a\equiv store(b,i,v),\quad w\equiv b'[j],\quad b\sim b'}{i\simeq j\vee a[j]\simeq b[j]}$$

- Upwards propagation distributes index over all modifications of same array.

Bottlenecks

$$\text{ext}\ \frac{a\colon(\sigma \Rightarrow \tau), \quad b\colon(\sigma \Rightarrow \tau)}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

Bottleneck:
Extensionality axiom is instantiated on every pair of array variables.

$$\text{ext}_{\not\simeq}\ \frac{p \equiv a \simeq b, \quad \Gamma(p) = \text{false}}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

$$\text{ext}_r\ \frac{a\colon(\sigma \Rightarrow \tau), \quad b\colon(\sigma \Rightarrow \tau), \quad \{a,b\} \subseteq \text{foreign}}{a \simeq b \vee a[k_{a,b}] \not\simeq b[k_{a,b}]}$$

Optimization: Restrict tc
variables asserted different, or shared.

Bottlenecks

$$\Uparrow \frac{a \equiv store(b,i,v), \quad w \equiv b'[j], \quad b \sim b'}{i \simeq j \vee a[j] \simeq b[j]}$$

$$\Uparrow_r \frac{a \equiv store(b,i,v), \quad w \equiv b'[j], \quad b \sim b', \quad b \in \text{non-linear}}{i \simeq j \vee a[j] \simeq b[j]}$$

- **Bottleneck:** Upwards propagation distributes index over all modifications of same array.
- **Optimization:** Only use $\Uparrow$ for updates where ancestor has multiple children. *Formulas from programs are well-behaved.*

Saturating K, map, δ

$$\text{K}\Downarrow\ \frac{a \equiv K(v), \quad w \equiv a'[j], \quad a \sim a'}{a[j] \simeq v}$$

$$\text{map}\Downarrow\ \frac{a \equiv map_f(b_1,\ldots,b_n), \quad w \equiv a'[j], \quad a \sim a'}{a[j] \simeq f(b_1[j],\ldots,b_n[j])}$$

$$\text{map}\Uparrow\ \frac{\begin{array}{c} a \equiv map_f(b_1,\ldots,b_n), \ \ w \equiv b'_k[j], \\ b_k \sim b'_k, \ \text{for some} \ \ k \in \{1,\ldots,n\} \end{array}}{a[j] \simeq f(b_1[j],\ldots,b_n[j])}$$

$$\epsilon\not\simeq\ \frac{v \equiv a[i], \quad i{:}\sigma, \quad i \text{ is not } \epsilon_\sigma}{\epsilon_\sigma \not\simeq i} \qquad \epsilon\delta\ \frac{a{:}(\sigma \Rightarrow \tau)}{a[\epsilon_\sigma] \simeq \delta_a}$$

Algebraic Data types

Scalars, Tuples and Composites

```
Fruit = Apple | Orange | Banana

Person = { name : String, age : Int, sex : M | F }

IntOption = Some of { ofSome : Int } | None
```

Recursive and Mutual Recursive types

List = Nil | Cons of { head : Int, tail : List }

Ping = DropP | WinP | Pi of { pong : Pong }

Pong = WinP | DropP | Po of { ping : Pong }

ADTs: Algebraic Data-types

- Constructors are injective:
  - head(cons(x,xs)) = x
  - tail(cons(x,xs)) = xs
- Terms are well-founded:
  - $xs \neq cons(x, xs)$
  - $xs \neq cons(x, cons(y, xs))$
  - $xs \neq cons(x, cons(y, cons(z, xs)))$
  - $xs \neq cons(x, cons(y, cons(z, cons(u, xs))))$

ADTs

- Outline of a decision Procedure:
  - Force injectivity:
    - For cons(t1,t2) add lemmas:
      - head(cons(t1,t2)) = t1
      - tail(cons(t1,t2)) = t2
  - Build pre-model for constants of data-type sort.
    - x = Nil y = Nil z = Nil
  - Perform occurs check in each equivalence class.
    - Q: can there be two constructors in an equivalence class?