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
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**

- Auth
- MSOL
- Sequences
- XDucers
- Queues
- ASP
- DL
- homomorphisms
- Optimization
- Orders
- Objects
- HOL
- MultiSets
- BAPA
- Strings
- Reg. Exprs.
- NRA
- NIA
- Floats
- f*
- *
- SAT
- EUF
- LRA
- LIA
- Arrays
- Bit-Vectors
- Alg. DT
Be afraid!

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 paper and read ascribe it.
Linear Real Arithmetic

- Many approaches
  - Graph-based for difference logic: \( a - b \leq 3 \)
  - Fourier-Motzkin elimination:
    \[
    t_1 \leq ax, \quad bx \leq t_2 \quad \Rightarrow \quad bt_1 \leq at_2
    \]
  - 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.

\[
x := x + 1
\]

\[
x_0 = x_1 + 1
\]

\[
x_1 - x_0 \leq 1, \ x_0 - x_1 \leq -1
\]
Job shop scheduling

<table>
<thead>
<tr>
<th>$d_{i,j}$</th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

$\max = 8$

**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**

$(t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land$
$(t_{2,1} \geq 0) \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \land$
$(t_{3,1} \geq 0) \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \land$

$((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \land$
$((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \land$
$((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \land$
$((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \land$
$((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \land$
$((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1)) \land$
Difference Logic

Chasing negative cycles!

Algorithms based on Bellman-Ford (O(mn)).
Unit Two Variables Per Inequality

\[ x + y \leq 5 \land -x + y \leq -4 \land y + y \geq 1 \]
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

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

\[ 2y \leq 1 \land 2y \geq 1 \]

\[ y \leq 0 \land y \geq 1 \]
Unit Two Variables Per Inequality: UTVPI

Reduce to Difference Logic:

• For every variable $x$ introduce fresh variables $x^+, x^-$

• Meaning: $2x := x^+ - x^-$

• Rewrite constraints as follows:

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

• \( x - y \leq k \) \implies \begin{cases} x^+ - y^+ \leq k \\ y^- - x^- \leq k \end{cases}

• \( x \leq k \) \implies x^+ - x^- \leq 2k

• \( x + y \leq k \) \implies \begin{cases} x^+ - y^- \leq k \\ y^+ - x^- \leq k \end{cases}

• \( x + y \leq k \) \implies \text{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 $x^-$

• $M(x) := (M(x^+) - M(x^-))/2$

• Nothing can go wrong...

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

• \( M(x) := (M(x^+) - M(x^-))/2 \)
• 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

\[ x^- - y^+ \leq 5 \]
\[ y^+ - z^- \leq -6 \]
\[ z^- - x^+ \leq -2 \]
\[ x^+ - v^+ \leq 3 \]
\[ v^+ - x^- \leq 0 \]  
\[ \Rightarrow \quad x^- - x^+ \leq -3 \]
\[ x^+ - x^- \leq 3 \]
General Form: \[ Ax = 0 \text{ and } l_j \leq x_j \leq u_j \]

Example:

\[ x \geq 0, (x + y \leq 2 \lor x + 2y \geq 6), (x + y = 2 \lor x + 2y > 4) \]

\[ \Rightarrow \]

\[ s_1 \equiv x + y, s_2 \equiv x + 2y, \]

\[ x \geq 0, (s_1 \leq 2 \lor s_2 \geq 6), (s_1 = 2 \lor s_2 > 4) \]

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

\[ s_1 \equiv x + y, \quad s_2 \equiv x + 2y \]

\[ s_1 = x + y, \quad s_2 = x + 2y \]
From Definitions to a Tableau

\[ s_1 \equiv x + y, \quad s_2 \equiv x + 2y \]

\[ s_1 = x + y, \]
\[ s_2 = x + 2y \]

\[ s_1 - x - y = 0 \]
\[ s_2 - x - 2y = 0 \]
From Definitions to a Tableau

\[ s_1 \equiv x + y, \quad s_2 \equiv x + 2y \]

\[ s_1 = x + y, \quad s_2 = x + 2y \]

\[ s_1 - x - y = 0 \quad s_1, s_2 \text{ are basic (dependent)} \]
\[ s_2 - x - 2y = 0 \quad x, y \text{ are non-basic} \]
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 $y$

\[
\begin{align*}
    s_1 - x - y &= 0 \\
    s_2 - x - 2y &= 0
\end{align*}
\]
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 $y$

\[
\begin{align*}
    s_1 - x - y &= 0 \\
    s_2 - x - 2y &= 0 \\
    -s_1 + x + y &= 0 \\
    s_2 - x - 2y &= 0
\end{align*}
\]
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 $y$

\[
\begin{align*}
  s_1 - x - y &= 0 \\
  s_2 - x - 2y &= 0 \\
  -s_1 + x + y &= 0 \\
  s_2 - x - 2y &= 0 \\
  -s_1 + x + y &= 0 \\
  s_2 - 2s_1 + x &= 0
\end{align*}
\]
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 $y$

\[
\begin{align*}
    s_1 - x - y &= 0 \\
    s_2 - x - 2y &= 0 \\
    -s_1 + x + y &= 0 \\
    s_2 - x - 2y &= 0 \\
    -s_1 + x + y &= 0 \\
    s_2 - 2s_1 + x &= 0
\end{align*}
\]

It is just substituting equals by equals.
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 $y$

\[
\begin{align*}
  s_1 - x - y &= 0 \\
  s_2 - x - 2y &= 0 \\
  -s_1 + x + y &= 0 \\
  s_2 - x - 2y &= 0 \\
  -s_1 + x + y &= 0 \\
  s_2 - 2s_1 + x &= 0
\end{align*}
\]

It is just substituting equals by equals.

Definition:

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

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

\begin{align*}
    s_1 - x - y &= 0 \\
    s_2 - x - 2y &= 0 \\
    -s_1 + x + y &= 0 \\
    s_2 - x - 2y &= 0 \\
    -s_1 + x + y &= 0 \\
    s_2 - 2s_1 + x &= 0
\end{align*}

It is just substituting equals by equals.

Example:
\begin{align*}
    M(x) &= 1 \\
    M(y) &= 1 \\
    M(s_1) &= 2 \\
    M(s_2) &= 3
\end{align*}

Key Property:
If an assignment satisfies the equations before a pivoting step, then it will also satisfy them after!
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: \( x = y - z, \ y \leq 2, \ z \geq 3 \Rightarrow x \leq -1. \)

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.

\[
\begin{align*}
  a &= c - d \\
  b &= c + d \\
  M(a) &= 0 \\
  M(b) &= 0 \\
  M(c) &= 0 \\
  M(d) &= 0 \\
  1 &\leq c
\end{align*}
\]

\[
\begin{align*}
  a &= c - d \\
  b &= c + d \\
  M(a) &= 1 \\
  M(b) &= 1 \\
  M(c) &= 1 \\
  M(d) &= 0 \\
  1 &\leq c
\end{align*}
\]
“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”.

\[
\begin{align*}
a &= c - d \\
b &= c + d \\
M(a) &= 0 \\
M(b) &= 0 \\
M(c) &= 0 \\
M(d) &= 0 \\
1 &\leq c \\
a &\leq 0
\end{align*}
\]
“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.

\[
\begin{align*}
  a &= c - d \\
  b &= c + d \\
  M(a) &= 0 \\
  M(b) &= 0 \\
  M(c) &= 0 \\
  M(d) &= 0 \\
  1 \leq a
\end{align*}
\]

\[
\begin{align*}
  c &= a + d \\
  b &= a + 2d \\
  M(a) &= 0 \\
  M(b) &= 0 \\
  M(c) &= 0 \\
  M(d) &= 0 \\
  1 \leq a
\end{align*}
\]

\[
\begin{align*}
  c &= a + d \\
  b &= a + 2d \\
  M(a) &= 1 \\
  M(b) &= 1 \\
  M(c) &= 1 \\
  M(d) &= 0 \\
  1 \leq a
\end{align*}
\]
“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
Extracting proof from failed repair attempts is easy.

\[ s_1 \equiv a + d, \ s_2 \equiv c + d \]

\[ a = s_1 - s_2 + c \]

\[ a \leq 0, \ 1 \leq s_1, \ s_2 \leq 0, \ 0 \leq c \]

\[ M(a) = 1 \]

\[ M(s_1) = 1 \]

\[ M(s_2) = 0 \]

\[ M(c) = 0 \]
Extracting proof from failed repair attempts is easy.

\[ s_1 = a + d, \ s_2 = c + d \]
\[ a = s_1 - s_2 + c \]
\[ a \leq 0, \ 1 \leq s_1, \ s_2 \leq 0, \ 0 \leq c \]
\[ M(a) = 1 \]
\[ M(s_1) = 1 \]
\[ M(s_2) = 0 \]
\[ M(c) = 0 \]

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

\{ a \leq 0, \ 1 \leq a + d, \ c + d \leq 0, \ 0 \leq c \} \text{ is inconsistent}
Arrays and Combinatory Array Logic
What are arrays?

• Applicative stores:
  \[ \text{write}(a, i, v)[i] = v \]
  \[ i \neq j \Rightarrow \text{write}(a, i, v)[j] = a[j] \]

• Or, special combinator:
  \[ \text{write}(a, i, v) = \lambda j. \text{ite}(i = j, v, a[j]) \]
What are arrays?

• Special combinator:

\[ write(a, i, v) = \lambda j.\text{ite}(i = j, v, a[j]) \]

• 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:**

  \[\text{write}(a, i, v) = \lambda j.\text{ite}(i = j, v, a[j])\]

  \[K(v) = \lambda j.v\]

  \[\text{map}_f(a, b) = \lambda j.f(a[j], b[j])\]

- **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:

\[
\text{write}(a, i, v) = \lambda j.\text{ite}(i = j, v, a[j])
\]

\[
K(v) = \lambda j.v
\]

\[
\text{map}_f(a, b) = \lambda j.f(a[j], b[j])
\]

\[
I = \lambda j.j
\]

• Easy to encode lambda calculus
What else are arrays++?

- Encoding lambda terms into CAL+:

\[
\begin{align*}
[[\lambda x. M]] &= tr(x, [[M]]) \\
[[x]] &= x \\
[[M N]] &= \text{map}_{\text{read}}([[M]], [[N]])
\end{align*}
\]

\[
\begin{align*}
tr(x, x) &= I \\
tr(x, y) &= K(y) \\
tr(x, f(M, N)) &= \text{map}_f(tr(x, M), tr(x, N))
\end{align*}
\]

- 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 \textit{read}. 
Example translation

$$[[\lambda x.((\lambda y.(yx))x)]]$$

$$= tr(x,[[((\lambda y.(yx))x)]])$$

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

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

$$= tr(x,map_{read}(tr(y,map_{read}(y,x)),x))$$

$$= tr(x,map_{read}(map_{map_{read}}(tr(y,y),tr(y,x))),x))$$

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

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

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

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

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

$$= map_{map_{read}}(map_{map_{map_{read}}}(K(I),map_K(I)),I)$$
... But there are arrays#:

- Restricted theory using $I$.
  
  $K(v) = \lambda j. v$

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

  $\text{map}_{=} (a, b) = \lambda j. (a[j] = b[j])$

  $I = \lambda j. j$

- Then:
  
  $\text{write}(a, i, v) = \text{map}_{\text{ite}}(\text{map}_{=} (K(i), I), K(v), a)$

- Theory of arrays# is decidable.
Last combinator for the road...

- Can I access a *default* array value?

\[ \delta(a) – \text{default} \]

\[ \delta(K(v)) = v \]

\[ \delta(map_f(a, b)) = f(\delta(a), \delta(b)) \]

\[ \delta(write(a, i, v)) = \delta(a) \]  

Only sound for infinite domains
Let’s use CAL:

• Simple set and bag operations:

<table>
<thead>
<tr>
<th>Set Operation</th>
<th>Bag Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset_{Bag})</td>
<td>(K(0))</td>
</tr>
<tr>
<td>({a})</td>
<td>write((\emptyset, a, true))</td>
<td>({a})</td>
</tr>
<tr>
<td>(a \in A)</td>
<td>A[a]</td>
<td>mult(a, A)</td>
</tr>
<tr>
<td>(A \cup B)</td>
<td>map_\vee(A, B)</td>
<td>A \oplus B</td>
</tr>
<tr>
<td>(A \cap B)</td>
<td>map_\wedge(A, B)</td>
<td>A \sqcup B</td>
</tr>
<tr>
<td>finite(A)</td>
<td>((\delta(A) = false))</td>
<td>finite_{Bag}(A)</td>
</tr>
<tr>
<td>finite(A)</td>
<td>((\delta(A) = false))</td>
<td>finite_{Bag}(A)</td>
</tr>
</tbody>
</table>

• But not cardinality \(|A|\), power-set \(2^A\), ...
CAL: Arrays as Combinators

• McCarthy Arrays: 
  
  \[ \text{store/select} \]

\[ \text{select}(\text{store}(a, i, v), i) = v \]
\[ i \neq j \Rightarrow \text{select}(\text{store}(a, i, v), j) = \text{select}(a, j) \]

• Array combiners:

\[ \text{store}(a, i, v) := \lambda j. \text{if } i = j \text{ then } v \text{ else } \text{select}(a, j) \]
\[ \text{const}(v) := \lambda i. v \]
\[ \text{map}_f(a, b) := \lambda i. f(\text{select}(a, i), \text{select}(b, i)) \]

• Takeaway: A common procedure for Array Combinators
A reduction-based approach

\[ \text{Sat}(T_{\text{Array}} \land \varphi) ? \]

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

\[ \text{Sat}(T_{\text{Equality}} \land \text{Closure}_{\text{Array}} (\varphi) \land \varphi) ? \]

Extract models for arrays as finite graphs
Deciding store

For every sub-term \( \text{store}(a, i, v) \), every index \( j \) in \( \varphi \), add equation to \( \varphi \):

\[
\text{select}(\text{store}(a, i, v), j) = \text{if } i = j \text{ then } v \text{ else select}(a, j)
\]

EUF model of \( \varphi \) => Array Model:

For each array \( a \) define

\[
M_{\text{array}}(a) := \{ M(i) \rightarrow M(\text{select}(a, i)), \text{else } \rightarrow \text{ }\}
\]

where select(a,i) occurs in \( \varphi \).
Deciding \textit{store}

For each array \(a\) in \(\varphi\) define
\[
M_{\text{array}}(a) := \{ M(i) \to M(\text{select}(a, i)), \text{else} \to \bigstar_M a \}
\]

Does \(M\) satisfy axioms for \textit{store}?

\[
M(\text{store}(a, i, v)) = \lambda j. \text{if } M(i) = j \text{ then } M(v) \text{ else } M(\text{select}(a, j))
\]

Recall, we added

\[
\text{select}(\text{store}(a, i, v), j) = \text{if } i = j \text{ then } v \text{ else } \text{select}(a, j)
\]

Thus, \(M(\text{select}(\text{store}(a, i, v), j))\)
\[
= M(\text{if } i = j \text{ then } v \text{ else } \text{select}(a, j))
= \text{if } M(i) = M(j) \text{ then } M(v) \text{ else } M(\text{select}(a, j))
\]
Extesionality

\[ \forall a, b \left( (\forall i . \text{select}(a, i) = \text{select}(b, i)) \Rightarrow a = b \right) \]

Not automatically satisfied by basic decision procedure.

Skolemized:

\[ \forall a, b \left( (\text{select}(a, \delta(a, b)) = \text{select}(b, \delta(a, b))) \Rightarrow a = b \right) \]

Add instance for every pair \( a, b \).
More Efficiently Deciding \( \text{store} \)

- \( a \sim b \) – \( a \) and \( b \) are equal in current context
- \( a \equiv t \) – \( a \) is a name for the term \( t \)

\[
\begin{align*}
\text{idx} \quad &a \equiv \text{store}(b, i, v) \\
&\frac{a[i] \approx v}{i \leq j \lor a[j] \approx b[j]} \\
\downarrow \quad &a \equiv \text{store}(b, i, v), \quad w \equiv a'[j], \quad a \sim a' \\
&i \leq j \lor a[j] \approx b[j] \\
\uparrow \quad &a \equiv \text{store}(b, i, v), \quad w \equiv b'[j], \quad b \sim b' \\
&i \leq j \lor a[j] \approx b[j] \\
\text{ext} \quad &a : (\sigma \Rightarrow \tau), \quad b : (\sigma \Rightarrow \tau) \\
&a \approx b \lor a[k_{a, b}] \not= b[k_{a, b}]
\end{align*}
\]
What makes it more *Efficient*?

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

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

\[
\text{ext} \quad \frac{a : (\sigma \Rightarrow \tau), \quad b : (\sigma \Rightarrow \tau)}{a \simeq b \lor a[k_{a,b}] \neq b[k_{a,b}]}
\]

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

\[
\uparrow \quad \frac{a \equiv \text{store}(b, i, v), \quad w \equiv b'[j], \quad b \sim b'}{i \simeq j \lor a[j] \simeq b[j]}
\]
Bottlenecks

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

\[
\begin{align*}
\text{ext} \quad & \frac{a : (\sigma \Rightarrow \tau), \quad b : (\sigma \Rightarrow \tau)}{a \simeq b \lor a[k_{a,b}] \neq b[k_{a,b}]} \\
\text{ext} \neq \quad & \frac{p \equiv a \simeq b, \quad \Gamma(p) = \text{false}}{a \simeq b \lor a[k_{a,b}] \neq b[k_{a,b}]} \\
\text{ext}_r \quad & \frac{a : (\sigma \Rightarrow \tau), \quad b : (\sigma \Rightarrow \tau), \quad \{a, b\} \subseteq \text{foreign}}{a \simeq b \lor a[k_{a,b}] \neq b[k_{a,b}]} 
\end{align*}
\]

**Optimization:** Restrict to variables asserted different, or shared.
Bottlenecks:

- **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, $\delta$

\[
\begin{align*}
\text{K}\downarrow & \quad a \equiv K(v), \quad w \equiv a'[j], \quad a \sim a' \\
& \quad a[j] \simeq v \\
\text{map}\downarrow & \quad a \equiv \text{map}_f(b_1, \ldots, b_n), \quad w \equiv a'[j], \quad a \sim a' \\
& \quad a[j] \simeq f(b_1[j], \ldots, b_n[j]) \\
\text{map}\uparrow & \quad a \equiv \text{map}_f(b_1, \ldots, b_n), \quad w \equiv b'_k[j], \\
& \quad b_k \sim b'_k, \text{ for some } k \in \{1, \ldots, n\} \\
& \quad a[j] \simeq f(b_1[j], \ldots, b_n[j]) \\
\epsilon \not\in \ & \quad v \equiv a[i], \quad i : \sigma, \quad i \text{ is not } \epsilon_\sigma \\
& \quad \epsilon_\sigma \not\in i \\
\epsilon \delta & \quad a : (\sigma \Rightarrow \tau) \\
& \quad a[\epsilon_\sigma] \simeq \delta a
\end{align*}
\]
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?