A Logic Your Typechecker Can Count On: Unordered Tree Types in Practice

Nate Foster (Penn)
Benjamin C. Pierce (Penn)
Alan Schmitt (INRIA Rhône-Alpes)

16 Feb 2007

\[
\begin{align*}
\mu X. \{ \} & (\text{hd}[T] + \text{tl}[X]) \\
\Downarrow \\
\phi(x_0, \ldots, x_4), \\
& [\text{hd}[T], \text{hd}[\neg T], \\
& \text{tl}[X], \text{tl}[\neg X], \\
& \{\text{hd}, \text{tl}\}[\text{True}]
\end{align*}
\]
\[
\mu X. \{\} | (hd[T] + tl[X]) \\
\downarrow \\
\phi(x_0, \ldots, x_4), \\
[hd[T], hd[\neg T],] \\
tl[X], tl[\neg X], \\
\{hd, tl\}[True]
\]
Harmony

A generic synchronization framework

- Architecture takes two replicas + original ⇒ updated replicas.
- Data model is “deterministic” trees: unordered, edge-labeled trees.
Harmony: Typed Synchronization [DBPL '05]

Behavior of synchronizer guided by type.

- If inputs well-typed, so are outputs.
- Required operations: membership of trees in type [also sets of names].
Types in Harmony

Harmony: Lenses [POPL ’05]

Pre-/post-process replicas using bi-directional programs.

- Facilitates heterogeneous synchronization.
- Types in conditionals, run-time asserts, static checkers.
- Required operations: membership, inclusion, equivalence, emptiness, [projection, injection, etc.].
## Deterministic Tree Types

### Syntax

$$T ::= \{\} | n[T] | T + T | T | T | \sim T | X | !\{n_1, \ldots, n_k\}[T] | *\{n_1, \ldots, n_k\}[T]$$
Deterministic Tree Types

Syntax

\[
T ::= \{\} | n[T] | T + T | T | T | \sim T | X \\
| \!\{n_1, \ldots, n_k\}[T] | \ast\{n_1, \ldots, n_k\}[T]
\]

Semantics

Singleton denoting the unique tree with no children:

\[
\circ \in \{\}
\]
## Deterministic Tree Types

### Syntax

\[
T ::= \{\} \mid n[T] \mid T + T \mid T \mid T \mid \sim T \mid X \\
\mid \!\backslash\{n_1, \ldots, n_k\}[T] \mid \ast\backslash\{n_1, \ldots, n_k\}[T]
\]

### Semantics

Atoms: trees with single child \( n \) and subtree in \( T \):

If \( t \in T \), then \( n \in n[T] \)

\[\begin{align*}
\text{If } t & \in T, \text{ then } n \in n[T] \\
\end{align*}\]
Deterministic Tree Types

Syntax

\[ T ::= \emptyset | n[T] | T+T | T | T | \sim T | X | \{ n_1, \ldots, n_k \} [T] | \ast \{ n_1, \ldots, n_k \} [T] \]

Semantics

Commutative concatenation operator:

If \( t \in T \) and \( t' \in T' \), then \( t + t' \in T + T' \)
Deterministic Tree Types

Syntax

\[ T ::= \{\} | n[T] | T+T | T|T | \sim T | X \]
| \[ !\{n_1, \ldots, n_k\}[T] | *\{n_1, \ldots, n_k\}[T] \]

Semantics

Boolean operations and recursion:

\[ X_1 = T_1 \]
\[ \vdots \]
\[ X_n = T_n \]
Deterministic Tree Types

Syntax

\[ T ::= \{\} | n[T] | T + T | T | T | \sim T | X | ![\{n_1, .., n_k\}[T] | *![\{n_1, .., n_k\}[T] \]

Semantics

If \( m \not\in \{n_1, .., n_k\} \) and

\( \in T \), then

\( \in ![\{n_1, .., n_k\}[T] \)
Deterministic Tree Types

Syntax

\[ T ::= \{\} | n[T] | T+T | T|T | \sim T | X \\
| \!\{n_1, \ldots, n_k\}[T] | *\{n_1, \ldots, n_k\}[T] \]

Semantics

If \( m_1, \ldots, m_k \not\in \{n_1, \ldots, n_k\} \) and

\[ t_1, \ldots, t_k \in T, \text{ then } m_1 \ldots m_k \in \{n_1, \ldots, n_k\}[T] \]

\[ t_1, \ldots, t_k \in T, \text{ then } *\{n_1, \ldots, n_k\}[T] \]

\[ t_1, \ldots, t_k \in T, \text{ then } m_1 \ldots m_k \in *\{n_1, \ldots, n_k\}[T] \]
## Deterministic Tree Types

### Syntax

\[ T ::= \{\} | n[T] | T + T | T \cdot T | \sim T | X | !\{n_1, \ldots, n_k\}[T] | *\{n_1, \ldots, n_k\}[T] \]

### Example: \(hd [True] + tl [True]\)
Deterministic Tree Types

Syntax

\[ T ::= \{\} | n[T] | T + T | T | T | \neg T | X \]
\[ \ | \ !\{n_1,..,n_k\}[T] | \ast\{n_1,..,n_k\}[T] \]

Example:  \{\} \mid (hd [True] + tl [True])
Deterministic Tree Types

Syntax

\[ T ::= \{\} | n[T] | T+T | T\upharpoonright T | \sim T | X \\
| \downarrow \{n_1, \ldots, n_k\}[T] | \uparrow \{n_1, \ldots, n_k\}[T] \]

Example: \( X = \{\} | (hd [True]+tl [X]) \)
## Deterministic Tree Types

**Syntax**

\[
T ::= \{\} | n[T] | T+T | T|T | \sim T | X \\
[ | ![\{n_1, \ldots, n_k\}[T] | ![\{n_1, \ldots, n_k\}[T] ]
\]

**Example:** ![\![True]\]+![True]
Deterministic Tree Types

Syntax

\[ T ::= \{\} | n[T] | T+T | T|T | \sim T | X \\
| !\{n_1, \ldots, n_k\}[T] | *\{n_1, \ldots, n_k\}[T] \]

Example: \(~(![True]+![True])\)

Can eliminate negations, and use direct algorithms, but types get large...
### Sheaves Formulas

**Formulas**

$$ S = \phi(x_0, \ldots, x_k), \quad [r_0[S_0], \ldots, r_k[S_k]] $$

where $\phi$ is a Presburger formula and $r_i$ a set of names.

[Dal Zilio, Lugiez, Meyssonnier, POPL ’04]
Sheaves Formulas

Formulas

\[ S = \phi(x_0, \ldots, x_k), \]
\[ [r_0[S_0], \ldots, r_k[S_k]] \]

where \( \phi \) is a Presburger formula and \( r_i \) a set of names.

\[ \phi(x_0, x_1), \]
\[ [b[True], \{a, c\}[True]] \]
Sheaves Formulas

Formulas

\[ S = \phi(x_0, \ldots, x_k), \quad [r_0[S_0], \ldots, r_k[S_k]] \]

where \( \phi \) is a Presburger formula and \( r_i \) a set of names.

\[ \phi(x_0, x_1), \quad [b[True], \{a, c\}[True]] \]
Sheaves Formulas

Formulas

\[ S = \phi(x_0, \ldots, x_k), \quad [r_0[S_0], \ldots, r_k[S_k]] \]

where \( \phi \) is a Presburger formula and \( r_i \) a set of names.

\[ \phi(x_0, x_1), \quad [b[True], \{a, c\}[True]] \]
Sheaves Formulas

Formulas

\[ S = \phi(x_0, \ldots, x_k), \]
\[ [r_0[S_0], \ldots, r_k[S_k]] \]

where \( \phi \) is a Presburger formula and \( r_i \) a set of names.

\[ \phi(x_0, x_1), \]
\[ [b[True], \{a, c\}[True]] \]
<table>
<thead>
<tr>
<th>Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = \phi(x_0, .., x_k), \newline [r_0[S_0], .., r_k[S_k]]$ where $\phi$ is a Presburger formula and $r_i$ a set of names.</td>
</tr>
</tbody>
</table>

$\phi(x_0, x_1), \newline [b[True], \{a, c\}[True]]$

$\models \phi(1, 2)$
Sheaves Formulas

Formulas

\[ S = \phi(x_0, \ldots, x_k), \]
\[ \{ r_i[S_i] \mid r_i \text{ a set of names} \} \]

where \( \phi \) is a Presburger formula and \( r_i \) a set of names.

\[ \phi(x_0, x_1, x_2), \]
\[ \{ b[True], \{ a, c \}[True], \{ a, b, c \}[True] \} \]

For coherence: \( r_i[S_i] \) must partition set of atoms.
Note: does not ensure determinism.
Examples as Sheaves Formulas

\[
X = (\{\} \mid \text{hd}[\text{True}] \lor \text{tl}[X])
\]

\[
X = \begin{cases} 
(x_0 = x_1 = x_2 = x_3 = 0) \lor \\
(x_0 = x_1 = 1 \land x_2 = x_3 = 0), \\
[\text{hd}[\text{True}], \text{tl}[X], \text{tl}[\neg X], \{\text{hd, tl}\}[\text{True}]]
\end{cases}
\]
Examples as Sheaves Formulas

\[ X = (\{\} \mid \text{hd}[\text{True}] + \text{tl}[X]) \]

\[ X = (x_0 = x_1 = x_2 = x_3 = 0) \lor \]
\[ (x_0 = x_1 = 1 \land x_2 = x_3 = 0), \]
\[ \text{hd}[\text{True}], \text{tl}[X], \text{tl}[\neg X], \{\text{hd, tl}\}[\text{True}] \]

\[ \sim (\neg [\text{True}] + \neg [\text{True}]) \]

\[ x_0 \neq 2, \]
\[ \{\}[\text{True}] \]
Challenges and Strategies

Blowup in naive compilation from types to formulas.
  ▶ **Syntactic optimizations** avoid blowup in common cases.
Backtracking in top-down, non-deterministic traversal.
  ▶ **Incremental algorithm** avoids useless paths.
Presburger arithmetic requires double-exponential time.
  ▶ Compile Presburger formulas to **MONA** representation.
  ▶ **Hash-consing** allocation + aggressive memoization.
Challenges and Strategies

Blowup in naive compilation from types to formulas.

- Syntactic optimizations avoid blowup in common cases.
- Backtracking in top-down, non-deterministic traversal.
  - Incremental algorithm avoids useless paths.
Presburger arithmetic requires double-exponential time.

- Compile Presburger formulas to MONA representation.
- Hash-consing allocation + aggressive memoization.

Contributions

- Strategies and algorithms;
- Implementation in Harmony;
- Experimental results.
Incremental Algorithm

$$\phi(x_0, \ldots, x_k), \quad [r_0[S_0], \ldots r_k[S_k]]$$
Incremental Algorithm

\[ \phi(x_0, \ldots, x_k), \]
\[ [r_0[S_0], \ldots r_k[S_k]] \]
Incremental Algorithm

\[ \phi(x_0, \ldots, x_k), \]
\[ [r_0[S_0], \ldots, r_k[S_k]] \]

\((\phi \land \psi_{\text{dom}})\)
Incremental Algorithm

\[ \phi(x_0, \ldots, x_k), \]
\[ [r_0[S_0], \ldots r_k[S_k]] \]

\[ (\phi \land \psi_{\text{dom}} \land \psi_1) \]
Incremental Algorithm

\[ \phi(x_0, \ldots, x_k), [r_0[S_0], \ldots r_k[S_k]] \]

\[ (\phi \wedge \psi_{\text{dom}} \wedge \psi_1 \wedge \psi_2) \]
Incremental Algorithm

\[ \phi(x_0, \ldots, x_k), \]
\[ [r_0[S_0], \ldots r_k[S_k]] \]
\[ (\phi \land \psi_{\text{dom}} \land \psi_1 \land \ldots \land \psi_{k-1}) \]
Incremental Algorithm

\[ \phi(x_0, \ldots, x_k), \]
\[ [r_0[S_0], \ldots, r_k[S_k]] \]

\[ (\phi \land \psi_{\text{dom}} \land \psi_1 \land \ldots \land \psi_k) \]
Hash-Consing and Memoization

Thousands of formulas and trees, but many repeats.

Suggests hash-consed allocation:
  ▶ Sheaves formulas;
  ▶ Presburger formulas;
  ▶ Trees.

Memoization of intermediate results:
  ▶ MONA representations of Presburger formulas;
  ▶ Satisfiability of Presburger formulas;
  ▶ Membership results;
  ▶ Partially-evaluated member functions.
Experiments

Programs:

- Structured text parser;
- Address book validator;
- iCalendar lens.

Experimental setup: structures populated with snippets of Joyce’s *Ulysses*; 1.4GHz Intel Pentium III, 2GB RAM, SuSE Linux OS kernel 2.6.16; execution times collected from POSIX functions.
Experiments: Address Book Validator

<table>
<thead>
<tr>
<th>States</th>
<th>Formulas</th>
<th>Sat</th>
<th>Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>107517</td>
<td>99.8%</td>
<td>25727</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.9%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>156615</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>42.1%</td>
</tr>
</tbody>
</table>
Experiments: Address Book Validator

<table>
<thead>
<tr>
<th>States</th>
<th>Formulas</th>
<th>Sat</th>
<th>Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>107517</td>
<td>99.8%</td>
<td>25727</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>156615</td>
</tr>
</tbody>
</table>
Experiments: Address Book Validator

<table>
<thead>
<tr>
<th>States</th>
<th>Formulas</th>
<th>Sat</th>
<th>Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>107517</td>
<td>99.8%</td>
<td>25727</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>156615</td>
</tr>
</tbody>
</table>
Experiments: Structured Text Parser

<table>
<thead>
<tr>
<th>States</th>
<th>Formulas</th>
<th>Sat</th>
<th>Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>12461</td>
<td>99.1%</td>
<td>222</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>92.8%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3507706</td>
</tr>
</tbody>
</table>
Experiments: iCalendar Lens

<table>
<thead>
<tr>
<th>States</th>
<th>Formulas</th>
<th>Sat</th>
<th>Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>361</td>
<td>116939</td>
<td>97.4%</td>
<td>17600</td>
</tr>
</tbody>
</table>
Related Work

Types and Automata:

▶ TQL [Cardelli and Ghelli, ESOP ’01]
▶ “A Logic You Can Count On” [Dal Zilio, Lugiez, Meyssonnier, POPL ’04]
▶ “Counting In Trees For Free” [Seidl, Schwentick, Muscholl, Habermehl, ICALP ’04]
▶ Survey and Foundations: [Boneva and Talbot, RTA ’05, LICS ’05]

Implementations:

▶ “Static Checkers for Tree Structures and Heaps” [Hague ’04]
▶ “Boolean Operations and Inclusion Test for Attribute Element Constraints” [Hosoya and Murata, ICALP ’03]
Conclusions and Future Work

Summary

- Strategies and algorithms;
- Implemented in Harmony;
- Reasonable performance.

Tune algorithm, hash-consing, memoization parameters.

Determinize sheaves formulas.

Implement Presburger arithmetic directly, optimized for adding constraints incrementally; also restricted fragments.

Extend to new structures and types: multitrees, ordered trees, also horizontal recursion, adjoint operators, etc.
Acknowledgements

Haruo Hosoya, Christian Kirkegaard, Stéphane Lescuyer, Thang Nguyen, Val Tannen, Penn PLClub and DB Group.

http://www.seas.upenn.edu/~harmony/