Deciding Unbounded Heaps in an SMT Framework

Zvonimir Rakamarić\textsuperscript{1}, Roberto Bruttomesso\textsuperscript{2}, Alan Hu\textsuperscript{1}, Alessandro Cimatti\textsuperscript{2}

\textsuperscript{1}University of British Columbia; \textsuperscript{2}ITC-IRST

5th International Workshop on Satisfiability Modulo Theories
July 1, 2007
Introduction

- Heap-manipulating programs (HMPs)
  - Manipulate unbounded heap structures via pointers
- Many specialized theories for verification of HMPs
  - Great for heap properties (unbounded reachability)
  - Poor handling of data (integers, reals, linear arithmetic)
  - Real code - we need both!
- SMT solvers
  - Widely used in software verification
  - Many important theories (linear arith., arrays, ...)
  - None supports a theory for HMP verification
  - Real code - we need both!
Introduction

Solution
- Integrate a heap theory into an SMT solver

Logic for HMPs
- Expressive enough logic
- Efficient decision procedure
- Only boolean data

MathSAT
- Supports many required theories
- Easy integration of new theories via Delayed Theory Combination (DTC)

Contributions
- Integration
- Experiments – it actually works (usable and efficient)
Overview

- Introduction
- Logic for HMPs
- Delayed Theory Combination (DTC)
- Experiments
- Conclusion
Overview

- Introduction
- Logic for HMPs
- Delayed Theory Combination (DTC)
- Experiments
- Conclusion
Logic for HMPs - Syntax

c ∈ Constants
x ∈ DataVariables  v ∈ PointerVariables
d, d' ∈ DataFields  f, f' ∈ PointerFields

NodeTerm ::= v | next(f, NodeTerm)
DataTerm ::= c | x | data(d, NodeTerm)
Atom ::= NodeTerm = NodeTerm | DataTerm = DataTerm | reach(f, NodeTerm, NodeTerm) | between(f, NodeTerm, NodeTerm, NodeTerm)

Literal ::= Atom | ~Atom |
update_pfield(f, NodeTerm, NodeTerm, f') | update_dfield(d, NodeTerm, DataTerm, d')

Formula ::= Lit. | Formula ∧ Formula | Formula ∨ Formula
Logic for HMPs

- **Theory of Data Fields**
  - \{ =, data, update_dfield \}
  - Handled by MathSAT
    - EUF + update axioms
  - Currently only boolean and integer data fields
    - Easily extendable to other MathSAT’s data types

- **Theory of Unbounded Reachability over Heaps**
  - \{ =, next, reach, between, update_pfield \}
  - Handled by the saturation based decision procedure described in previous work

- **Signatures disjoint from other MathSAT’s theories**
Theory of Unbounded Reachability

- Semantics defined over *heap structures*
  - Set of heap nodes
  - Set of pointer variables
    - Pointers to heap nodes
  - Set of pointer fields
    - Links between heap nodes
- Reachability and between atoms
- Stably-infinite
- Non-convex
Unbounded Reachability

- \( \text{reach}(f, x, y) \) – unbounded reachability (i.e. transitive closure)
  - Node \( y \) is reachable from node \( x \) following 0 or more \( f \) pointer fields
Between Atom

- \texttt{between}(f, x, y, z)
  - Node \(y\) is between nodes \(x\) and \(z\) following \(f\) pointer fields
  - Crucial for expressing necessary properties about cyclic lists

\[
\begin{array}{c}
\text{between}(f, x, y, z) \\
\text{Node } y \text{ is between nodes } x \text{ and } z \text{ following } f \text{ pointer fields} \\
\text{Crucial for expressing necessary properties about cyclic lists}
\end{array}
\]
Example

- Logic for HMPs with integer data fields and linear arithmetic over integers

Some true literals:
- next = curr
- data(d, curr) = 13
- data(d, curr) = data(d, prev) + 4
- reach(f, head, prev)

Some false literals:
- next = nil
- data(d, prev) > data(d, curr)
- reach(f, next, prev)
Example with Update

True literals:
update_pfield(f, prev, head, g)
update_pfield(g, prev, next, f)

False literals:
update_pfield(f, prev, nil, g)
Overview

- Introduction
- Logic for HMPs
  - Delayed Theory Combination (DTC)
- Experiments
- Conclusion
Delayed Theory Combination

- Uses boolean engine for communication between theory solvers
  - Eagerly introduce the set of all possible equality atoms between shared variables
  - The set contains interface equalities that theory solvers might need to exchange
  - The communication is emulated by the enumeration of all possible interface equalities
Delayed Theory Combination

- Advantages
  - Integration is implicitly handled at the boolean level and not at the solver level
    - No need to build a Nelson-Open “box” around theory solvers
  - Disjunction in case of non-convexity is automatically handled at the boolean level
  - Theory solvers don’t need deduction capabilities
Integration

- Easily accomplished because of advantages of DTC
  - Almost no changes to the decision procedure or its interface
- Data updates $\text{update_dfield}(d, t, v, d')$ are handled by adding eagerly the following set of axioms
  \[
  \{d'(t) \approx v\} \cup \{d'(s) \approx d(s) \mid s \in \text{NodeTerm}, s \neq t\}
  \]
where $\approx$ is the equality $=$ for integer data and $\leftrightarrow$ for boolean data.
Overview

- Introduction
- Logic for HMPs
- Delayed Theory Combination (DTC)
  - Experiments
- Conclusion
Experiments

- Tested extended MathSAT using queries generated from predicate abstraction of HMP examples
- Comparison with pure unbounded reachability decision procedure from previous work
- HMPs we couldn’t handle before
- Three sets of results
  - No data (old DP and MathSAT)
  - Boolean data (old DP and MathSAT); Integer data, difference logic (MathSAT)
  - Integer data, linear arithmetic (MathSAT)
## Old DP vs. MathSAT – No Data

<table>
<thead>
<tr>
<th>Program</th>
<th>Old DP (s)</th>
<th>MathSAT (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST-REVERSE</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>LIST-ADD</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>ND-INSERT</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>ND-REMOVE</td>
<td>0.9</td>
<td>1.2</td>
</tr>
<tr>
<td>ZIP</td>
<td>17.3</td>
<td>27.3</td>
</tr>
<tr>
<td>CREATE-INSERT</td>
<td>14.8</td>
<td>15.6</td>
</tr>
<tr>
<td>CREATE-FREE</td>
<td>457.4</td>
<td>489.2</td>
</tr>
<tr>
<td>REMOVE-DOUBLY</td>
<td>24.3</td>
<td>33</td>
</tr>
<tr>
<td>REMOVE-CYCLIC-DOUBLY</td>
<td>15.6</td>
<td>15.7</td>
</tr>
<tr>
<td>LINUX-LIST-ADD</td>
<td>6.4</td>
<td>8.9</td>
</tr>
<tr>
<td>LINUX-LIST-ADD-TAIL</td>
<td>7.3</td>
<td>10</td>
</tr>
<tr>
<td>LINUX-LIST-DEL</td>
<td>24.7</td>
<td>25.2</td>
</tr>
</tbody>
</table>
## Old DP vs. MathSAT

<table>
<thead>
<tr>
<th>Program</th>
<th>Boolean</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Old DP</td>
<td>MathSAT</td>
</tr>
<tr>
<td>SOR TED-Z IP</td>
<td>22.8</td>
<td>46.2</td>
</tr>
<tr>
<td>SOR TED-INSERT</td>
<td>13.8</td>
<td>25.3</td>
</tr>
<tr>
<td>BUB B LE-S ORT *</td>
<td>11.1</td>
<td>16.5</td>
</tr>
<tr>
<td>BUB B LE-S ORT *</td>
<td>114.9</td>
<td>209</td>
</tr>
<tr>
<td>REMOVE-ELEMENTS</td>
<td>8.8</td>
<td>14.9</td>
</tr>
<tr>
<td>REMOVE-SEGMENT</td>
<td>2.2</td>
<td>10</td>
</tr>
<tr>
<td>SEARCH-AND-SET</td>
<td>5.3</td>
<td>10.8</td>
</tr>
<tr>
<td>SET-UNION *</td>
<td>1.4</td>
<td>2.2</td>
</tr>
<tr>
<td>CREATE-INSERT-DATA</td>
<td>39.7</td>
<td>47.3</td>
</tr>
<tr>
<td>INIT-LIST</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>INIT-LIST-V AR</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>INIT-CYCLIC</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>SOR TED-INSERT-DNODES</td>
<td>77.9</td>
<td>108.1</td>
</tr>
</tbody>
</table>
## New Examples

<table>
<thead>
<tr>
<th>Program</th>
<th>MathSAT (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAZY-SIMPLE</td>
<td>33.4</td>
</tr>
<tr>
<td>LAZY-SIMPLE-BACKW</td>
<td>2.2</td>
</tr>
<tr>
<td>INIT-INCREMENT</td>
<td>1.6</td>
</tr>
<tr>
<td>INIT-ADD</td>
<td>1.8</td>
</tr>
<tr>
<td>INIT-ADD-FLAG</td>
<td>1.4</td>
</tr>
<tr>
<td>INIT-MULT</td>
<td>1.8</td>
</tr>
</tbody>
</table>
Comparison

- Reasonable performance penalty
  - Could be improved with better predicate abstraction
    - Queries are conjunctions of literals - no boolean structure
    - Large number of small queries
    - No benefit from MathSAT’s enumeration heuristics
- Completely new examples we couldn’t handle before
  - Use combination of theories
Overview

- Introduction
- Logic for HMPs
- Delayed Theory Combination (DTC)
- Experiments

Conclusion
Conclusion

- Integrated the unbounded reachability theory into MathSAT
  - First solver that supports such a theory
  - Access to a rich set of theories MathSAT supports
  - Easy because of DTC
- The integration works
  - Verified HMP examples we couldn’t handle before
  - Reasonable performance penalty
- Available at [http://mathsat.itc.it](http://mathsat.itc.it)
The End

Thank you!
HMP Example

1: procedure INIT-ADD-FLAG(head, val)
2:   assume next*(head, t) ∧ next*(head, nil) ∧ ¬t = nil ∧ oldData = data(t) ∧ oldFlag = flag(t)
3:   curr := head;
4:   while ¬curr = nil do
5:     if ¬flag(curr) then
6:       data(curr) := data(curr) + val;
7:       flag(curr) := true;
8:     end if
9:     curr := next(curr);
10:   end while
11: assert next*(head, t) ∧ next*(head, nil) ∧ ¬t = nil ∧ flag(t) ∧ (oldFlag ∨ data(t) = oldData + val)
12: end procedure
MathSAT Query Example 1

VAR curr, t : H_NODE
VAR tmpi : INTEGER
CONST d

FORMULA data_int(t,d)=tmpi+2 &
    data_int(curr,d)=tmpi+5 & curr=t
MathSAT Query Example 2

VAR nil, curr, t, head : H_NODE
VAR tmpd : BOOLEAN
VAR tmpi, tmpi1 : INTEGER
CONST f
CONST d, d1

FORMULA nil=curr & data_bool(curr,d1) &
    tmpi1=data_int(t,d) & (tmpd<->data_bool(t,d1)) &
    star(head,t,f) & star(head,nil,f) & ~t=nil &
data_int(t,d)=tmpi1+tmpi & data_bool(t,d1) &
~tmpd
MathSAT Query Example 3

VAR x, curr, t, nil : H_NODE
VAR _true : BOOLEAN
CONST f, d, dp

FORMULA UPDATED_INT(x,10,d,dp)
FORMULA x=curr & curr=next(x,f) & star(x,t,f) & star(next(x,f),x,f) & _true & between(curr,t,x,f) & ~x=nil & between(x,t,curr,f) & star(t,curr,f) & t=x & data_int(t,dp)=10
Basic IRs

- IDENT: $x = x$
- REFLEX: $f^*(x, x)$
- TRANS1: $f(x) = y \Rightarrow f^*(x, y)$
- TRANS2: $f^*(x, y) \cdot f^*(y, z) \Rightarrow f^*(x, z)$
- FUNC: $x = z \Rightarrow f^*(y, z)$
- CYCLE$_k$: $f(x_k) = x_1 \Rightarrow f^*(x_k, y)$
- SCC: $f^*(x, y) \cdot f^*(y, x) \cdot f^*(z, x)$
- SHARE: $f(x) = z \Rightarrow f^*(x, y)$
- TOTAL: $f^*(x, y)$
- NOTEQNODES: $d(x) \Rightarrow \neg d(y)$
### Between IRs

<table>
<thead>
<tr>
<th>BTW1</th>
<th>BTW2</th>
<th>BTW3</th>
<th>BTW4</th>
<th>BTW5</th>
<th>BTW6</th>
<th>BTW7</th>
<th>BTW8</th>
<th>BTW9</th>
<th>BTW10</th>
<th>BTW11</th>
<th>BTW12</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f^<em>(x, y) ) ( f^</em>(y, z) ) ( f(z) = x ) ( \text{btwn}_f(x, y, z) )</td>
<td>( f^*(x, y) ) ( \text{btwn}_f(x, y, z) )</td>
<td>( f(x) = w ) ( \text{btwn}_f(x, y, z) ) ( x = y )</td>
<td>( \text{btwn}_f(x, y, z) ) ( \text{btwn}_f(x, z, y) ) ( y = z )</td>
<td>( f^<em>(x, y) ) ( f^</em>(y, z) ) ( f^*(z, x) )</td>
<td>( \text{btwn}_f(x, y, z) ) ( \text{btwn}_f(x, z, y) ) ( y = z )</td>
<td>( \text{btwn}_f(x, y, z) ) ( \text{btwn}_f(x, z, y) ) ( x = y ) ( x = z ) ( y = z )</td>
<td>( \text{btwn}_f(x, y, z) ) ( f(x) = z ) ( y = x ) ( y = z )</td>
<td>( f(z) = w ) ( \text{btwn}_f(x, y, w) ) ( f^*(x, z) ) ( y = w )</td>
<td>( \text{btwn}_f(x, y, z) ) ( f^*(x, w) ) ( y = z )</td>
<td>( \text{btwn}_f(x, y, z) ) ( \text{btwn}_f(x, w, z) ) ( f^*(x, w) ) ( y = z )</td>
<td>( \text{btwn}_f(v, x, z) ) ( \text{btwn}_f(v, u, x) ) ( \text{btwn}_f(u, x, y) )</td>
</tr>
</tbody>
</table>
Typical loop invariant
- Node $x$ between $\text{head}$ and $\text{iter}$ has $\text{data}(d, x) = \text{true}$

For cyclic lists the invariant can’t be expressed using previous logic (i.e. using $\text{reach}$)
- $\text{reach}(f, \text{head}, x) \land \text{reach}(f, x, \text{iter})$ doesn’t mean that $x$ is between $\text{head}$ and $\text{iter}$

$\text{between}(f, \text{head}, x, \text{iter})$ solves the problem