Deciding Unbounded Heaps in an SMT Framework

Zvonimir Rakamarić[1], Roberto Bruttomesso[2], Alan Hu[1], Alessandro Cimatti[2]

[1]University of British Columbia; [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

$c \in Constants$

$x \in DataVariables$ $\qquad$ $v \in PointerVariables$

$d, d' \in DataFields$ $\qquad$ $f, f' \in PointerFields$

$$
\begin{array}{lll}
NodeTerm & ::= & v \mid \mathrm{next}(f, NodeTerm) \\
DataTerm & ::= & c \mid x \mid \mathrm{data}(d, NodeTerm) \\
Atom & ::= & NodeTerm = NodeTerm \mid \\
 & & DataTerm = DataTerm \mid \\
 & & \mathrm{reach}(f, NodeTerm, NodeTerm) \mid \\
 & & \mathrm{between}(f, NodeTerm, NodeTerm, NodeTerm) \\
Literal & ::= & Atom \mid \sim Atom \mid \\
 & & \mathrm{update\_pfield}(f, NodeTerm, NodeTerm, f') \mid \\
 & & \mathrm{update\_dfield}(d, NodeTerm, DataTerm, d') \\
Formula & ::= & Lit. \mid Formula \wedge Formula \mid Formula \vee Formula
\end{array}
$$

Logic for HMPs

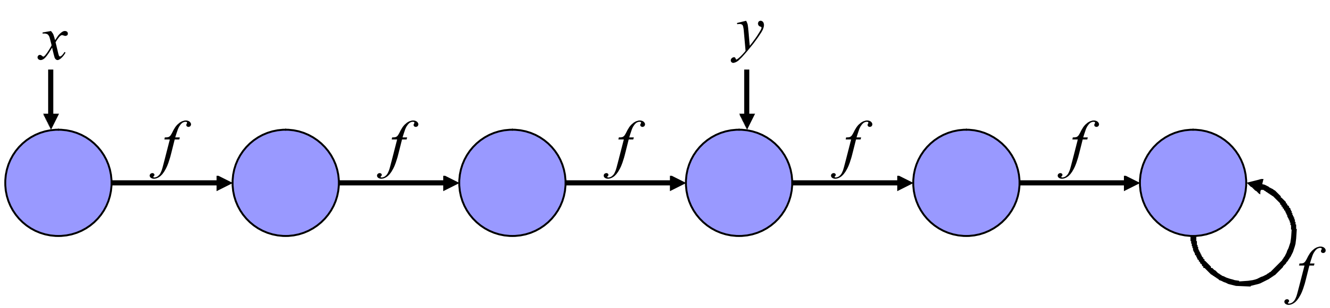
- Theory of Data Fields
  - { =, data, update_dfield }
  - Handled by MathSAT
    - EUF + update axioms
  - Currently only boolean and integer data fields
    - Easily extendable to other MathSAT's data types
- Theory of Unbounded Reachability over Heaps
  - { =, next, reach, between, update_pfield }
  - 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

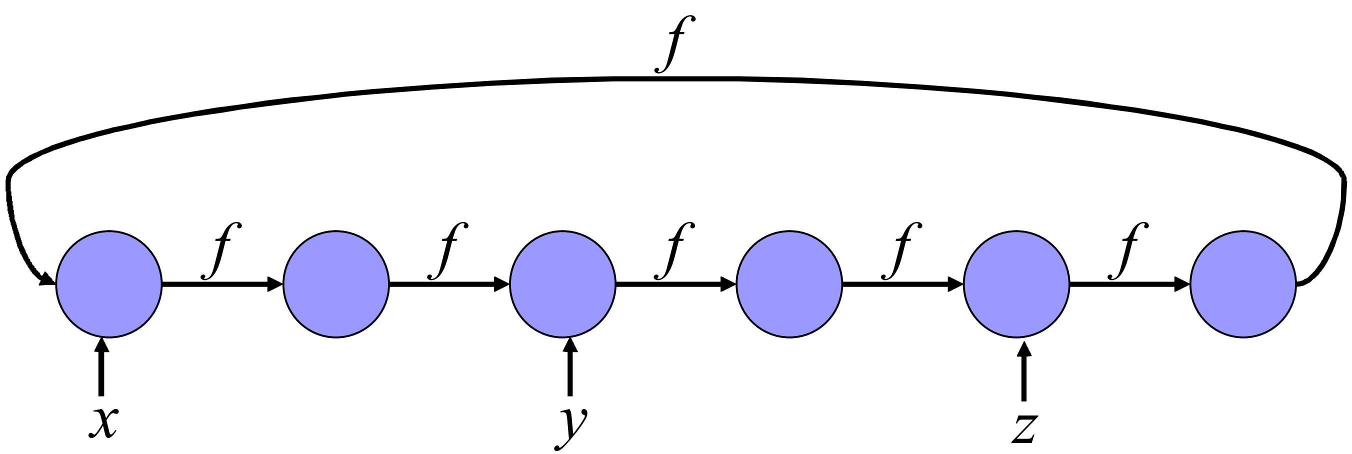
Unbounded Reachability

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



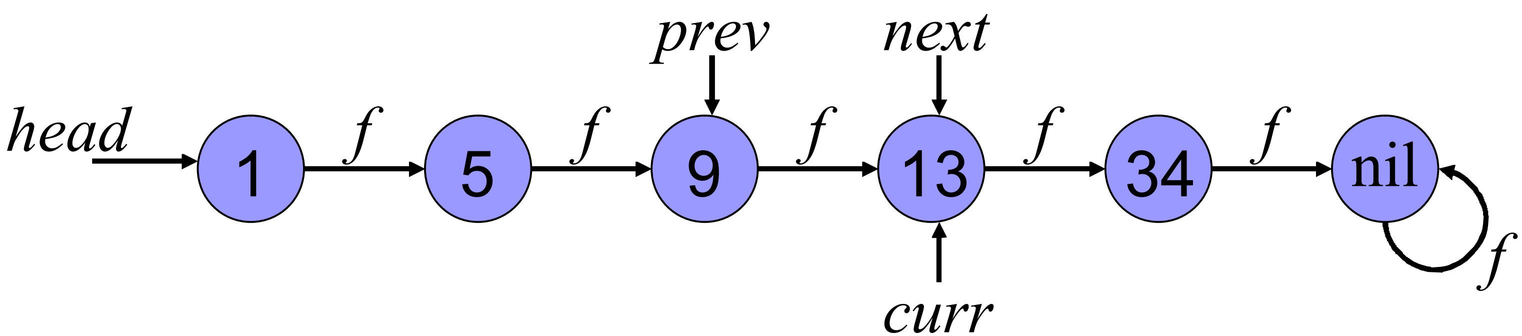
Between Atom

- 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



Example

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



Some true literals:

$next = curr$

$\mathrm{data}(d, curr) = 13$

$\mathrm{data}(d, curr) = \mathrm{data}(d, prev) + 4$

$\mathrm{reach}(f, head, prev)$

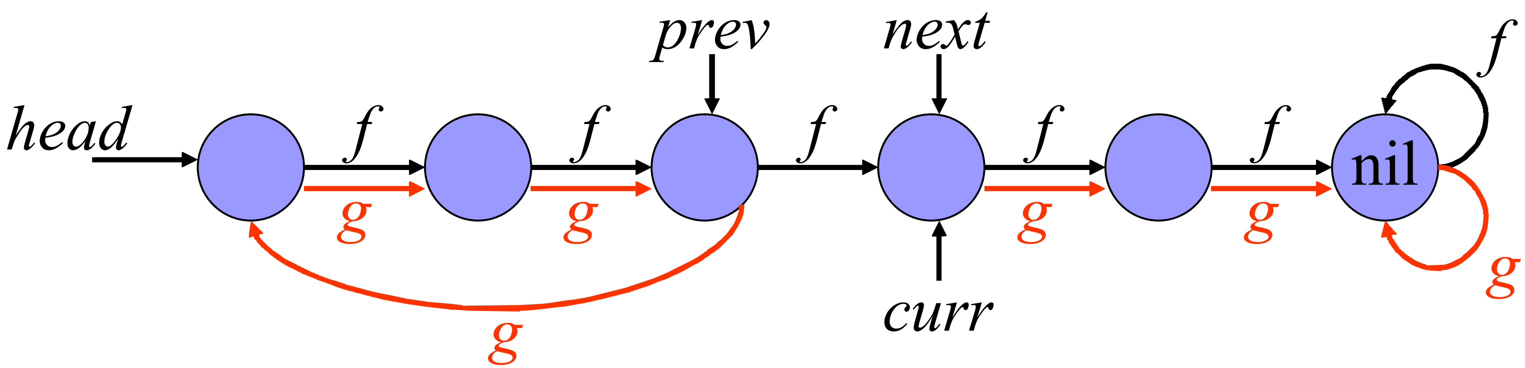
Some false literals:

$next = nil$

$\mathrm{data}(d, prev) > \mathrm{data}(d, curr)$

$\mathrm{reach}(f, next, prev)$

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

  $$\{d'(t) \approx v\} \cup \{d'(s) \approx d(s) \mid s \in NodeTerm, s \neq t\}$$

  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

Program	Old DP (s)	MathSAT (s)
LIST-REVERSE	0.2	0.2
LIST-ADD	0.1	0.1
ND-INSERT	0.5	0.6
ND-REMOVE	0.9	1.2
ZIP	17.3	27.3
CREATE-INSERT	14.8	15.6
CREATE-FREE	457.4	489.2
REMOVE-DOUBLY	24.3	33
REMOVE-CYCLIC-DOUBLY	15.6	15.7
LINUX-LIST-ADD	6.4	8.9
LINUX-LIST-ADD-TAIL	7.3	10
LINUX-LIST-DEL	24.7	25.2

Old DP vs. MathSAT

	Boolean		Integer
Program	Old DP	MathSAT	MathSAT
SORTED-ZIP	22.8	46.2	53.9
SORTED-INSERT	13.8	25.3	40.4
BUBBLE-SORT *	11.1	16.5	16.9
BUBBLE-SORT *	114.9	209	371.3
REMOVE-ELEMENTS	8.8	14.9	16.4
REMOVE-SEGMENT	2.2	10	10.3
SEARCH-AND-SET	5.3	10.8	13.7
SET-UNION *	1.4	2.2	5.8
CREATE-INSERT-DATA	39.7	47.3	53.6
INIT-LIST	0.1	0.1	0.1
INIT-LIST-VAR	0.2	0.4	0.4
INIT-CYCLIC	0.2	0.4	0.4
SORTED-INSERT-DNODES	77.9	108.1	175.7

New Examples

Program	MathSAT (s)
LAZY-SIMPLE	33.4
LAZY-SIMPLE-BACKW	2.2
INIT-INCREMENT	1.6
INIT-ADD	1.8
INIT-ADD-FLAG	1.4
INIT-MULT	1.8

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

The End

Thank you!

HMP Example

procedure INIT-ADD-FLAG($head, val$)
  assume $next^*(head, t) \wedge next^*(head, \mathrm{nil}) \wedge \neg t = \mathrm{nil} \wedge oldData = data(t) \wedge oldFlag = flag(t)$
  $curr := head$;
  while $\neg curr = \mathrm{nil}$ **do**
    if $\neg flag(curr)$ **then**
      $data(curr) := data(curr) + val$;
      $flag(curr) := \mathrm{true}$;
    end if
    $curr := next(curr)$;
  end while
  assert $next^*(head, t) \wedge next^*(head, \mathrm{nil}) \wedge \neg t = \mathrm{nil} \wedge flag(t) \wedge (oldFlag \vee data(t) = oldData + val)$
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

$$\frac{}{x=x}\textsc{Ident} \qquad \frac{}{f^*(x,x)}\textsc{Reflex} \qquad \frac{f(x)=y}{f^*(x,y)}\textsc{Trans1}$$

$$\frac{f^*(x,y) \quad f^*(y,z)}{f^*(x,z)}\textsc{Trans2} \qquad \frac{f(x)=y \quad f^*(x,z)}{x=z \qquad f^*(y,z)}\textsc{Func}$$

$$\frac{f(x_1)=x_2 \quad f(x_2)=x_3 \quad \cdots \quad f(x_k)=x_1 \quad f^*(x_1,y)}{y=x_1 \qquad y=x_2 \qquad \cdots \qquad y=x_k}\textsc{Cycle}_k$$

$$\frac{f^*(x,y) \quad f^*(y,x) \quad f^*(x,z)}{x=y \qquad f^*(z,x)}\textsc{Scc} \qquad \frac{f^*(x,y) \quad f^*(x,z)}{f^*(y,z) \qquad f^*(z,y)}\textsc{Total}$$

$$\frac{f(x)=z \quad f(y)=z \quad f^*(x,y) \quad f^*(y,x)}{x=y}\textsc{Share} \qquad \frac{d(x) \quad \neg d(y)}{\neg x=y}\textsc{NotEqNodes}$$

Between IRs

$$\frac{}{\mathsf{btwn}_f(x,x,x)}\ \textsc{BtwReflex}$$

$$\frac{f^*(x,y) \qquad f^*(y,z) \qquad f(z)=x}{\mathsf{btwn}_f(x,y,z)}\ \textsc{Btw1}$$

$$\frac{\mathsf{btwn}_f(x,y,z)}{\begin{array}{c} f^*(x,y) \\ f^*(y,z) \end{array}}\ \textsc{Btw2}$$

$$\frac{f(x)=w \qquad \mathsf{btwn}_f(x,y,z)}{\mathsf{btwn}_f(w,y,z) \qquad x=y}\ \textsc{Btw3}$$

$$\frac{\mathsf{btwn}_f(x,y,z) \qquad \mathsf{btwn}_f(x,z,y)}{y=z}\ \textsc{Btw4}$$

$$\frac{f^*(x,y) \qquad f^*(x,z)}{\mathsf{btwn}_f(x,y,z) \qquad \mathsf{btwn}_f(x,z,y)}\ \textsc{Btw5}$$

$$\frac{f^*(x,y) \qquad f^*(y,z) \qquad f^*(z,x)}{\begin{array}{ccccc} \mathsf{btwn}_f(x,y,z) & \mathsf{btwn}_f(x,z,y) & & & \\ \mathsf{btwn}_f(y,z,x) & \mathsf{btwn}_f(z,y,x) & x=y & x=z & y=z \\ \mathsf{btwn}_f(z,x,y) & \mathsf{btwn}_f(y,x,z) & & & \end{array}}\ \textsc{Btw6}$$

$$\frac{f^*(x,y)}{\begin{array}{c} \mathsf{btwn}_f(x,x,y) \\ \mathsf{btwn}_f(x,y,y) \end{array}}\ \textsc{Btw7}$$

$$\frac{\mathsf{btwn}_f(x,y,z) \qquad f(x)=z}{y=x \qquad y=z}\ \textsc{Btw8}$$

$$\frac{f(z)=w \qquad \mathsf{btwn}_f(x,y,w) \qquad f^*(x,z)}{\mathsf{btwn}_f(x,y,z) \qquad y=w}\ \textsc{Btw9}$$

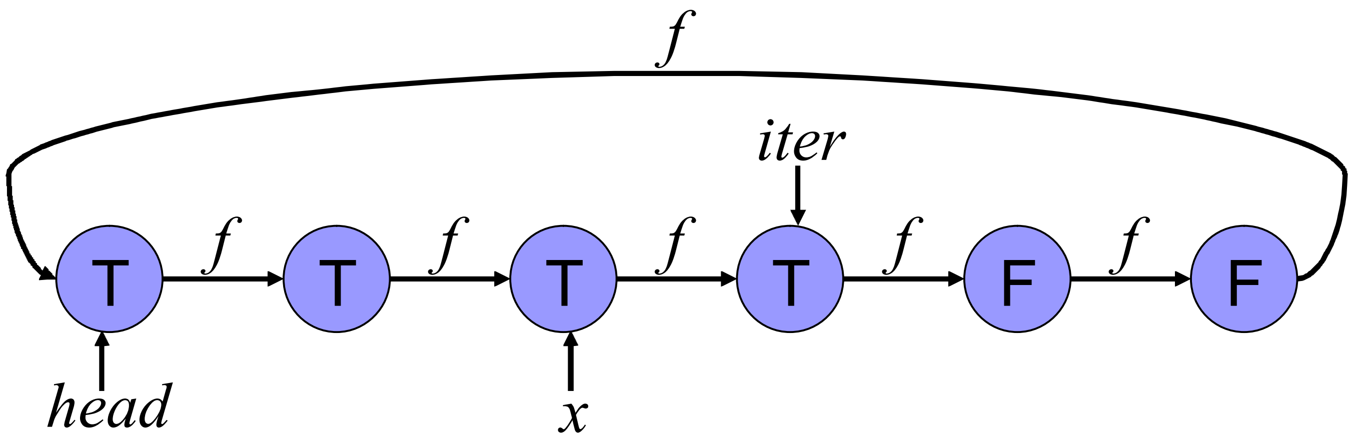
$$\frac{\mathsf{btwn}_f(x,y,z) \qquad \mathsf{btwn}_f(w,z,y) \qquad f^*(x,w)}{f^*(z,w) \qquad y=z}\ \textsc{Btw10}$$

$$\frac{\mathsf{btwn}_f(w,x,y) \qquad \mathsf{btwn}_f(w,y,z)}{\mathsf{btwn}_f(w,x,z)}\ \textsc{Btw11}$$

$$\frac{\mathsf{btwn}_f(v,u,x) \qquad \mathsf{btwn}_f(v,u,y) \qquad \mathsf{btwn}_f(u,x,y)}{\mathsf{btwn}_f(v,x,y)}\ \textsc{Btw12}$$

$$\frac{\mathsf{btwn}_f(x,y,z) \qquad \neg x=z}{\mathsf{btwn}_{f'}(x,y,z) \qquad \begin{array}{l} \mathsf{btwn}_f(x,\tau_1,z) \\ \neg \tau_1=z \end{array}}\ \textsc{UpdBtwn}$$

Between Example



- Typical loop invariant
  - Node x between $head$ and $iter$ has $\text{data}(d, x) = \text{true}$
- For cyclic lists the invariant can’t be expressed using previous logic (i.e. using reach)
  - $\text{reach}(f, head, x) \wedge \text{reach}(f, x, iter)$ doesn’t meen that x is between $head$ and $iter$
- $\text{between}(f, head, x, iter)$ solves the problem