Field Constraint Analysis

Thomas Wies

Max-Planck-Institut für Informatik, Saarbrücken, Germany
wies@mpi-inf.mpg.de

VMCAI 2006

Joint work with

Viktor Kuncak, Patrick Lam, Andreas Podelski, and Martin Rinard

Motivation

Shape Analysis

Verify consistency properties of linked data structures.

acyclicity, heap reachability, sharing, . . .

Motivation

Shape Analysis

Verify consistency properties of linked data structures.

acyclicity, heap reachability, sharing, . . .

Conflicting objectives

1. generality: support a large class of data structures
2. predictability: provide completeness guarantees
3. degree of automation: synthesize loop invariants
4. scalability: verify data structures in the context of larger programs

Motivation

Shape Analysis

Verify consistency properties of linked data structures.

acyclicity, heap reachability, sharing, . . .

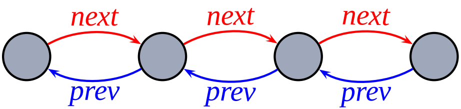
Conflicting objectives

1. generality: support a large class of data structures
2. predictability: provide completeness guarantees
3. degree of automation: synthesize loop invariants
4. scalability: verify data structures in the context of larger programs

Reduce verification problem to problem of reasoning over logical constraints, e.g. in MSOL over trees.

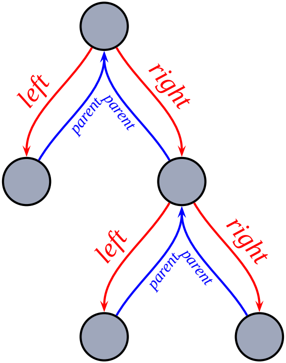
Backbone and Derived Fields

Doubly-linked lists



Backbone fields

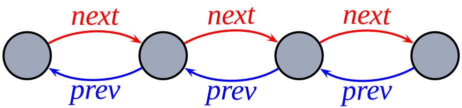
Derived fields



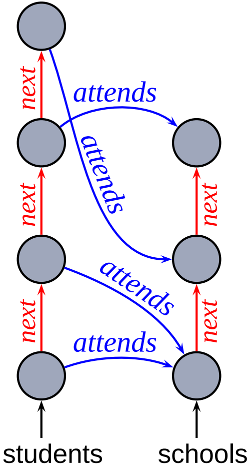
Trees with parent pointers

Backbone and Derived Fields

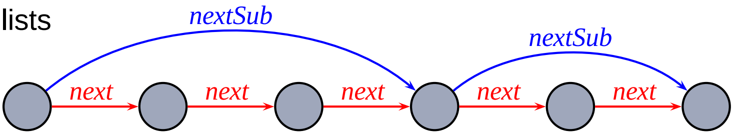
Doubly-linked lists



Backbone fields
Derived fields



Skip lists



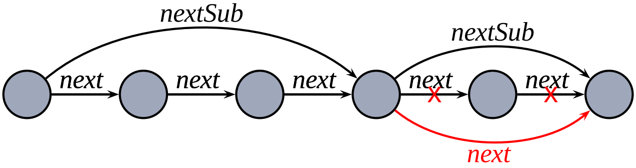
Road Map

MSOL		MSOL
lists		skip lists
$next$	$\hookleftarrow$	$next, nextSub$
decidable		

$$\models^{!} G' \qquad\qquad \models^{?} G$$

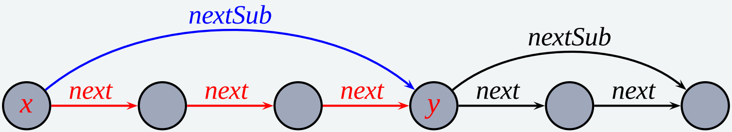
$$G \equiv \text{acyclic}(next) \rightarrow \underbrace{\text{wlp}(c, \text{acyclic}(next))}_{\text{may contain } nextSub}$$

Effect of c



Field Constraints

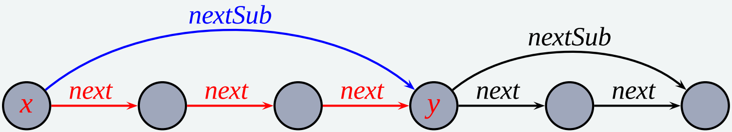
Example



Field constraint: $\forall x\, y\, .\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

Field Constraints

Example



Field constraint: $\forall x\, y\, .\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

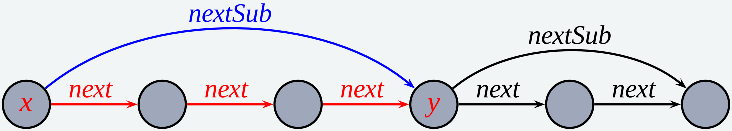
Field constraint for a derived field f:

$$\begin{aligned} & \forall x\, y\, .\, f(x) = y \;\rightarrow\; F(x, y) \\ \iff \quad & \forall x\, .\, F(x, f(x)) \end{aligned}$$

F may be arbitrary formula over backbone fields relating x and $f(x)$.

Field Constraints

Example



Field constraint: $\forall x\, y\, .\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

Idea

Use field constraints to eliminate derived field occurrences in query.

Eliminating Derived Fields

Example

Field Constraint: $\forall x\, y\,.\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

Query: $x_1 = x_2 \;\rightarrow\; nextSub(x_1) = nextSub(x_2)$

Eliminating Derived Fields

Example

Field Constraint: $\forall x\, y\,.\, nextSub(x) = y \;\rightarrow\; next^{+}(x,y)$

Query: $x_1 = x_2 \;\rightarrow\; nextSub(x_1) = nextSub(x_2)$

Idea

Replace derived fields by approximating formula.

Soundness

Result of elimination must be stronger or equivalent.

➜ Replacing negative occurrences is sound.
➜ Replacing positive occurrences is not sound.

➜ Rewrite all occurrences into negative ones.

Eliminating Derived Fields

Example

Field Constraint: $\forall x\, y\, .\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

Query: $x_1 = x_2 \;\rightarrow\; nextSub(x_1) = nextSub(x_2)$

$\forall y_1\, .\, x_1 = x_2 \wedge nextSub(x_1) = y_1 \;\rightarrow\; y_1 = nextSub(x_2)$

Eliminating Derived Fields

Example

Field Constraint: $\forall x\, y \,.\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

Query: $x_1 = x_2 \;\rightarrow\; nextSub(x_1) = nextSub(x_2)$

$$\forall y_1\, y_2 \,.\, x_1 = x_2 \wedge nextSub(x_1) = y_1 \wedge nextSub(x_2) = y_2 \;\rightarrow\; y_1 = y_2$$

Final query: $\forall y_1\, y_2 \,.\, x_1 = x_2 \wedge next^{+}(x_1, y_1) \wedge next^{+}(x_2, y_2) \;\rightarrow\; y_1 = y_2$

Eliminating Derived Fields

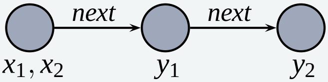
Example

Field Constraint: $\forall x\, y\,.\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

Query: $x_1 = x_2 \;\rightarrow\; nextSub(x_1) = nextSub(x_2)$

Final query: $\forall y_1\, y_2\,.\, x_1 = x_2 \wedge next^{+}(x_1, y_1) \wedge next^{+}(x_2, y_2) \;\rightarrow\; y_1 = y_2$

Counterexample:



Eliminating Derived Fields

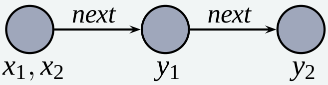
Example

Field Constraint: $\forall x\, y\,.\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

Query: $x_1 = x_2 \;\rightarrow\; nextSub(x_1) = nextSub(x_2)$

Final query: $\forall y_1\, y_2\,.\, x_1 = x_2 \wedge next^{+}(x_1, y_1) \wedge next^{+}(x_2, y_2) \;\rightarrow\; y_1 = y_2$

Counterexample:



➜ Keep track of equalities between replaced terms.

Eliminating Derived Fields

Example

Field Constraint: $\forall x\, y\,.\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

Query: $x_1 = x_2 \;\rightarrow\; nextSub(x_1) = nextSub(x_2)$

Final query: $\forall y_1\, y_2\,.\, x_1 = x_2 \wedge next^{+}(x_1, y_1) \wedge next^{+}(x_2, y_2)$
$\wedge\, (x_1 = x_2 \;\rightarrow\; y_1 = y_2) \;\rightarrow\; y_1 = y_2$

Eliminating Derived Fields

Example

Field Constraint: $\forall x\, y\, .\, nextSub(x) = y \;\rightarrow\; next^{+}(x, y)$

Query: $x_1 = x_2 \;\rightarrow\; nextSub(x_1) = nextSub(x_2)$

Final query: $\forall y_1\, y_2\, .\, x_1 = x_2 \wedge next^{+}(x_1, y_1) \wedge next^{+}(x_2, y_2)$
$\wedge\, (x_1 = x_2 \;\rightarrow\; y_1 = y_2) \;\rightarrow\; y_1 = y_2$

Final query is valid.

Elimination Algorithm

$$
\begin{array}{l}
\textbf{proc } \mathsf{Elim}(G) = \mathsf{elim}(G, \emptyset) \\
\textbf{proc } \mathsf{elim}(G : \text{formula in negation normal form}; \\
\qquad\qquad K : \text{set of (variable,field,variable) triples}): \\
\quad \textbf{let } T = \{f(t) \in \mathsf{Ground}(G).\ f \in \mathsf{Derived}(G) \land \mathsf{Derived}(t) = \emptyset\} \\
\quad \textbf{if } T \neq \emptyset \textbf{ do} \\
\qquad \textbf{choose } f(t) \in T \\
\qquad \textbf{choose } x, y \text{ fresh first-order variables} \\
\qquad \textbf{let } F = \mathsf{FC}(f) \\
\qquad \textbf{let } F_1 = F(x,y) \land \bigwedge_{(x_i,f,y_i) \in K} (x = x_i \ \rightarrow\ y = y_i) \\
\qquad \textbf{let } G_1 = G[f(t) := y] \\
\qquad \textbf{return } \forall x.\ x = t \ \rightarrow\ \forall y.\ (F_1 \ \rightarrow\ \mathsf{elim}(G_1, K \cup \{(x,f,y)\})) \\
\quad \textbf{else case } G \textbf{ of} \\
\quad \mid \quad Qx.\ G_1 \textbf{ where } Q \in \{\forall, \exists\}: \\
\qquad\quad \textbf{return } Qx.\ \mathsf{elim}(G_1, K) \\
\quad \mid \quad G_1 \ op\ G_2 \textbf{ where } op \in \{\land, \lor\}: \\
\qquad\quad \textbf{return } \mathsf{elim}(G_1, K) \ op\ \mathsf{elim}(G_2, K) \\
\quad \mid \quad \textbf{else return } G
\end{array}
$$

Soundness

Theorem

Field constraint analysis is sound.

Completeness?

Completeness

Requirement: $\models \mathsf{Elim}(G) \leftrightarrow G$

In general incomplete.

Critical part of derived field elimination:
Replacement of derived field by approx. formula.

$$\forall x\, y\,.\, f(x) = y \;\rightarrow\; F(x, y)$$

Completeness for Interesting Special Cases

Deterministic Field Constraints + General Formulas

$\forall x\, y\, .\, f(x) = y \leftrightarrow F(x, y)$

➜ Subsumes previous approaches

Completeness for Interesting Special Cases

Deterministic Field Constraints + General Formulas

$\forall x\, y\,.\, f(x) = y \leftrightarrow F(x, y)$

➜ Subsumes previous approaches

General Field Constraints + Quite Nice Formulas

Quite nice formulas: all derived field occurrences $f(t)$ satisfy

free variables in t are outermost universally quantified (or free in G)

➜ interesting in practice, because:

- field constraints itself are quite nice
- quite nice formulas are closed under wlp.

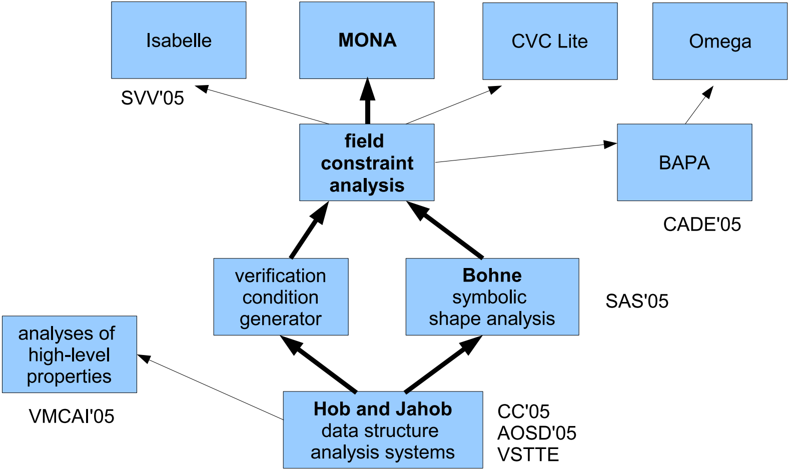
Preservation of Field Constraints

```
...
sprev = root; scurrent = root.nextSub;
 while
"(∀ x y. nextSub x = y  →  next+ x y) ∧
 scurrent = nextSub sprev ∧
 next* root scurrent ∧ ..."
((scurrent != null) && (scurrent.v < v)) {
   sprev = scurrent;
   scurrent = scurrent.nextSub;
 }
 ...
```

Complete method for checking preservation of field constraints if

loop invariants are field constraints
conjoined with other quite nice formulas.

Deployment in Hob



Hob Modules

```
impl module Skiplist {
  format Entry {
    v : int;
    next, nextSub : Entry;
  }
  var root : Entry;

  proc add(e:Entry) {
    int v = e.v;
    Entry sprev = root, scurrent = root.nextSub;
    while ((scurrent != null) && (scurrent.v < v)) {
      sprev = scurrent; scurrent = scurrent.nextSub;
    }
    Entry prev = sprev, current = sprev.next;
    while ((current != scurrent) && (current.v < v)) {
      prev = current; current = current.next;
    }
    e.next = current; prev.next = e;
    choice { sprev.nextSub = e; e.nextSub = scurrent; }
         | { e.nextSub = null; }
}
```

```
spec module Skiplist {
  format Entry;
  specvar Content : Entry set;

  proc add(e:Entry)
    requires card(e) = 1 & not (e in Content)
    modifies Content
    ensures Content' = Content + e';
}
```

```
abst module Skiplist {
  use plugin "Bohne";

  Content = {x : Entry | "next+ root x"};
  invariant "∀ x y. nextSub x = y → next+ x y";

  ...
}
```

Bohne Plugin

Symbolic shape analsysis for loop invariant inference.

Bohne Plugin

Boolean heaps [1,2]:

$\forall x \,.\, \bigvee_i \bigwedge_j p_{i,j}(x).$

Infered loop invariants:

disjunctions of Boolean heaps.

(Heap) predicate abstraction:

$$p_1(x) \wedge \ldots \wedge p_n(x) \overset{?}{\models} \mathsf{wlp}(c, p(x)).$$

Decision procedure is black box.

➜ Use field constraint analysis.

```
abst module Skiplist {
  use plugin "Bohne";

  Content = {x : Entry | "next+ root x"};
  invariant "∀ x y. nextSub x = y  →  next+ x y";

  proc add {
    p_1 = {x : Entry | "∃ y. next y = x"};
    p_2 = {x : Entry | "next* current x"};
    p_3 = {x : Entry | "next* scurrent x"};
    p_4 = {x : Entry | "next* sprev x"};
    p_5 = {x : Entry | "next x = null"};
    p_6 = "nextSub sprev = scurrent";
    p_7 = "next prev = current";
  }
}
```

[1] Wies, Symbolic Shape Analysis. Master's Thesis, 2004.
[2] Podelski, Wies. Boolean Heaps. In SAS, 2005.

Some Results

Analyzed Data Structures

- singly-linked lists
- doubly-linked lists (with iterators)
- binary trees (with parent pointers)
- two-level skip lists

Analyzed Programs

- minesweeper game
- process scheduler
- web server

Hob project homepage:
`http://hob.csail.mit.edu/`

Related Work

Previous Approaches

- Graph Types: Klarlund and Schwarzbach (*POPL* 1993)
- PALE: Møller and Schwarzbach (*PLDI* 2001)
- Structure Simulation:
  Immerman, Rabinovich, Reps, Sagiv, Yorsh (*CAV* 2004)

Shape Analysis

- TVLA: Sagiv, Reps, Wilhelm (*TOPLAS* 2002),
  ...
- Symb. computing most-precise abstr. op. for shape analysis:
  Yorsh, Reps, Sagiv (TACAS 2004)

Conclusion

Field Constraint Analysis

- enables application of decidable logics to verify data structures that are beyond the scope of these logics
- is applicable to data structures where fields cross-cut a backbone in arbitrary ways
- is always sound
- is complete for a class of formulas that is of practical interest.

Ongoing and Future Work

- more efficient decision procedures for list backbones
- user-defined backbones (e.g. for cyclic lists)
- combinations with other decision procedures

➜ Jahob project.