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.
Motivation

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
Motivation

Road Map

MSOL

lists

next

decidable

\[ ! \models G' \]

\[ G \equiv \text{acyclic}(\text{next}) \rightarrow \text{wlp}(c, \text{acyclic}(\text{next})) \]

may contain nextSub

Effect of \( c \)

MSOL

skip lists

next, nextSub

\[ ? \models G \]
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 $f$:

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

$$\iff \forall x. \ F(x, f(x))$$

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

Field Constraints

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.

\( \rightarrow \) Replacing negative occurrences is sound.
\( \rightarrow \) Replacing positive occurrences is not sound.

\( \rightarrow \) Rewrite all occurrences into negative ones.
Eliminating Derived Fields

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

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

$\forall y_1 . x_1 = x_2 \land nextSub(x_1) = y_1 \rightarrow y_1 = nextSub(x_2)$
Eliminating Derived Fields

**Example**

**Field Constraint:** \( \forall x \ y \ . \ nextSub(x) = y \rightarrow next^+(x, y) \)

**Query:** \( x_1 = x_2 \rightarrow nextSub(x_1) = nextSub(x_2) \)

\[ \forall y_1 \ y_2 \ . \ x_1 = x_2 \land nextSub(x_1) = y_1 \land nextSub(x_2) = y_2 \rightarrow y_1 = y_2 \]

**Final query:** \( \forall y_1 \ y_2 \ . \ x_1 = x_2 \land next^+(x_1, y_1) \land next^+(x_2, y_2) \rightarrow y_1 = y_2 \)
Eliminating Derived Fields

Example

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

Query: \( x_1 = x_2 \ \rightarrow \ \text{nextSub}(x_1) = \text{nextSub}(x_2) \)

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

Counterexample:

\[ \text{next} \]

\( x_1, x_2 \rightarrow y_1 \rightarrow y_2 \)
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: \( \forall y_1 \ y_2 \ . \ x_1 = x_2 \ \land \ next^+(x_1, y_1) \ \land \ next^+(x_2, y_2) \ \rightarrow \ y_1 = y_2 \)

Counterexample:

\[ x_1, x_2 \xrightarrow{\text{next}} y_1 \xrightarrow{\text{next}} y_2 \]

\( \Rightarrow \) Keep track of equalities between replaced terms.
Eliminating Derived Fields

Example

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

Query: \( x_1 = x_2 \rightarrow \text{nextSub}(x_1) = \text{nextSub}(x_2) \)

Final query: \( \forall y_1 \ y_2 . \ x_1 = x_2 \land \text{next}^+(x_1, y_1) \land \text{next}^+(x_2, y_2) \land (x_1 = x_2 \rightarrow y_1 = y_2) \rightarrow y_1 = y_2 \)
Eliminating Derived Fields

Example

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

Query: \( x_1 = x_2 \rightarrow \text{nextSub}(x_1) = \text{nextSub}(x_2) \)

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

Final query is valid.
proc Elim(G) = elim(G, ∅)
proc elim(G : formula in negation normal form;
    K : set of (variable,field,variable) triples):
    let $T = \{ f(t) \in \text{Ground}(G). f \in \text{Derived}(G) \land \text{Derived}(t) = \emptyset \}$
    if $T \neq \emptyset$
        choose $f(t) \in T$
        choose $x, y$ fresh first-order variables
        let $F = FC(f)$
        let $F_1 = F(x, y) \land \bigwedge_{(x_i, f, y_i) \in K}(x = x_i \rightarrow y = y_i)$
        let $G_1 = G[f(t) := y]$
        return $\forall x. \ x = t \rightarrow \forall y. \ (F_1 \rightarrow \text{elim}(G_1, K \cup \{(x, f, y)\}))$
    else case $G$ of
        | $Qx. \ G_1$ where $Q \in \{\forall, \exists\}$:
            return $Qx. \ \text{elim}(G_1, K)$$
        | $G_1 \ op G_2$ where $op \in \{\land, \lor\}$:
            return $\text{elim}(G_1, K) \ op \text{elim}(G_2, K)$$
        | else return $G$
Soundness

Theorem

*Field constraint analysis is sound.*

Completeness?
Completeness

Requirement: $\models $ 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

- Isabelle
- MONA
- CVC Lite
- Omega
- field constraint analysis
- verification condition generator
- Bohne symbolic shape analysis
- Hob and Jahob data structure analysis systems

- BAPA
- CADE'05
- SAS'05
- CC'05
- AOSD'05
- VSTTE

- analyses of high-level properties
- SVV'05
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 analysis for loop invariant inference.
Bohne Plugin

Boolean heaps [1,2]:
\[ \forall x. \bigvee_i \bigwedge_j p_{i,j}(x). \]

Inferred loop invariants:
disjunctions of Boolean heaps.

(Heap) predicate abstraction:
\[ p_1(x) \land \ldots \land p_n(x) \models wlp(c, p(x)). \]

Decision procedure is black box.
⇒ Use field constraint analysis.

\begin{verbatim}
abst module Skiplist {
   use plugin "Bohne";

   Content = {x : Entry | "next^+ root x"};
   invariant "\forall x y. nextSub x = y \rightarrow next^+ x y";

   proc add {
      p_1 = {x : Entry | "\exists 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";
   }
}
\end{verbatim}

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**.