Modelling, Specification and Formal Analysis of Complex Software Systems
Precise Static Analysis of Programs with Dynamic Memory

Mihaela Sighireanu

IRIF, University Paris Diderot & CNRS

VTSA 2015
Formal Semantics for Procedure Call

Stack  \[ \text{Stacks} \triangleq \left[ \left( \text{CP} \times \text{P} \times (\text{DV} \rightarrow \text{D} \cup \text{RV} \rightarrow \text{L}) \right)^\ast \right] \ni S \]

Memory  \[ \text{Mem} \triangleq \text{Stacks} \times \text{Heaps} \ni m \]

Configurations  \[ \text{Config} \triangleq \text{CP} \times (\text{Mem} \cup \{\text{merr}\}) \ni C \]

\[
\forall v_i \in \text{vin}. (S, H) \vdash v_i \rightsquigarrow c_i \neq \text{merr} \\
(\ell, (S, H)) \vdash v = P(\text{vin}, \text{vout}) \rightsquigarrow (\text{start}_P, (\text{push}(S, \ell + 1, P, v, \text{vout}, c_i, l_{v_P}), H))
\]

\[
\text{top}(S) = (\ell, P, v, \ldots) \quad (S, H)(v') = c \\
(\ell', (S, H)) \vdash \text{return } v' \rightsquigarrow (\ell, (\text{pop}(S), H)[v \leftarrow c])
\]

Another source of infinity is the unbounded stack that usually stores locations in the heap.
**Inter-procedural Analyses**

**Aim**

Compute an abstraction of the relation between the input and output configurations of a procedure, *i.e.* the procedure summary or contract.

Context sensitive: the summary depends on an abstraction of the calling stack

“If p is called before q, it returns 0, otherwise 1.”

→ insight on the full program behaviour, expressive

→ analysis done for each call point

Context insensitive: the summary is independent of the calling stack

“If p is called it returns 0 or 1.”

→ insight on the procedure behaviour, but less precise

→ analysis done independently of callers
Main steps:

Case 1: Compute summary information for each procedure ... at each calling point with “equivalent stack” runs

Case 2: Use summary information at procedure calls... ... if the abstraction of reaching stack fits the already computed ones
**Context-Sensitive Approaches**

**Main steps:**

- **Case 1:** Compute summary information for each procedure... at each calling point with “equivalent stack” runs

- **Case 2:** Use summary information at procedure calls... if the abstraction of reaching stack fits the already computed ones

**Classic approaches for summary computation:**

- **Functional approach:** [Sharir&Pnueli,81],[Knoop&Steffen,92]
  Summary is a function mapping abstract input to abstract output

- **Relational approach:** [Cousot&Cousot,77]
  Summary is a relation between input and output

- **Call string approach:** [Sharir&Pnueli,81], [Khedker&Karkare,08]
  Maps string abstractions of the call stack to abstract configs.
**Aim**

Compute a function $\text{summary}_P : \mathbf{CP} \rightarrow (A_H \rightarrow A_{mH})$ mapping each control point of the procedure $q \in \mathbf{CP}$ to a function which associates every $(G_0, W_0)$ abstract heap reachable at $\text{start}_P$ to the abstract heap $(G_q, W_q)$ reachable at $q$.

```c
int length(list* l) {
    int len = 0;
    if (l == NULL)
        len=0;
    else {
        len=1+length(l->next);
    }
    return len;
}
```

<table>
<thead>
<tr>
<th>q</th>
<th>$(G_0, W_0)$</th>
<th>$(G_q, W_q)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$l = \emptyset$</td>
<td>$l = \emptyset$</td>
</tr>
<tr>
<td></td>
<td>$ls^+(l, \emptyset)$</td>
<td>$ls^+(l, \emptyset)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>8</td>
<td>$l = \emptyset$</td>
<td>$\text{ret} = 0 \land l = \emptyset$</td>
</tr>
<tr>
<td></td>
<td>$ls^+(l, \emptyset)$</td>
<td>$\text{ret} \geq 1 \land ls^+(l, \emptyset)$</td>
</tr>
</tbody>
</table>
Problem
The local heap of a procedure may be accessed from the stack bypassing the actual parameters.

Bad Consequence
Context sensitive analyses shall track also these interferences!
**Observation**
In a large class of programs with procedure calls, the local heap is reachable from the stack by passing through the actual parameters.

**Consequence**
For this class, the computation of summaries is compositional.
Cut-point Free Programs

Definition

A call is **cut point free** if all local heap cut nodes are reachable from the stack through the procedure parameters. A cut point free program has only cut point free procedure calls.
Let $V$ be the set of formal parameters and local variables.

**Definition**

A concrete inter-procedural configurations is a pair of heap configurations $(H_0, H_q)$ where:

- $H^0$ is the local heap at $\text{start}_P$ over a new vocabulary $V^0$ → similar to old notation in JML
- $H$ is the heap at the control point $q$ of the procedure over $V \cup \{\text{\$ret}\}$

**Definition**

A concrete procedure summary is the set $\{(H^0, H_{end_P})\}$. 
1. Introduction
2. Formal Models and Semantics for IMPR
3. Foundations of Static Analysis by Abstract Interpretation
4. Application: Programs with Lists and Data
5. Application: Decision Procedures by Static Analysis
6. Elements of Inter-procedural Analysis
7. Application: Programs with Lists, Data, and Procedures
8. Extension: Programs with Complex Data Structures
9. Extension: Programs with Inductive Data Structures
Programs with Lists and Data

— Inter-procedural Analysis —

joint work with A. Bouajjani, C. Drăgoi, C. Enea

PLDI’11
Running Example: Quicksort on Lists (with Copy)

Split w.r.t. pivot p

\[ \leq p \quad \quad \quad > p \]

\[ \text{p} \]
Running Example: Quicksort on Lists (with Copy)

\[\text{left} = \text{quicksort(left)}\]
Running Example: Quicksort on Lists (with Copy)
Running Example: Quicksort on Lists (with Copy)
\[
\alpha((H^0, H)) \triangleq (G^0 \ast G, W^0 \land W)
\]
Analysing Quicksort: $A^U$ Domain

\[
\text{len}(x^0) = 1 + \text{len}(p) + \text{len(left)} + \text{len(right)}
\]

\[
\forall i \in \text{left} \Rightarrow \text{left}[i] \leq p[0]
\]

\[
\forall i \in \text{right} \Rightarrow \text{right}[i] > p[0]
\]
Analysing Quicksort: $A^M$ Domain

\[
\text{ms(left)} + \text{ms(p)} + \text{ms(left)} = \text{ms}(x^0)
\]
Quicksort: Loss of Precision

\[ \text{left} = \text{quicksort}(\text{left}) \]

\[ \text{input} \rightarrow \text{equal multisets} \rightarrow \text{sorted} \rightarrow \text{output} \]

\[ \text{left} \leq p \quad p \quad > p \quad \text{right} \]
left = quicksort(left)
Quicksort: Loss of Precision

\[ \text{left} = \text{quicksort} (\text{left}) \]

\[
\begin{align*}
\text{sorted} & \quad \land \quad \leq \ p \quad \land \quad \text{equal multisets} \quad (\square, \square) \\
\text{sorted} & \quad \land \quad \leq \ p (\square) \quad \land \quad \leq \ p (\square)
\end{align*}
\]
Quicksort: Loss of Precision

left = quicksort(left)

\[
\text{strengthen}(\text{sorted}(\square) \land \leq p(\square), \ \text{equal multisets}(\square, \square))
\]

\[
\text{sorted}(\square) \land \leq p(\square) \land \leq p(\square)
\]
STRENGTHEN PROCEDURE

1. **unfold** a bounded length prefix of $n$ and $m$

   \[ \psi^U \in \mathcal{A}_U \]

   \[ \psi^W \in \mathcal{A}_W \]

   \[ n \]

   \[ m \]

   \[ \text{unfold} (\psi^U) \]

   \[ \text{unfold} (\psi^W) \]

2. **intersect** the constraints on the prefixes

   \[ \text{unfold} (\psi^U) \wedge \varphi \]

   \[ \text{unfold} (\psi^W) \]

3. **fold** the prefixes and collect the information using universal formulas

   \[ \text{fold w.r.t. patterns in } \mathcal{A}_U \]

   \[ \text{fold} \]
**Strengthen Procedure: Example**

\[ \forall y. \ y \in n \Rightarrow d(y) \leq d(p) \]

\[ n \]

\[ m \]

\[ ms(n) = ms(m) \]

1. **unfold** a bounded length prefix of \( n \) and \( m \)

\[ d(q) \leq d(p) \]

\[ \forall y. \ y \in n' \Rightarrow d(y) \leq d(p) \]

\[ q \]

\[ n' \]

\[ 0 \ 1 \ 2 \ 3 \ldots \]

\[ ms(n') + ms(q) = ms(m') + ms(r) \]

\[ r \]

\[ m' \]

\[ 0 \ 1 \ 2 \ 3 \ldots \]
**Strengthen Procedure: Example**

\[
d(q) \leq d(p) \\
\forall y. y \in n' \Rightarrow d(y) \leq d(p)
\]

2. intersect the constraints on the prefixes

\[
\text{ms}(r) \subseteq \text{ms}(n') \quad \forall y. y \in n' \Rightarrow d(y) \leq d(p)
\]

\[
d(r) \leq d(p)
\]

\[
\text{ms}(n') + \text{ms}(q) = \text{ms}(m') + \text{ms}(r)
\]
**Strengthen Procedure: Example**

\[ d(q) \leq d(p) \]
\[ \forall y. y \in n' \Rightarrow d(y) \leq d(p) \]

2. **intersect** the constraints on the prefixes

\[ ms(n') + ms(q) = ms(m') + ms(r) \]

\[ ms(r) = ms(q) \quad d(q) \leq d(p) \]
\[ d(r) \leq d(p) \]
**Strengthen Procedure: Example**

\[ d(q) \leq d(p) \]
\[ \forall y. y \in n' \Rightarrow d(y) \leq d(p) \]

2. **intersect** the constraints on the prefixes

\[ ms(n') + ms(q) = ms(m') + ms(r) \]

\[ ms(r) = ms(q) \quad d(q) \leq d(p) \]
\[ d(r) \leq d(p) \]
### Experimental Results

<table>
<thead>
<tr>
<th>class</th>
<th>fun</th>
<th>nesting (loop, rec)</th>
<th>$A_M$ t (s)</th>
<th>$P$</th>
<th>$A_U$ t (s)</th>
<th>Examples of summaries synthesized</th>
</tr>
</thead>
<tbody>
<tr>
<td>sll</td>
<td>create</td>
<td>(0, -)</td>
<td>&lt; 1</td>
<td>$P =, P_1 &lt; 1$</td>
<td>$P^#_U (create(&amp;x, \ell))$: $\text{hd}(x) = 0 \land \text{len}(x) = \ell \land \forall y \in t_1(x) \Rightarrow x[y] = 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>addfst</td>
<td>-</td>
<td>&lt; 1</td>
<td>$P =$</td>
<td>$P^#_U (addfst(x))$: $\text{hd}(x) = x \land \forall y \in t_1(x) \Rightarrow x[y] = y$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>addlst</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =$</td>
<td>$P^#_U (addlst(x))$: $\text{hd}(x) = x \land \forall y \in t_1(x) \Rightarrow x[y] = y$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delfst</td>
<td>-</td>
<td>&lt; 1</td>
<td>$P =$</td>
<td>$P^#_U (delfst(x))$: $\text{hd}(x) = x \land \forall y \in t_1(x) \Rightarrow x[y] = y$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dellst</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =$</td>
<td>$P^#_U (dellst(x))$: $\text{hd}(x) = x \land \forall y \in t_1(x) \Rightarrow x[y] = y$</td>
<td></td>
</tr>
<tr>
<td>map</td>
<td>init(v)</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =, P_1 &lt; 1$</td>
<td>$P^#_U (init(v, x))$: $\text{len}(x^0) = \text{len}(x) \land \text{hd}(x) = v \land \forall y \in t_1(x). x[y] = v$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>initSeq</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =, P_1 &lt; 1$</td>
<td>$P^#_U (initSeq(v, x))$: $\text{len}(x^0) = \text{len}(x) \land \text{hd}(x) = v \land \forall y \in t_1(x). x[y] = v$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>add(v)</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P = &lt; 1$</td>
<td>$P^#_U (add(v, x))$: $\text{len}(x^0) = \text{len}(x) \land \text{hd}(x) = v \land \forall y \in t_1(x). x[y] = v$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>copy</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =$</td>
<td>$P^#_U (copy(x))$: $\text{len}(x^0) = \text{len}(x) \land \text{hd}(x) = v \land \forall y \in t_1(x). x[y] = v$</td>
<td></td>
</tr>
<tr>
<td>fold</td>
<td>delPred</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =, P_1 &lt; 1$</td>
<td>$P^#_U (delPred(v, x, l, u))$: $\text{ms}(x) = \text{ms}(x^0) \land \text{ms}(l) \lor \text{ms}(u)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =, P_1 &lt; 1$</td>
<td>$P^#_U (max(v, x, l, u))$: $\text{equal}(x, x^0) \land \text{len}(x) = \text{len}(l) + \text{len}(u) \land \forall y \in t_1(l). l[y] = v \land \forall y \in t_1(u). u[y] = v$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>clone</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =$</td>
<td>$P^#_U (clone(x, y))$: $\text{equal}(x, x^0) \land \text{len}(x) = \text{len}(l) + \text{len}(u) \land \forall y \in t_1(l). l[y] = v \land \forall y \in t_1(u). u[y] = v$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>split</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =$</td>
<td>$P^#_U (split(x, y))$: $\text{equal}(x, x^0) \land \text{len}(x) = \text{len}(l) + \text{len}(u) \land \forall y \in t_1(l). l[y] = v \land \forall y \in t_1(u). u[y] = v$</td>
<td></td>
</tr>
<tr>
<td>fold2</td>
<td>equal</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =$</td>
<td>$P^#_U (merge(x, z))$: $\text{ms}(x) \lor \text{ms}(z) = \text{ms}(r) \land \text{has}(x^0) = \text{ms}(x) \lor \ldots$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>concat</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =, P_1 &lt; 3$</td>
<td>$P^#_U (concat(x, z))$: $\text{equal}(x, x^0) \land \text{equal}(z, z^0) \land \text{sorted}(x^0) \land \text{sorted}(z^0) \land \forall y \in t_1(r). r[y] = v$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>merge</td>
<td>(0, 1)</td>
<td>&lt; 1</td>
<td>$P =$</td>
<td>$P^#_U (merge(x, z))$: $\text{equal}(x, x^0) \land \text{equal}(z, z^0) \land \text{sorted}(x^0) \land \text{sorted}(z^0) \land \forall y \in t_1(r). r[y] = v$</td>
<td></td>
</tr>
<tr>
<td>sort</td>
<td>bubble</td>
<td>(1, -)</td>
<td>&lt; 1</td>
<td>$P =, P_1 &lt; 3$</td>
<td>$P^#_U (bubble(x))$: $\text{ms}(x) = \text{ms}(x^0) \land \text{ms}(r)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>insert</td>
<td>(1, -)</td>
<td>&lt; 1</td>
<td>$P =, P_1 &lt; 3$</td>
<td>$P^#_U (insert(x, y))$: $\text{equal}(x, x^0) \land \text{sorted}(x^0) \land \text{sorted}(r^0) \land \forall y \in t_1(r). r[y] = v$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>quick</td>
<td>(-, 2)</td>
<td>&lt; 2</td>
<td>$P =, P_1 &lt; 4$</td>
<td>$P^#_U (quick(x))$: $\text{ms}(x) = \text{ms}(x^0) = \text{ms}(res)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>merge</td>
<td>(-, 2)</td>
<td>&lt; 2</td>
<td>$P =, P_1 &lt; 4$</td>
<td>$P^#_U (quick(x))$: $\text{equal}(x, x^0) \land \text{sorted}(res)$</td>
<td></td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Formal Models and Semantics for IMPR
3. Foundations of Static Analysis by Abstract Interpretation
4. Application: Programs with Lists and Data
5. Application: Decision Procedures by Static Analysis
6. Elements of Inter-procedural Analysis
7. Application: Programs with Lists, Data, and Procedures
8. Extension: Programs with Complex Data Structures
9. Extension: Programs with Inductive Data Structures
Reasoning about Composite Data Structures
— using a FO Logic Framework —

joint work with A. Bouajjani, C. Drăgoi, C. Enea

CONCUR’09
Properties of Complex Data Structures

```c
struct a_ty {
    int id;
    dll_ty* head;
}
struct dll_ty {
    bool flag;
    a_ty* root;
    dll_ty* next, *prev;
}
a_ty arr[N];
```

**Structure:** “the array contains in each cell a reference to an acyclic doubly linked list”

**Sizes:** “the array is sorted in decreasing order w.r.t. the lengths of lists stored”

**Data:** “the array is sorted w.r.t. the values of the field id”
Recall: Heap Graph Model

Heaps are represented as labeled directed graphs called heap graphs

```c
struct a_ty {
    int id;
    dll_ty* head;
}
struct dll_ty {
    bool flag;
    a_ty* root;
    dll_ty* next, *prev;
}
a_ty arr[N];
```

The graph is deterministic
The array fields create acyclic distinct paths
A Very Expressive Logic

- assume $\mathcal{D}$ the data domain where data fields take values
- assume $\text{FO}(\mathcal{D}, \mathcal{O}, \mathcal{P})$ a first order logic on $\mathcal{D}$, with operations in $\mathcal{O}$ and predicates in $\mathcal{P}$

$\text{gCSL}$ is a multi-sorted first order logic on graphs parametrized by $\text{FO}(\mathcal{D}, \mathcal{O}, \mathcal{P})$

$$\text{gCSL} = \text{FO} + \text{reachability} + \text{arithmetical constraints} + \text{FO}(\mathcal{D}, \mathcal{O}, \mathcal{P})$$
Reachability Predicates

$x \overset{\{f,g,h\}}{\longrightarrow} y$

Diagram:

- $x \rightarrow f \rightarrow g \rightarrow h \rightarrow g \rightarrow f \rightarrow y$

- The sequence $x \rightarrow f \rightarrow g \rightarrow h \rightarrow g \rightarrow f \rightarrow y$ illustrates the reachability from $x$ to $y$ through the predicates $f$, $g$, and $h$.
Reachability Predicates

\[ x \xrightarrow{\{f, g, h\}} y \quad \text{link}(z) = x \]
Reachability Predicates

\[ x \xrightarrow{\{f, g, h\}, l} y \]

\[ x \xrightarrow{\{f, g, h\}} y \quad \text{link}(z) = x \]
Data Constraints

\[ \text{id}(x) = 3 \]
\[ \exists c. \text{id}(x) + \text{id}(y) + c \geq 9 \]
\[ x \xrightarrow{\{f,g,h\},l} y \land l = 3 \land v \xrightarrow{\{g\},l'} w \]
\[ l' < l \land l + l' \geq 4 \]
\[ x \xrightarrow{\{f,g,h\}} y \quad \text{link}(z) = x \]
**Properties of Complex Data Structures in gCSL**

**Structure:** “the array contains in each cell a reference to an acyclic doubly linked list”

\[
\forall i \: \exists x, y \cdot x = \text{head}(a[i]) \land x \xrightarrow{\{\text{next,} \text{prev}\}} y
\]

**Sizes:** “the array is sorted w.r.t. the lengths of lists stored”

\[
\forall j, j'. \: j < j' \implies \exists x, x', l, l'. \: (x = \text{head}(a[j]) \land x' = \text{head}(a[j']) \land \\
\quad x \xrightarrow{\{\text{next}, l\}} \text{null} \land x' \xrightarrow{\{\text{next}, l'\}} \text{null} \land l' \leq l)
\]

**Data:** “the array is sorted w.r.t. the values of the field id”

\[
\forall i, j. \: i < j \implies \text{id}(a[i]) < \text{id}(a[j])
\]
The satisfiability problem of $g$CSL is undecidable

- when data are restricted to finite domains (such as booleans), the logic subsumes the first-order logic on graphs with reachability
- when the models are restricted to simple structures, like sequences or arrays, for very simple data logics such as $(\mathbb{N}, =)$, the fragment $\forall^* \exists^*$ is undecidable
An ordered partition over $\mathcal{RT}$ is a mapping $\sigma : \mathcal{RT} \rightarrow \{1, \ldots, N\}$

- a type $R \in \mathcal{RT}$ is of level $k$ iff $\sigma(R) = k$
An ordered partition over $\mathcal{R}_T$ is a mapping $\sigma : \mathcal{R}_T \to \{1, \ldots, N\}$ such that:

- A type $R \in \mathcal{R}_T$ is of level $k$ iff $\sigma(R) = k$

For $1 \leq k \leq |\sigma|$, CSL is the smallest set of formulas closed under disjunction and conjunction, which contains all the closed formulas of the form:

$$
\exists_{\leq k} \ \forall_k \ \exists_{\leq k-1} \ \forall_{k-1} \ \cdots \ \exists_1 \ \forall_1 \ \{\exists_d, \forall_d\}^* \ \phi
$$

$\phi$ is a quantifier-free formula in gCSL.
For $1 \leq k \leq |\sigma|$, CSL is the smallest set of formulas closed under disjunction and conjunction, which contains all the closed formulas of the form:

$$\exists^{*}_{k} \forall^{*}_{k} \exists^{*}_{k-1} \forall^{*}_{k-1} \ldots \exists^{*}_{1} \forall^{*}_{1} \{\exists_{d}, \forall_{d}\}^{*} \cdot \phi$$

$\phi$ is a quantifier-free formula in gCSL such that:

- **Reach**: for any $x \xrightarrow{A, ind} x'$, $x$ and $x'$ are free or existential variables
- **UnivIdx**: two universal index variables can be used only in $j < j'$ or $j = j'$

**YES** $\forall j, j'. j < j' \Rightarrow data(a[j]) < data(a[j'])$

**NO** $\forall j, j'. j + 1 = j' \Rightarrow data(a[j]) < data(a[j'])$

- **LEV**: atomic constraints on lengths of lists and array indexes involve only one level

**YES** $\exists x, x', l_1 \exists z, z', l_2. x \xrightarrow{\{f\}, h_1} x' \land z \xrightarrow{\{f\}, h_2} z' \land l_1 \geq 4 \land l_2 \geq 0$

**NO** $\exists x, x', l_1 \exists z, z', l_2. x \xrightarrow{\{f\}, h_1} x' \land z \xrightarrow{\{f\}, h_2} z' \land l_1 + l_2 \geq 0$
**Structure:** “the array contains in each cell a reference to an acyclic doubly linked list”

\[
\forall i \exists x, y. \ x = \text{head}(a[i]) \land x \xrightarrow{\{\text{next, prev}\}} y
\]

**Sizes:** “the array is sorted w.r.t. the lengths of lists stored”

\[
\forall j, j'. j < j' \implies \exists x, x', l, l'. (x = \text{head}(a[j]) \land x' = \text{head}(a[j'])) \land
\]

\[
x \xrightarrow{\{\text{next}, l\}} \text{null} \land x' \xrightarrow{\{\text{next}, l'\}} \text{null} \land l' \leq l
\]

**Data:** “the array is sorted w.r.t. the values of the field id”

\[
\forall i, j. i < j \implies \text{id}(a[i]) < \text{id}(a[j])
\]
The satisfiability of CSL is decidable if the satisfiability of the underlying first order logic FO(\(\mathcal{D}, \mathcal{O}, \mathcal{P}\)) is decidable.

Let

\[ \varphi_k = \exists^*_\leq k \mathbf{r} \ \forall^*_k \mathbf{p} \ \exists^*_{k-1} \mathbf{r'} \ \forall^*_k \mathbf{p'} \ \ldots \ \exists^*_1 \mathbf{r''} \ \forall^*_1 \mathbf{p''} \ \{\exists_d, \forall_d\}^*. \phi \]

1. compute \(\varphi_{k-1}\) equi-satisfiable to \(\varphi_k\) such that

\[ \varphi_{k-1} = \exists^*_{\leq k-1} \mathbf{z} \ \forall^*_{k-1} \mathbf{w} \ \ldots \ \exists^*_1 \mathbf{z'} \ \forall^*_1 \mathbf{w'} \ \{\exists_d, \forall_d\}^*. \phi' \]

until it ends up with a formula over variables of level 1

\[ \varphi = \exists^*_1 \mathbf{x} \ \forall^*_1 \mathbf{y} \ \{\exists_d, \forall_d\}^*. \phi'' \]

2. reduce the satisfiability of \(\varphi\) to the satisfiability of a formula in FO(\(\mathcal{D}, \mathcal{O}, \mathcal{P}\))
Satisfiability of CSL Formulas

Theorem

The satisfiability of CSL is decidable if the satisfiability of the underlying first order logic \( FO(\mathcal{D}, \mathcal{O}, \mathcal{P}) \) is decidable.

Let

\[ \varphi = \exists_1^* x \ \forall_1^* y \ \{\exists_d, \forall_d\}^* \phi'' \]

1. compute the set of small models for the reachability and size constraints
2. for each small model, build a \( FO(\mathcal{D}, \mathcal{O}, \mathcal{P}) \) formula \( \psi \)

If one of \( \psi \) is satisfiable then \( \varphi \) is satisfiable.
\( \varphi = \exists x, q, z . \ x \xrightarrow{\{f\}} q \land x \xrightarrow{\{f\}} z \land q \neq z \)

- \( \varphi \) has two small models of size three
Computing Small Models

\[ \varphi = \exists x, q, z \exists l_1, l_2. \ x \xrightarrow{\{f\}, h} q \land x \xrightarrow{\{f\}, l_2} z \land q \neq z \land l_1 + l_2 \geq 8 \]

\[ l_1 + l = l_2 \land l_1 + l_2 \geq 8 \]
Computing Small Models

\[ \varphi = \exists x, q, z \exists l_1, l_2. x \xrightarrow{\{f\}, h} q \land x \xrightarrow{\{f\}, l_2} z \land q \neq z \land l_1 + l_2 \geq 8 \]

- minimal solutions \((l_1, l_2, l)\) for \(l_1 + l = l_2 \land l_1 + l_2 \geq 8\)

\[ \mathcal{M} = \{(1, 7, 6), (2, 6, 4), (3, 5, 2)\} \]

- small models for \(\varphi\)
$\varphi = \exists x, q, z \exists l_1, l_2. x \xrightarrow{\{f\}, l_1} q \land x \xrightarrow{\{f\}, l_2} z \land q \neq z$

$\land l_1 + l_2 \geq 8$

$\land g(x) = 0 \land g(q) = 2$

$\land \forall y, y'. (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$
\[ \varphi = \exists x, q, z \, \exists l_1, l_2. \, x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z \wedge q \neq z \wedge l_1 + l_2 \geq 8 \wedge g(x) = 0 \wedge g(q) = 2 \wedge \forall y, y'. (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y')) \]
\[ \varphi = \exists x, q, z \exists l_1, l_2. x \xrightarrow{\{f\}, l_1} q \land x \xrightarrow{\{f\}, l_2} z \land q \neq z \land l_1 + l_2 \geq 8 \land g(x) = 0 \land g(q) = 2 \land \forall y, y'. (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y')) \]
\[ \varphi = \exists x, q, z \exists l_1, l_2. x \xrightarrow{\{f\}, l_1} q \land x \xrightarrow{\{f\}, l_2} z \land q \neq z \]
\[ \land l_1 + l_2 \geq 8 \]
\[ \land g(x) = 0 \land g(q) = 2 \]
\[ \land \forall y, y'. (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y')) \]

Diagram:
- **g:0**
- **f**
- **x**
- **f**
- **q**
- **g:2**
- **f**
- **f**
- **z**
\[ \varphi = \exists x, q, z \exists l_1, l_2. x \xrightarrow{\{f\}, l_1} q \land x \xrightarrow{\{f\}, l_2} z \land q \neq z \land l_1 + l_2 \geq 8 \land g(x) = 0 \land g(q) = 2 \land \forall y, y'. (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y')) \]
$\varphi = \exists x, q, z \exists l_1, l_2. x \xrightarrow{\{f\}, l_1} q \land x \xrightarrow{\{f\}, l_2} z \land q \neq z$

$\land l_1 + l_2 \geq 8$

$\land g(x) = 0 \land g(q) = 2$

$\land \forall y, y'. (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$

$\psi_1 = \exists c_1, c_2, c_3, c_4, c_5, c_6, c_7. \land_{i \neq j} c_i \neq c_j$

$true \land true \land true \land true$

$\land c_1 = 0 \land c_3 = 2$

$\land \land_{1 \leq i < j \leq 7} c_i < c_j$
Checking Data Constraints (4/4)

\[ \varphi = \exists x, q, z \ \exists l_1, l_2. \ x \xrightarrow{\{f\}, l_1} q \land x \xrightarrow{\{f\}, l_2} z \land q \neq z \]

\[ \land l_1 + l_2 \geq 8 \]

\[ \land g(x) = 0 \land g(q) = 2 \]

\[ \land \forall y, y'. (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y')) \]

\[ \psi_1 = \exists c_1, c_2, c_3, c_4, c_5, c_6, c_7. \land_{i \neq j} c_i \neq c_j \]

\[ \land true \land true \land true \land true \]

\[ \land c_1 = 0 \land c_3 = 2 \]

\[ \land \land_{1 \leq i < j \leq 7} c_i < c_j \]
**Decision Procedure: Summary**

1. Choose a small model for the reachability and size constraints; if there are no models then $\varphi$ is unsatisfiable.
2. Build a $\text{FO}(\mathcal{D}, \mathcal{O}, \mathcal{P})$ formula $\psi$ for the selected small model.
3. Check the satisfiability of $\psi$.

**Remark**

The complexity of the reduction procedure is $\text{NP}^{\text{MOILP}}$ when the number of universally quantified variables is fixed.
Theorem

If the satisfiability of the underlying first order logic $FO(D, O, P)$ is decidable, then the satisfiability of CSL is decidable.

Theorem

For any basic statement $S$ and any CSL formula $\varphi$, we can compute in polynomial time a formula $\text{post}(S, \varphi)$ describing the strongest post-condition of $\varphi$ by $S$. 
1. Introduction
2. Formal Models and Semantics for IMPR
3. Foundations of Static Analysis by Abstract Interpretation
4. Application: Programs with Lists and Data
5. Application: Decision Procedures by Static Analysis
6. Elements of Inter-procedural Analysis
7. Application: Programs with Lists, Data, and Procedures
8. Extension: Programs with Complex Data Structures
9. Extension: Programs with Inductive Data Structures
The limits of specifying complex heap shapes in SL are given by the class of inductive predicates allowed. However, the classical data structures may be specified.

**Exercise:** Specify the shape of the following data structures:

- Binary trees
- Doubly linked lists segments
- Tree with linked leaves
Observation

The limits of specifying complex heap shapes in SL are given by the class of inductive predicates allowed.

However, the classical data structures may be specified.

**Exercise:** Specify the shape of the following data structures:

\[
\begin{align*}
\text{dll}(E, L, P, F) & \triangleq (E = F \land L = P \land \text{emp}) \lor (E \neq F \land L \neq P \land \exists X. E \mapsto \{ (\text{nxt}, X), (\text{prv}, P) \} * \text{dll}(X, L, E, F) ) \\
\text{btree}(E) & \triangleq (E = \emptyset \land \text{emp}) \lor (E \neq \emptyset \land \exists X, Y. E \mapsto \{ (\text{lson}, X), (\text{rson}, Y) \} * \text{btree}(X) * \text{btree}(Y) ) \\
\text{tll}(R, P, E, F) & \triangleq (R = E \land R \mapsto \{ (\text{lson}, \emptyset), (\text{rson}, \emptyset), (\text{parent}, P), (\text{nxt}, F) \} ) \lor (R \neq E \land \exists X, Y, Z. R \mapsto \{ (\text{lson}, X), (\text{rson}, Y), (\text{parent}, P), (\text{nxt}, Z) \} * \text{tll}(X, R, E, Z) * \text{tll}(Y, R, Z, F) )
\end{align*}
\]
The fragment allowing these specifications has good theoretical properties:

- decidability of satisfiability \[ \text{[Brotherston et al, 14]} \] → by reduction boolean equations
- decidability of the entailment \[ \text{[Iosif et al, 13]} \] → by reduction to MSO on graphs with bounded width
Recently, efficient dedicated solvers have been released, *e.g.*:

- Asterix [Perez & Rybalchenko, 11]
- Cyclist-SL and SAT-SL [Gorogiannis et al, 12]
- SLEEK [Chin et al, 10]
- SLIDE [Iosif et al, 14]
- SPEN [Enea, Lengal, S., Vojnar, 14]

Follow them on SL-COMP competition:

- 6 solvers involved (freely available on StarExec)
- more than 600 benchmarks

www.liafa.univ-paris-diderot.fr/slcomp
Extensions of Separation Logic

- Introducing content and size constraints  [Chin et al, 10],[S. et al, 15]

- Adding pre-field separation to express overlaid data structures  [Yang et al,11],[Enea et al, 13]

\[ \text{nll}_\beta(h, \Box, \Box) \otimes \text{ls}_\delta(dl, \Box) \wedge \beta(\Diamond) = \delta(\Diamond) \]
Conclusion of the Part

- Shape analysis benefits from Separation Logic compositional reasoning.

- Shape analysis may be extended to content and size analysis.

- Efficiency is obtained using sound syntax-oriented procedures.

- Sound procedures for undecidable logic fragments may be obtained by applying static analysis.