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

Establish automatically that a program meets a specification.

Specified Properties

- Explicit: "the program sorts the input list"
  $\longrightarrow$ specified using some formalism in assertions
- Implicit: "the program never dereferences a null pointer"
  $\longrightarrow$ specified as bad behaviour in semantics

Automatic Technique

The user could not interact with the analyser during its running.
$\longrightarrow$ but may write program assertions, choose analysis parameters, ...

Consequence of Rice's Theorem

It is impossible to build sound, complete, and automatic analysers of non trivial semantic properties for programs written in a Turing-complete programming language.

What can be done?

- Confine to "trivial" classes of programming languages
  $\longrightarrow$ model-checking finite automata, but manage state explosion ☹
- Give up "automation"
  $\longrightarrow$ interactive theorem provers 😑
- Give up "soundness" by looking at bounded executions
  $\longrightarrow$ testing, bounded model-checking, but manage false negatives 😑
- Give up "completeness" by using property preserving abstractions
  $\longrightarrow$ type-checking, data flow analysis, abstract interpretation ☺

- Numerical programs: PolySpace (1996-), Astrée (2002-)
- Device drivers: SLAM (2000-)
- Programs with dynamic memory: Infer (2015-)

Inputs:

- Program and its formal concrete semantics
- User's assertions
- Targeted class of properties

Outputs:

- Program properties valid for all concrete executions
- Alarms about unsatisfied specifications, some may be false alarms

Abstraction process

Interpret the program according to a simplified, “abstract” semantics.

Property-preserving abstraction

Formally show that the “abstract” semantics preserves the relevant properties of the “concrete” (program) semantics.

Preservation of properties

Interpretation with the abstract semantics therefore gives sound information about the properties of concrete executions.

Find abstractions with

1. High precision ⟶ fewer false alarms
2. Low complexity ⟶ scale-up to bigger programs

Objective of analysis

Discover for each program point if a pointer variable may have the null value at the run time.

The abstract semantics tracks the following properties for each pointer variable x:

$$x = \mathrm{null}$$

$$x \neq \mathrm{null}$$

Static Analysis Example: Null Pointer Aliasing

```
list* search(list* h, int key) {

  list* it = h;

  bool b = false;


  while (it != NULL && !b) {

    if (it->data == key)
       b = true;
    else

       it = it->next;

  }


  return it;
}
```

```
list* search(list* h, int key) {
```

$h = \text{null} \quad \bigvee \quad h \neq \text{null}$

```
  list* it = h;

  bool b = false;


  while (it != NULL && !b) {

    if (it->data == key)
       b = true;
    else

       it = it->next;

  }


  return it;
}
```

```
list* search(list* h, int key) {
```
$h = null \quad \bigvee \quad h \neq null$
```
  list* it = h;
```
$h = null \ \wedge \ it = null \quad \bigvee \quad h \neq null \ \wedge \ it \neq null$
```
  bool b = false;
```
$h = null \ \wedge \ it = null \quad \bigvee \quad h \neq null \ \wedge \ it \neq null$
```
  while (it != NULL && !b) {

    if (it->data == key)
      b = true;
    else

      it = it->next;

  }


  return it;
}
```

Static Analysis Example: Null Pointer Aliasing

```
list* search(list* h, int key) {
```
$h = null \quad \bigvee \quad h \neq null$
```
  list* it = h;
```
$h = null \wedge it = null \quad \bigvee \quad h \neq null \wedge it \neq null$
```
  bool b = false;
```
$h = null \wedge it = null \quad \bigvee \quad h \neq null \wedge it \neq null$
```
  while (it != NULL && !b) {
```
$h \neq null \wedge it \neq null$
```
    if (it->data == key)
       b = true;
    else
```
$h \neq null \wedge it \neq null$
```
       it = it->next;

  }


  return it;
}
```

Static Analysis Example: Null Pointer Aliasing

```
list* search(list* h, int key) {
```
$h = null \quad \bigvee \quad h \neq null$
```
  list* it = h;
```
$h = null \;\wedge\; it = null \quad \bigvee \quad h \neq null \;\wedge\; it \neq null$
```
  bool b = false;
```
$h = null \;\wedge\; it = null \quad \bigvee \quad h \neq null \;\wedge\; it \neq null$
```
  while (it != NULL && !b) {
```
$h \neq null \;\wedge\; it \neq null$
```
    if (it->data == key)
       b = true;
    else
```
$h \neq null \;\wedge\; it \neq null$
```
       it = it->next;
```
$h \neq null \;\wedge\; it \neq null \quad \bigvee \quad h \neq null \;\wedge\; it = null$
```
  }


  return it;
}
```

Static Analysis Example: Null Pointer Aliasing

```
list* search(list* h, int key) {
                h = null   ⋁   h ≠ null
  list* it = h;
                h = null ∧ it = null   ⋁   h ≠ null ∧ it ≠ null
  bool b = false;
                h = null ∧ it = null   ⋁   h ≠ null ∧ it ≠ null
                                       ⋁   h ≠ null ∧ it = null
  while (it != NULL && !b) {
                h ≠ null ∧ it ≠ null
    if (it->data == key)
      b = true;
    else
                h ≠ null ∧ it ≠ null
      it = it->next;
                h ≠ null ∧ it ≠ null   ⋁   h ≠ null ∧ it = null
  }


  return it;
}
```

```
list* search(list* h, int key) {
```
$$h = null \quad \bigvee \quad h \neq null$$
```
  list* it = h;
```
$$h = null \ \wedge \ it = null \quad \bigvee \quad h \neq null \ \wedge \ it \neq null$$
```
  bool b = false;
```
$$h = null \ \wedge \ it = null \quad \bigvee \quad h \neq null \ \wedge \ it \neq null \quad \bigvee \quad h \neq null \ \wedge \ it = null$$
```
  while (it != NULL && !b) {
```
$$h \neq null \ \wedge \ it \neq null$$
```
    if (it->data == key)
       b = true;
    else
```
$$h \neq null \ \wedge \ it \neq null$$
```
       it = it->next;
```
$$h \neq null \ \wedge \ it \neq null \quad \bigvee \quad h \neq null \ \wedge \ it = null$$
```
  }
```
$$h = null \ \wedge \ it = null \quad \bigvee \quad h \neq null \ \wedge \ it \neq null \quad \bigvee \quad h \neq null \ \wedge \ it = null$$
```
  return it;
}
```

Pioneering works in ’70, applied to ALGOL68 and Pascal programs.

Main properties

Variables aliasing, data structures separation, shape of the heap, size of the heap, . . .

Applications

- Understand design choices of new programming languages
- Program optimisation
- Verification of implicit and explicit specifications
- Optimise compilers for imperative and functional languages

Pioneering works in '70, applied to ALGOL68 and Pascal programs.

PASTE 2001:

Pointer Analysis: Haven't We Solved This Problem Yet?

Michael Hind
IBM Watson Research Center
30 Saw Mill River Road
Hawthorne, New York 10532
hind@watson.ibm.com

ABSTRACT

During the past twenty-one years, over seventy-five papers and nine Ph.D. theses have been published on pointer analysis. Given the tomes of work on this topic one may wonder, "Haven't we solved this problem yet?" With input from many researchers in the field, this paper describes issues related to pointer analysis and remaining open problems.

Pioneering works in '70, applied to ALGOL68 and Pascal programs.

PASTE 2001:

Pointer Analysis: Haven't We Solved This Problem Yet?

Michael Hind
IBM Watson Research Center
30 Saw Mill River Road
Hawthorne, New York 10532
hind@watson.ibm.com

ABSTRACT

During the past twenty-one years, over seventy-five papers and nine Ph.D. theses have been published on pointer analysis. Given the tomes of work on this topic one may wonder, "Haven't we solved this problem yet?" With input from many researchers in the field, this paper describes issues related to pointer analysis and remaining open problems.

Open problem: increase precision while preserving scalability

⟶ shape analysis [Larus&Hilfinger,88, Horwitz&al,89,...]

```
/* @brief Reverse list @p l in place and return the new head */
list* reverse(list* l) {
  list* f = l;
  list* r = NULL;

  while (f != NULL) { // !!!
    list *t = f->next;

    f->next = r;

    r = f; f = t;
  }

  return r;
}
```

Targeted properties: $\mathrm{list}(l, \mathrm{null})$, $\mathrm{list}(r, \mathrm{null})$, $r \xrightarrow{\mathrm{next}} l$, ...

```
/* @brief Search @p key in the sorted list @p l*/
list* search(list* l, int key);

int main(void) {
  ...
  list* h = list_init(d); // initialises with 0..d
  ...
  x = search(h, d-1);

  y = x->next; // !!!
  ...
}
```

Targeted properties: $\mathrm{list}(h, null)$ of data 0..d

Automata-based:

- finite automata in PALE [Møller,01]
- tree automata in Forester & Predator [Vojnar *et al*,11]
- counter automata [Bouajjani *et al*,06]

Logic-based:

- Boolean abstraction [Wies *et al*,09]
- 3-valued logic [TVLA – Sagiv *et al*]
- Separation Logic [Smallfoot, Infer – O'Hearn *et al*,01], [MemCAD – Rival *et al*,07], [Celia – S. *et al*,10]
- FO with reachability [Yorsh *et al*,06], [Bouajjani *et al*,09], [Madhusudan *et al*,11]

... and many others! (see [Hind,01]) Sorry for no credits in this talk... ☹

Logic-based abstractions for static analysis of shape and content properties using abstract interpretation.

Desired properties for the logic

- High expressivity $\longrightarrow$ precision of analysis
- High efficiency for $\longrightarrow$ scaling-up to large programs
  - computing interpretation of program statements
  - soundly testing satisfiability and entailment of assertions
- Encoding in a complete lattice

  $\longrightarrow$ abstract interpretation

Outline

Outline

Class of Programs

We consider the IMPR toy language to focus on the properties targeted by our analyses.

Included:

- numeric types
- pointer to record types
- strong typing
- explicit heap (de-)allocation
- recursive functions

Excluded:

- expressions with side effect 😑
- uninitialised allocation of memory (stack or heap) 😑
- union and array types 😑
- pointer arithmetics and casting ☹
- pointer to functions ☹
- pointers inside the stack ☹
- ...

Some excluded features are easy (😑) or elaborate (😑) for our analyses.

Class of Programs: Types

Basic Types

Numeric types DT $\in \mathbf{DT}$ on which are defined operations o $\in \mathbf{O}$ and boolean relations r $\in \mathbf{R}$; $\mathbb{D}$ is the union of numerical domains.

Record types

User defined types are record types RT $\in \mathbf{RT}$, defined by a set of numeric df $\in \mathbf{DF}$ or reference rf $\in \mathbf{RF}$ fields as follows:

```
struct RT { ty1 f1; ... tyn fn; };
```

with tyi ::= DT | RT$*$ and `fi` $\in \mathbf{FS} = \mathbf{DF} \cup \mathbf{RF}$.

Language strongly typed except the `null` constant!

Variable declaration

Numeric variables $dv \in \mathbf{DV}$ and reference variables $rv \in \mathbf{RV}$ are either declared global or local to some procedure.

Procedure declaration

Explicit syntax for output parameters; all procedures return a result.

```
          ty  P(ty1 v1, ..., tyn out vn)
             { // declarations for local variables
               ty v;
startP:       // sequence of statements
               ...; v = ...; ...
               return v;
endP:        }
```

Boolean and numeric expressions

Fixed evaluation order of arguments; no side effects.

$$\mathrm{be} ::= \mathrm{bcst} \mid \mathrm{bv} \mid \mathrm{r}(\overrightarrow{\mathrm{de}}) \mid \mathrm{rv}_1\texttt{==}\mathrm{rv}_2 \mid \texttt{!}\mathrm{be} \mid \mathrm{be} \wedge \mathrm{be} \mid \mathrm{be} \vee \mathrm{be}$$

$$\mathrm{de} ::= \mathrm{dcst} \mid \mathrm{dv} \mid \mathrm{o}(\overrightarrow{\mathrm{de}}) \mid \mathrm{rv}{\rightarrow}\mathrm{df}$$

Reference expressions

No arithmetics on references!

$$\mathrm{re} ::= \texttt{null} \mid \mathrm{rv} \mid \mathrm{rv}{\rightarrow}\mathrm{rf}$$

Statements

Restricted procedure call; explicit dynamic memory (de)allocation.

$$\begin{aligned}\mathrm{astmt} &::= \mathrm{dv}\texttt{=}\mathrm{de} \mid \mathrm{rv}{\rightarrow}\mathrm{df}\texttt{=}\mathrm{de} \mid \mathrm{rv}\texttt{=}\mathrm{re} \mid \mathrm{rv}{\rightarrow}\mathrm{rf}\texttt{=}\mathrm{re} \mid \mathrm{bv}\texttt{=}\mathrm{be} \\ &\mid \mathrm{rv}\texttt{=new}\ \mathrm{RT} \mid \texttt{free}(\mathrm{rv}) \mid \texttt{nop} \\ \mathrm{stmt} &::= \mathrm{astmt} \mid \mathrm{v}\texttt{=}\mathrm{P}(\overrightarrow{\mathrm{v}}) \mid \mathrm{stmt};\mathrm{stmt} \mid \texttt{if}\ldots \mid \texttt{while}\ldots\end{aligned}$$

```
struct list { int data; list* next};

list* search(list* h, int key) {
  list* it; bool b;
  it = h;
  b = false;
  while (!(it == NULL ∨ b)) {
    if (it->data == key)
       b = true;
    else
       it = it->next;
  }
  return it;
}
```

Memory configs $\mathtt{Mem} \triangleq \mathtt{Stacks} \times \mathtt{Heaps} \ni \mathtt{m}$

Store-less Heap

Absence of arithmetics over addresses permits the store-less semantics, *i.e.* the heap locations are represented by a domain $(\mathbb{L}, =)$.

Strongly-typed Heap

Strong typing permits indexing of heap locations by program types, *i.e.*

$$\mathbb{L} = \{\boxtimes\} \cup \bigcup_{\mathtt{ty} \in \mathbf{RT}*} \mathbb{L}_{\mathtt{ty}}$$

with $\boxtimes$ (for null) the only untyped value.

Then, the heap formal model is:

$$\mathtt{H} \in \mathsf{Heaps} \triangleq [(\mathbb{L} \times \mathbf{FS}) \nrightarrow (\mathbb{D} \cup \mathbb{L})]$$

with $\mathtt{H}(\boxtimes, \mathtt{f})$ undefined for any $\mathtt{H}$ and $\mathtt{f}$.

Formal Semantics: Executions

Control points	$\mathbf{CP} \ni \ell, \ell'$
	$\ni \mathit{start}_{\mathrm{P}}, \mathit{end}_{\mathrm{P}}$ for each procedure $\mathrm{P} \in \mathbf{P}$
Stack	$\mathtt{Stacks} \triangleq [(\mathbf{CP} \times \mathbf{P} \times (\mathbf{DV} \not\to \mathbb{D} \ \cup \ \mathbf{RV} \not\to \mathbb{L}))^*] \ni \mathtt{S}$
Memory	$\mathtt{Mem} \triangleq \mathtt{Stacks} \times \mathtt{Heaps} \ni \mathtt{m}$
Configurations	$\mathtt{Config} \triangleq \mathbf{CP} \times (\mathtt{Mem} \cup \{\mathtt{merr}\}) \ni \mathtt{C}$

Natural Semantics Predicates

$$\mathtt{C} \vdash \mathit{stmt} \rightsquigarrow \mathtt{C}'$$

$$\mathtt{m} \vdash \mathit{astmt} \rightsquigarrow \mathtt{m}' \mid \mathtt{merr}$$

$$\mathtt{m} \vdash \mathit{be} \rightsquigarrow \mathrm{b} \mid \mathtt{merr}$$

$$\mathtt{m} \vdash \mathit{de} \rightsquigarrow \mathrm{c} \mid \mathtt{merr}$$

$$\mathtt{m} \vdash \mathit{re} \rightsquigarrow \mathrm{a} \mid \mathtt{merr}$$

with $\mathrm{b} \in \{\mathtt{true}, \mathtt{false}\}$, $\mathrm{c} \in \mathbb{D}$, $\mathrm{a} \in \mathbb{L}$.

Natural Semantics: Rules (Some)

$$\frac{\forall i.\ \mathtt{m} \vdash \mathrm{de}_i \rightsquigarrow c_i \quad r(c_1, \ldots, c_n) = \mathtt{true}}{\mathtt{m} \vdash r(\mathrm{de}_1, \ldots, \mathrm{de}_n) \rightsquigarrow \mathtt{true}} \qquad \frac{\exists i.\ \mathtt{m} \vdash \mathrm{de}_i \rightsquigarrow \mathtt{merr}}{\mathtt{m} \vdash r(\mathrm{de}_1, \ldots, \mathrm{de}_n) \rightsquigarrow \mathtt{merr}}$$

Natural Semantics: Rules (Some)

$$\frac{\forall i.\ \mathtt{m} \vdash \mathrm{de}_i \rightsquigarrow \mathrm{c}_i \quad \mathrm{r}(\mathrm{c}_1, \ldots, \mathrm{c}_n) = \mathtt{true}}{\mathtt{m} \vdash \mathrm{r}(\mathrm{de}_1, \ldots, \mathrm{de}_n) \rightsquigarrow \mathtt{true}} \qquad \frac{\exists i.\ \mathtt{m} \vdash \mathrm{de}_i \rightsquigarrow \mathtt{merr}}{\mathtt{m} \vdash \mathrm{r}(\mathrm{de}_1, \ldots, \mathrm{de}_n) \rightsquigarrow \mathtt{merr}}$$

$$\frac{\mathtt{m}(rv) = \mathrm{a} \neq \boxtimes \quad \mathtt{m}(\mathrm{a}, \mathit{df}) = \mathrm{c}}{\mathtt{m} \vdash rv{\longrightarrow}\mathit{df} \rightsquigarrow \mathrm{c}} \qquad \frac{\mathtt{m}(rv) = \boxtimes}{\mathtt{m} \vdash rv{\longrightarrow}\mathit{df} \rightsquigarrow \mathtt{merr}}$$

$$\frac{\forall i.\ \mathtt{m} \vdash \mathsf{de}_i \leadsto c_i \quad \mathsf{r}(c_1, \ldots, c_n) = \mathtt{true}}{\mathtt{m} \vdash \mathsf{r}(\mathsf{de}_1, \ldots, \mathsf{de}_n) \leadsto \mathtt{true}} \qquad \frac{\exists i.\ \mathtt{m} \vdash \mathsf{de}_i \leadsto \mathtt{merr}}{\mathtt{m} \vdash \mathsf{r}(\mathsf{de}_1, \ldots, \mathsf{de}_n) \leadsto \mathtt{merr}}$$

$$\frac{\mathtt{m}(\mathit{rv}) = a \neq \boxtimes \quad \mathtt{m}(a, \mathit{df}) = c}{\mathtt{m} \vdash \mathit{rv}{\longrightarrow}\mathit{df} \leadsto c} \qquad \frac{\mathtt{m}(\mathit{rv}) = \boxtimes}{\mathtt{m} \vdash \mathit{rv}{\longrightarrow}\mathit{df} \leadsto \mathtt{merr}}$$

$$\frac{\mathtt{m}(\mathit{rv}) = a \neq \boxtimes}{\mathtt{m} \vdash \mathtt{free}(\mathit{rv}); \leadsto \mathtt{m}[\mathit{rv} \leftarrow \boxtimes]} \qquad \frac{\mathtt{m}(\mathit{rv}) = \boxtimes}{\mathtt{m} \vdash \mathtt{free}(\mathit{rv}); \leadsto \mathtt{merr}}$$

$$\frac{\forall i.\ \mathtt{m} \vdash de_i \rightsquigarrow c_i \qquad r(c_1, \ldots, c_n) = \mathtt{true}}{\mathtt{m} \vdash r(de_1, \ldots, de_n) \rightsquigarrow \mathtt{true}} \qquad \frac{\exists i.\ \mathtt{m} \vdash de_i \rightsquigarrow \mathtt{merr}}{\mathtt{m} \vdash r(de_1, \ldots, de_n) \rightsquigarrow \mathtt{merr}}$$

$$\frac{\mathtt{m}(rv) = a \neq \boxtimes \qquad \mathtt{m}(a, df) = c}{\mathtt{m} \vdash rv{\longrightarrow}df \rightsquigarrow c} \qquad \frac{\mathtt{m}(rv) = \boxtimes}{\mathtt{m} \vdash rv{\longrightarrow}df \rightsquigarrow \mathtt{merr}}$$

$$\frac{\mathtt{m}(rv) = a \neq \boxtimes}{\mathtt{m} \vdash \mathtt{free}(rv); \rightsquigarrow \mathtt{m}[rv \leftarrow \boxtimes]} \qquad \frac{\mathtt{m}(rv) = \boxtimes}{\mathtt{m} \vdash \mathtt{free}(rv); \rightsquigarrow \mathtt{merr}}$$

No garbage detection

$$\frac{\forall i.\ \mathtt{m} \vdash de_i \leadsto c_i \quad r(c_1, \ldots, c_n) = \mathtt{true}}{\mathtt{m} \vdash r(de_1, \ldots, de_n) \leadsto \mathtt{true}} \qquad \frac{\exists i.\ \mathtt{m} \vdash de_i \leadsto \mathtt{merr}}{\mathtt{m} \vdash r(de_1, \ldots, de_n) \leadsto \mathtt{merr}}$$

$$\frac{\mathtt{m}(rv) = a \neq \boxtimes \quad \mathtt{m}(a, \mathit{df}) = c}{\mathtt{m} \vdash rv{\rightarrow}\mathit{df} \leadsto c} \qquad \frac{\mathtt{m}(rv) = \boxtimes}{\mathtt{m} \vdash rv{\rightarrow}\mathit{df} \leadsto \mathtt{merr}}$$

$$\frac{\mathtt{m}(rv) = a \neq \boxtimes}{\mathtt{m} \vdash \mathtt{free}(rv); \leadsto \mathtt{m}[rv \leftarrow \boxtimes]} \qquad \frac{\mathtt{m}(rv) = \boxtimes}{\mathtt{m} \vdash \mathtt{free}(rv); \leadsto \mathtt{merr}}$$

$$\frac{a \text{ fresh in } \mathbb{L}_{RT*} \quad \forall df \in RT.\ \mathtt{m}.H(a, df) = c \quad \forall rf \in RT.\ \mathtt{m}.H(a, rf) = \boxtimes}{\mathtt{m} \vdash rv{=}\ \mathtt{new}\ RT; \leadsto \mathtt{m}[rv \leftarrow a]}$$

$$\frac{\forall i.\ \mathtt{m} \vdash \mathtt{de}_i \rightsquigarrow \mathtt{c}_i \quad \mathtt{r}(\mathtt{c}_1, \ldots, \mathtt{c}_n) = \mathtt{true}}{\mathtt{m} \vdash \mathtt{r}(\mathtt{de}_1, \ldots, \mathtt{de}_n) \rightsquigarrow \mathtt{true}} \qquad \frac{\exists i.\ \mathtt{m} \vdash \mathtt{de}_i \rightsquigarrow \mathtt{merr}}{\mathtt{m} \vdash \mathtt{r}(\mathtt{de}_1, \ldots, \mathtt{de}_n) \rightsquigarrow \mathtt{merr}}$$

$$\frac{\mathtt{m}(rv) = a \neq \boxtimes \quad \mathtt{m}(a, \mathit{df}) = \mathtt{c}}{\mathtt{m} \vdash rv{\longrightarrow}\mathit{df} \rightsquigarrow \mathtt{c}} \qquad \frac{\mathtt{m}(rv) = \boxtimes}{\mathtt{m} \vdash rv{\longrightarrow}\mathit{df} \rightsquigarrow \mathtt{merr}}$$

$$\frac{\mathtt{m}(rv) = a \neq \boxtimes}{\mathtt{m} \vdash \mathtt{free}(rv); \rightsquigarrow \mathtt{m}[rv \leftarrow \boxtimes]} \qquad \frac{\mathtt{m}(rv) = \boxtimes}{\mathtt{m} \vdash \mathtt{free}(rv); \rightsquigarrow \mathtt{merr}}$$

$$\frac{a \text{ fresh in } \mathbb{L}_{RT*} \quad \forall \mathrm{df} \in RT.\ \mathtt{m}.H(a, \mathrm{df}) = \mathtt{c} \quad \forall \mathrm{rf} \in RT.\ \mathtt{m}.H(a, \mathrm{rf}) = \boxtimes}{\mathtt{m} \vdash rv{=}\ \mathtt{new}\ RT; \rightsquigarrow \mathtt{m}[\mathtt{m}\ldots}$$

Infinite heap

By default initialisation

$$\frac{\forall i.\ \mathtt{m} \vdash de_i \leadsto c_i \quad r(c_1, \ldots, c_n) = \mathtt{true}}{\mathtt{m} \vdash r(de_1, \ldots, de_n) \leadsto \mathtt{true}}$$

$$\frac{\exists i.\ \mathtt{m} \vdash de_i \leadsto \mathtt{merr}}{\mathtt{m} \vdash r(de_1, \ldots, de_n) \leadsto \mathtt{merr}}$$

$$\frac{\mathtt{m}(rv) = \text{[illegible]} = c}{\mathtt{m} \vdash rv \longrightarrow df \leadsto c}$$

See procedure call in the second part!

$$\frac{\mathtt{m}(rv) = \boxtimes}{\mathtt{m} \vdash rv \longrightarrow df \leadsto \mathtt{merr}}$$

$$\frac{\mathtt{m}(rv) = a \neq \boxtimes}{\mathtt{m} \vdash \mathtt{free}(rv); \leadsto \mathtt{m}[rv \leftarrow \boxtimes]}$$

$$\frac{\mathtt{m}(rv) = \boxtimes}{\mathtt{m} \vdash \mathtt{free}(rv); \leadsto \mathtt{merr}}$$

$$\frac{a \text{ fresh in } \mathbb{L}_{RT*} \quad \forall df \in RT.\ \mathtt{m}.H(a, df) = c \quad \forall rf \in RT.\ \mathtt{m}.H(a, rf) = \boxtimes}{\mathtt{m} \vdash rv \mathtt{=\ new}\ RT; \leadsto \mathtt{m}[\text{[illegible]}}$$

Infinite heap

By default initialisation

Definition

An inter-procedural control flow graph (ICFG) over a set of operations $\mathtt{Op}$ is a tuple $\langle V, \mathtt{Op}, \longrightarrow, \mathit{start}, \mathit{end}\rangle$ where:

- V is a finite set of vertices,
- $\mathit{start} \in V$ is a *starting vertex* and $\mathit{end} \in V$ is a *final vertex*,
- $\mathtt{Op}$ is a finite set of *labels*,
- $\longrightarrow \in V \times \mathtt{Op} \times V$ is a finite set of *edges*.

For our class of program

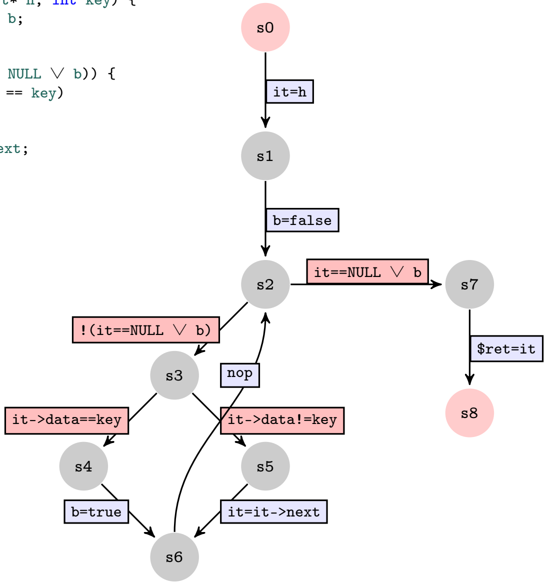
$$\mathtt{Op} \ni \mathrm{op} \ ::= \ \mathrm{be} \mid \mathrm{astmt} \mid \mathtt{call}\ v = \mathrm{P}(\ldots) \mid \mathtt{return}\ v = \mathrm{P}(\ldots)$$

where (recall)

$$\begin{array}{rcl} \mathrm{be} & ::= & \mathrm{bcst} \mid \mathrm{bv} \mid \mathrm{r}(\overrightarrow{\mathrm{de}}) \mid \mathrm{rv}_1{==}\mathrm{rv}_2 \mid \,!\mathrm{be} \mid \mathrm{be} \wedge \mathrm{be} \mid \mathrm{be} \vee \mathrm{be} \\ \mathrm{astmt} & ::= & \mathrm{dv}{=}\mathrm{de} \mid \mathrm{rv}{\rightarrow}\mathrm{df}{=}\mathrm{de} \mid \mathrm{rv}{=}\mathrm{re} \mid \mathrm{rv}{\rightarrow}\mathrm{rf}{=}\mathrm{re} \mid \mathrm{bv}{=}\mathrm{be} \\ & \mid & \mathrm{rv}{=}\mathtt{new}\ \mathrm{RT} \mid \mathtt{free}(\mathrm{rv}) \mid \mathtt{nop} \end{array}$$

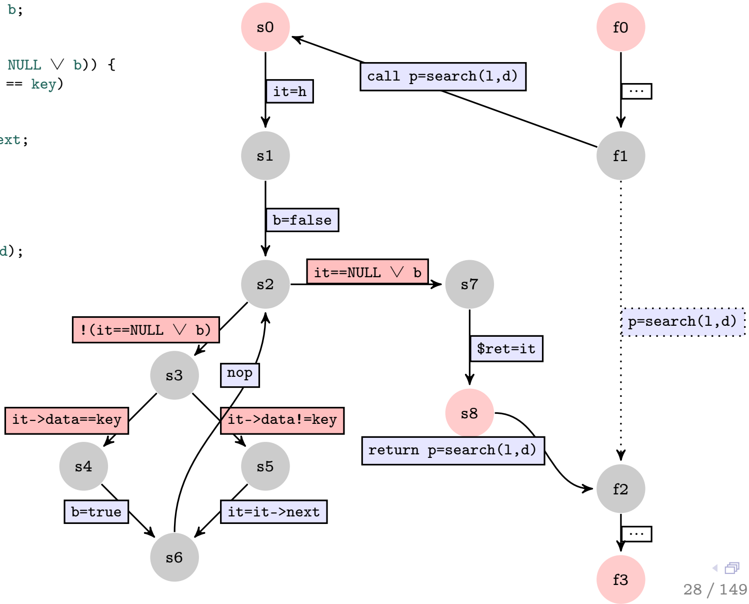
ICFG for search

```
    list* search(list* h, int key) {
      list* it; bool b;
s0:   it = h;
s1:   b = false;
s2:   while (!(it == NULL ∨ b)) {
s3:     if (it->data == key)
s4:       b = true;
        else
s5:       it = it->next;
s6:   }
s7:   return it;
s8: }
```



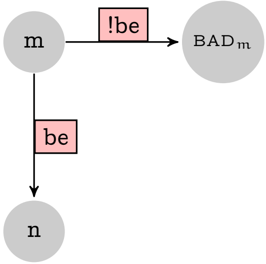
ICFG for search and foo

```
    list* search(list* h, int key) {
      list* it; bool b;
s0:   it = h;
s1:   b = false;
s2:   while (!(it == NULL ∨ b)) {
s3:     if (it->data == key)
s4:       b = true;
        else
s5:       it = it->next;
s6:   }
s7:   return it;
s8: }
    int foo() {
f0:   ...
f1:   p = search(l, d);
f2:   ...
f3: }
```

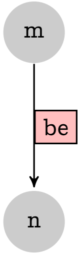


ICFG for User Assertions

```
m:  assert(be);
n:
```




```
m:  assume(be);
n:
```



ICFG is a finite, syntactic object.

The interpretation of ICFG using the natural semantics produces a model of program executions, *i.e.* LTS.

Definition LTS

A labeled transition system is a tuple $\langle \mathrm{C}, \mathrm{Init}, \mathrm{Out}, \Sigma, \rightarrow \rangle$ where:

- C is a set of *configurations*,
- $\mathrm{Init} \in \mathrm{C}$ and $\mathrm{Out} \in \mathrm{C}$ are sets of initial and exit configurations,
- Σ is a finite set of *actions*,
- $\rightarrow \subseteq \mathrm{C} \times \Sigma \times \mathrm{C}$ is a set of transitions.

LTS is an infinite, semantic object.

A control path is a path in the control flow graph:

$$q_0 \xrightarrow{op_0} q_1 \ldots q_k \xrightarrow{op_k} q_{k+1}$$

An execution path is a path in the labeled transition system:

$$(q_0, m_0) \xrightarrow{op_0} (q_1, m_1) \ldots (q_k, m_k) \xrightarrow{op_k} (q_{k+1}, m_{k+1})$$

A run is an execution path starting from the initial configuration:

$$(q_{Init}, m_{Init}) \xrightarrow{op_0} (q_1, m_1) \ldots (q_k, m_k) \xrightarrow{op_k} (q_{k+1}, m_{k+1})$$

Goal

Over-approximate the set of configurations reachable from the initial configuration.

The exact set of reachable configurations is:

$$\begin{aligned} \mathrm{Post}^* &= \bigcup_{\sigma\ :\ \mathtt{run}} \{(q, m) \mid (q, m) \text{ occurs in } \sigma\} \\ &= \bigcup_{q_{\mathtt{Init}} \xrightarrow{\mathtt{op}_0} \ldots \xrightarrow{\mathtt{op}_k} q} \{q\} \times \left(\mathtt{post}_{\mathtt{op}_k} \circ \ldots \circ \mathtt{post}_{\mathtt{op}_0}\right)(m_{\mathtt{Init}}) \end{aligned}$$

where

$$\mathtt{post}_{\mathtt{op}} \ :\ \mathtt{Mem} \cup \{\mathtt{merr}\} \nrightarrow \mathtt{Mem} \cup \{\mathtt{merr}\}$$

defined by the operational semantics, *i.e.* standard semantics.

We focus on forward analysis. Exercise: Transpose to backward analysis.

Goal

Over-approximate the set of configurations reachable from the initial configuration at each program point.

Project Post* on each program point (ICFG vertex):

$$\mathrm{Post}^* = \bigcup_{q \in \mathrm{ICFG}} q \mapsto \overline{\mathrm{Post}}^*(q) \qquad \text{with } q \mapsto \emptyset \equiv \emptyset,$$

$$\overline{\mathrm{Post}}^*(q) = \bigcup_{q_{\mathrm{Init}} \xrightarrow{op_0} \ldots \xrightarrow{op_{k-1}} q_k \xrightarrow{op_k} q} \left(\overline{\mathrm{post}}_{op_k} \circ \ldots \circ \overline{\mathrm{post}}_{op_0}\right)(\{m_{\mathrm{Init}}\})$$

and

$$\overline{\mathrm{post}}_{op} : \mathcal{P}(\mathtt{Mem} \cup \{\mathtt{merr}\}) \longrightarrow \mathcal{P}(\mathtt{Mem} \cup \{\mathtt{merr}\})$$

is the collecting semantics, $\overline{\mathrm{post}}_{op}(M) = \cup_{m \in M}\, \mathrm{post}_{op}(m)$.

$\overline{\text{Post}}^*$ is called $\overrightarrow{\text{MOP}}$ for (forward) "Meet Over All Paths" and is the most precise abstraction of the reachable configurations.

However, in the presence of control loops, the set of runs is infinite, so:

$\overrightarrow{\text{MOP}}$ is not computable in general!

Sound Solution

Over-approximate the initial system of equations over runs to a system of in-equations over ICFG edges:

$$\begin{aligned} \overline{\text{Post}}^*(q_{\text{Init}}) &\supseteq \{m_{\text{Init}}\} \\ \overline{\text{Post}}^*(q') &\supseteq \overline{\text{post}}_{\text{op}}(\overline{\text{Post}}^*(q)) \text{ for all } q \xrightarrow{\text{op}} q' \in \text{ICFG} \end{aligned}$$

$\overline{\text{Post}}^*$ is called $\overrightarrow{\text{MOP}}$ for (forward) "Meet Over All Paths" and is the most precise abstraction of the reachable configurations.

However, in the presence of control loops, the set of runs is infinite, so:

$\overrightarrow{\text{MOP}}$ is not computable in general!

Sound Solution

Over-approximate the initial system of equati… a system of in-equations over ICFG ed…

Do solutions always exist? Yes, see Knaster-Tarski Fixpoint Theorem!

$$\overline{\text{Post}}^*(q_{\text{Init}}) \supseteq \{m_{\text{Init}}\}$$

$$\overline{\text{Post}}^*(q') \supseteq \overline{\text{post}}_{\text{op}}(\overline{\text{Post}}^*(q)) \text{ for all } q \xrightarrow{\text{op}} q' \in \text{ICFG}$$

Complete Lattice

Definition

A partially ordered set $(L, \sqsubseteq)$ is a complete lattice if every $X \subseteq L$ has both a greatest lower bound $\sqcap X$ and a least upper bound $\sqcup X$ in $(L, \sqsubseteq)$.

In a complete lattice $(L, \sqsubseteq)$

- $\sqcup\, X$ is the most precise information consistent with all $x \in X$
- $\sqcap\, X$ is the infimum of X, i.e., $\sqcup\, \{x \mid x \sqsubseteq X\}$
- least element exists $\bot$, $\bot = \sqcup\, L = \sqcap\, \emptyset$
- greatest element exists $\top$, $\top = \sqcup\, \emptyset = \sqcap\, L$

Example: Powerset Lattice

For any set S, $(\mathcal{P}(S), \subseteq)$ is a complete lattice.

Definitions

Let $(L, \sqsubseteq)$ be a partial order,

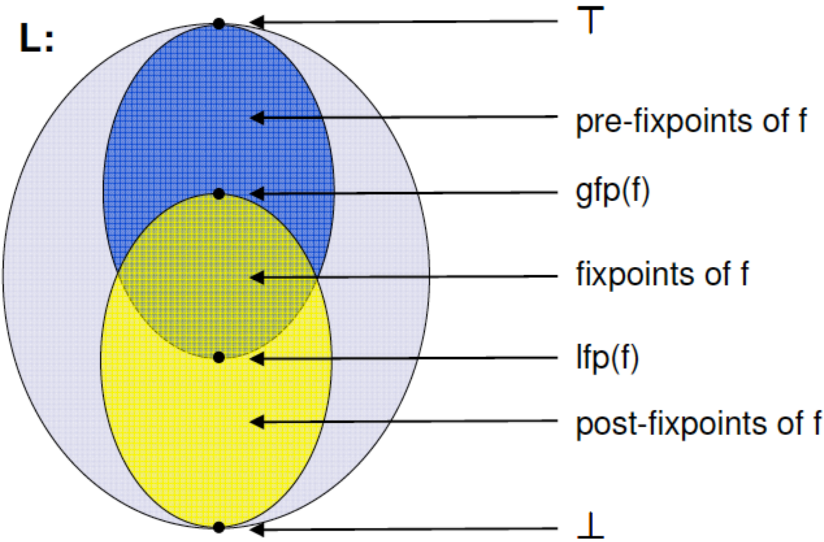
- $f : L \to L$ is monotonic iff $\forall x, y \in L.\ x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$.
- $x \in L$ is a fixpoint of f iff $f(x) = x$.

Theorem Knaster-Tarski

Let L be a complete lattice and $f : L \to L$ a monotonic function. The set of fixpoints of f is also a complete lattice.

Consequently, least fixpoint $\mathtt{lfp}(f)$ and greatest fixpoint $\mathtt{gfp}(f)$ exist and:

$\mathtt{lfp}(f)$	$=$	$\sqcap \{x \in L \mid f(x) \sqsubseteq x\}$	least pre-fixpoint
$\mathtt{gfp}(f)$	$=$	$\sqcup \{x \in L \mid x \sqsubseteq f(x)\}$	greatest post-fixpoint



Picture from: Nielson/Nielson/Hankin, *Principles of Program Analysis*

Reformulate Our Goal

To avoid loss of precision, we focus on the least fixpoint of the system:

$$\begin{aligned} \overline{\mathrm{Post}}^*(q_{\mathrm{Init}}) &\supseteq \{m_{\mathrm{Init}}\} \\ \overline{\mathrm{Post}}^*(q') &\supseteq \overline{\mathrm{post}}_{op}(\overline{\mathrm{Post}}^*(q)) \text{ for all } q \xrightarrow{op} q' \in \mathrm{ICFG} \end{aligned}$$

called $\overrightarrow{\mathrm{MFP}}$ for (forward) "Maximal Fixpoint".

How to compute the smallest solution? See Kleene iteration!

Kleene Fixpoint Theorem

Let $(L, \sqsubseteq)$ be a complete partial order and $f : L \to L$ monotonic. Then $\mathtt{lfp}(f)$ is the supremum of the ascending Kleene chain of f, *i.e.*

$$\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \sqsubseteq \ldots \sqsubseteq f^n(\bot) \sqsubseteq \ldots$$

Observe that if $f^i(\bot) = f^{i+1}(\bot)$ for some i, then $f^i(\bot)$ is $\mathtt{lfp}(f)$.

Definition

$(L, \sqsubseteq)$ satisfies the ascending chain condition if every ascending chain $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ of elements of L is eventually stationary.

Termination

$(f^i(\bot))_{i \in \mathbb{N}}$ converges for $(L, \sqsubseteq)$ satisfying the ascending chain condition.

Improved Kleene Iteration: Workset Algorithm

```
W = ∅;
for (all vertex q) { P[q] = ⊥; W = Add(W,q); }
P[qInit] = { mInit };
/* ∀q. P[q] ⊑ MFP→(q)  ∧  {m_Init} ⊑ P[q_Init]  ∧
     ∀q′ ∉ W. post‾_op(P[q]) ⊑ P[q′] with (q, op, q′) edge */
while (W != ∅) {
q = Extract(W);
for (all edge (q,op,r)) {
  t = post‾_op (P[q]);
  if (! (t ⊑ P[r])) {
    P[r] = P[r] ⊔ t;
    W = Add(W,r);
} }
} /* ∀q. P[q] ⊑ MFP→(q)  ∧  P solution ⟹  P = MFP→ */
```

Invariant: $\forall q.\ P[q] \sqsubseteq \overrightarrow{MFP}(q) \ \wedge\ \{m_{Init}\} \sqsubseteq P[q_{Init}] \ \wedge\ \forall q' \notin W.\ \overline{\text{post}}_{op}(P[q]) \sqsubseteq P[q']$ with (q, op, q') edge

Postcondition: $\forall q.\ P[q] \sqsubseteq \overrightarrow{MFP}(q) \ \wedge\ P$ solution $\implies P = \overrightarrow{MFP}$

Workset Algorithm Analysis

Termination

```
8:    if (! (t ⊑ P[r])) {
9:      P[r] = P[r] ⊔ t;
10:     W = Add(W,r);
11:   }
```

WS terminates if $(L, \sqsubseteq)$ satisfies the ascending chain condition.

> *For any vertex* r *the following sequence converges:*
> $\bot \sqsubseteq P[r] \sqsubseteq P^2[r] \sqsubseteq \ldots$ *where* $P^k[r]$ *is the value of* `P[r]` *after visiting* k *edges with target* r*.*

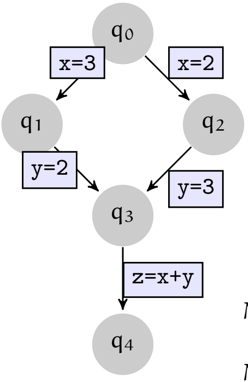
Otherwise, change computation at line 9 to obtain convergence of $(P^i[r])_{i \in \mathbb{N}}$ for any r, *e.g.* widening (see later).

Variants

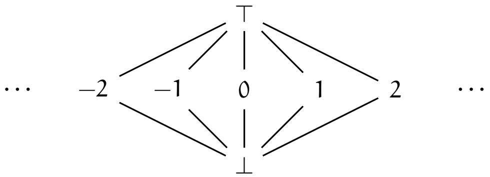
Different iteration strategies are obtained by changing the selection of the visited vertices (`Extract`) and edges (line 6).

Precision of MFP

For monotonic F, $\overrightarrow{MOP}[q] \sqsubseteq \overrightarrow{MFP}[q]$ for any reachable vertex q.



$(L, \sqsubseteq)$:



$$
\begin{aligned}
MOP[q_4] &= (x \mapsto 3, y \mapsto 2, z \mapsto 5) \sqcup (x \mapsto 2, y \mapsto 3, z \mapsto 5) \\
&= (x \mapsto \top, y \mapsto \top, z \mapsto 5) \\
MFP[q_3] &= (x \mapsto 3, y \mapsto 2, z \mapsto \bot) \sqcup (x \mapsto 2, y \mapsto 3, z \mapsto \bot) \\
&= (x \mapsto \top, y \mapsto \top, z \mapsto \bot) \\
MFP[q_4] &= (x \mapsto \top, y \mapsto \top, z \mapsto \top)
\end{aligned}
$$

Complete Lattice: Examples

Power Set

For any set S, $(\mathcal{P}(S), \sqsubseteq)$ is a complete lattice where

$$\sqsubseteq \;=\; \subseteq, \qquad \sqcap = \bigcap, \qquad \sqcup = \bigcup, \qquad \bot = \emptyset, \qquad \top = S$$

If S is finite then $(\mathcal{P}(S), \sqsubseteq)$ satisfies a.c.c.

Functions

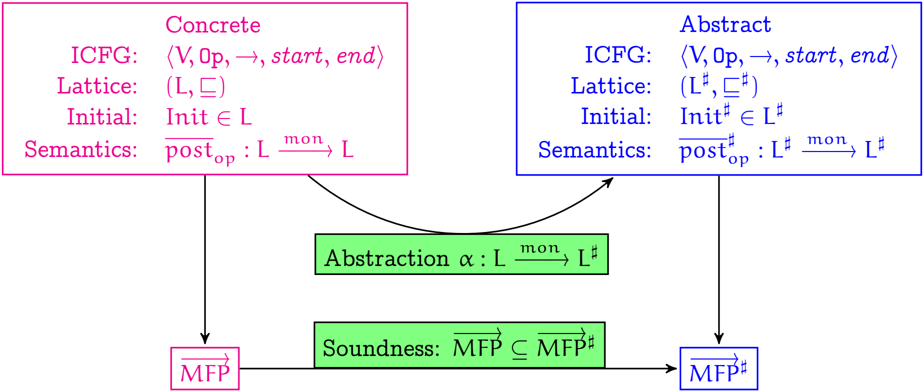
For any set S and a complete lattice $(L, \sqsubseteq)$, $(S \to L, \sqsubseteq)$ is a complete lattice where

$$f \sqsubseteq g \quad \text{if} \quad \forall x \in S.\; f(x) \sqsubseteq g(x)$$

$$\sqcap F \;=\; \lambda x. \sqcap \{f(x) \mid f \in F\}, \qquad \sqcup F \;=\; \lambda x. \sqcup \{f(x) \mid f \in F\},$$

$$\bot \;=\; \lambda x.\bot, \qquad \top \;=\; \lambda x.\top$$

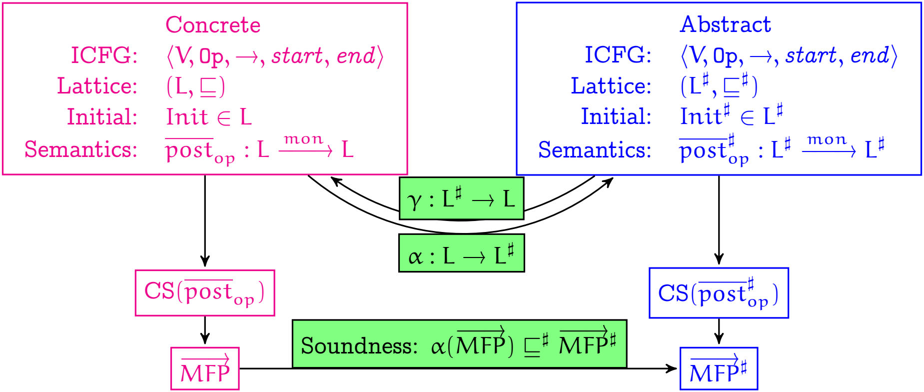
If S is finite and $(L, \sqsubseteq)$ satisfies a.c.c. then $(S \to L, \sqsubseteq)$ satisfies a.c.c.



How to systematically ensure the correctness of this principle?

⟶ Abstract Interpretation [Cousot&Cousot,79].

Abstract Interpretation



Conditions

- Correct abstraction: (α, γ) is a Galois connection.
- Correct interpretation: $\alpha(\overline{\mathrm{post}}_{\mathrm{op}}(x)) \sqsubseteq^\sharp \overline{\mathrm{post}}^\sharp_{\mathrm{op}}(\alpha(x))$

Definition

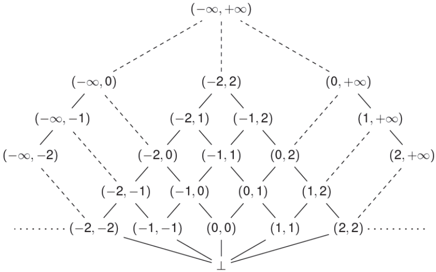
A Galois connection between two lattices $(L, \sqsubseteq)$ and $(L^\sharp, \sqsubseteq^\sharp)$ is a pair of functions (α, γ) with $\alpha : L \to L^\sharp$ and $\gamma : L^\sharp \to L$ satisfying, for all $x \in L$ and $y^\sharp \in L^\sharp$:

$$\alpha(x) \quad \sqsubseteq^\sharp \quad y^\sharp \qquad \text{iff} \qquad x \quad \sqsubseteq \quad \gamma(y^\sharp)$$

Vocabulary and intuition:

- γ is the concretisation function,
  $\longrightarrow$ $\gamma(y^\sharp)$ is the concrete value represented by $y^\sharp$.
- α is the abstraction function,
  $\longrightarrow$ $\alpha(x)$ is the most precise abstract value representing x
  $\longrightarrow$ concretisation of $\alpha(x)$ approximates x, *i.e.* $\sqsupseteq x$.

$$(L, \sqsubseteq) = (\mathcal{P}(\mathbb{Z}), \subseteq) \quad \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \quad (\mathrm{Int}, \subseteq)$$

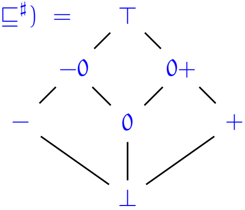


$$S \subset \mathbb{Z} \xrightarrow{\alpha} (\inf(S), \sup(S)) \quad e.g. \quad \{-1, 2\} \xrightarrow{\alpha} (-1, 2)$$

$$\{\ell, \ell+1, \ldots, u\} \xleftarrow{\gamma} (\ell, u) \quad e.g. \quad \{-1, 0, 1, 2\} \xleftarrow{\gamma} (-1, 2)$$

Exercise: Explicit the Galois connection for the Sign Abstraction, *i.e.*,

$$(L, \sqsubseteq) \;=\; (\mathcal{P}(\mathbb{R}), \subseteq) \quad \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \quad (L^\sharp, \sqsubseteq^\sharp) \;=$$



$$e.g.,\ \alpha(x) = \begin{cases} \bot & \text{if } x = \emptyset \\ + & \text{if } x \subseteq \{r \mid r > 0\} \\ \ldots & \ldots \end{cases}$$

Let $(L, \sqsubseteq)$ and $(L^\sharp, \sqsubseteq^\sharp)$ be two lattices.

For any two functions $\alpha : L \to L^\sharp$ and $\gamma : L^\sharp \to L$,

$$(L, \sqsubseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (L^\sharp, \sqsubseteq^\sharp) \quad \text{iff} \quad \begin{cases} x \sqsubseteq \gamma(\alpha(x)) & \text{for any } x \in L \\ \alpha(\gamma(y^\sharp)) \sqsubseteq^\sharp y^\sharp & \text{for any } y^\sharp \in L^\sharp \\ \alpha \text{ is monotonic} & \\ \gamma \text{ is monotonic} & \end{cases}$$

Correct Semantics Interpretation

Given a monotonic function $f : L \longrightarrow L$, let consider a monotonic function $g^\sharp : L^\sharp \longrightarrow L^\sharp$ that is a sound approximation of f, *i.e.*

$$\alpha \circ f(x) \;\sqsubseteq^\sharp\; g^\sharp \circ \alpha(x)$$

Theorem

For any monotonic function $f : L \longrightarrow L$ and any monotonic sound approximation of f, $g^\sharp : L^\sharp \longrightarrow L^\sharp$ then

$$\mathtt{lfp}(f) \;\sqsubseteq\; \gamma\big(\mathtt{lfp}(g^\sharp)\big)$$

Definition

A sound approximation of $\overline{\mathrm{post}}_{\mathrm{op}}$ is called (correct) abstract transformer.

Example: Abstract Transformer for Intervals

Let consider the Galois connection $(\mathcal{P}(\mathbb{Z}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathsf{Int}, \sqsubseteq)$

The concrete transformer for op $\equiv (z \geq 0)$ is

$$\overline{\mathtt{post}}_{z \geqslant 0}(S) \;=\; \{v \in S \mid v \geq 0\} \qquad \forall S \subseteq \mathbb{Z}$$

Example: Abstract Transformer for Intervals

Let consider the Galois connection $(\mathcal{P}(\mathbb{Z}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathrm{Int}, \subseteq)$

The concrete transformer for op $\equiv (z \geq 0)$ is

$$\overline{\mathtt{post}}_{z \geqslant 0}(S) \;=\; \{v \in S \mid v \geq 0\} \qquad \forall S \subseteq \mathbb{Z}$$

Several abstract transformers may be defined:

- $g^\sharp_{z\geqslant 0}((\ell, u)) \;=\; (\max(0, \ell), u)$ $\qquad (\ell, u) = \bot$ if $\ell > u$
- $h^\sharp_{z\geqslant 0}((\ell, u)) \;=\; (\max(0, \ell), \infty)$
- $f^\sharp_{z\geqslant 0}((\ell, u)) \;=\; \top$

and notice that $g^\sharp_{z\geqslant 0} \sqsubseteq^\sharp h^\sharp_{z\geqslant 0} \sqsubseteq f^\sharp_{z\geqslant 0}$.

Example: Abstract Transformer for Intervals

Let consider the Galois connection $(\mathcal{P}(\mathbb{Z}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\text{Int}, \sqsubseteq)$

The concrete transformer for $\text{op} \equiv (z \geq 0)$ is

$$\overline{\texttt{post}}_{z \geqslant 0}(S) \;=\; \{v \in S \mid v \geq 0\} \qquad \forall S \subseteq \mathbb{Z}$$

Several abstract transformers may be defined:

- $g^\sharp_{z\geqslant 0}((\ell, u)) \;=\; (\max(0, \ell), u)$ $\qquad (\ell, u) = \bot$ if $\ell > u$
- $h^\sharp_{z\geqslant 0}((\ell, u)) \;=\; (\max(0, \ell), \infty)$
- $f^\sharp_{z\geqslant 0}((\ell, u)) \;=\; \top$

and notice that $g^\sharp_{z\geqslant 0} \sqsubseteq^\sharp h^\sharp_{z\geqslant 0} \sqsubseteq f^\sharp_{z\geqslant 0}$.

What happens with $\texttt{lfp}$ (recall, used in $\overrightarrow{\text{MFP}}$) of $g^\sharp_{z\geqslant 0}, h^\sharp_{z\geqslant 0}, f^\sharp_{z\geqslant 0}$?

Theorem

For any two monotonic functions f, g on a complete lattice $(L, \sqsubseteq)$, if $f(x) \sqsubseteq g(x)$ for all $x \in L$ then $\texttt{lfp}(f) \sqsubseteq \texttt{lfp}(g)$.

In the previous example, $g^\sharp$ is better than $h^\sharp$!

Theorem

For any two monotonic functions f, g on a complete lattice $(L, \sqsubseteq)$, if $f(x) \sqsubseteq g(x)$ for all $x \in L$ then $\texttt{lfp}(f) \sqsubseteq \texttt{lfp}(g)$.

In the previous example, $g^\sharp$ is better than $h^\sharp$!

Definition

For any monotonic function $f : L \longrightarrow L$, the best abstraction of f is the monotonic function $f^\sharp : L^\sharp \longrightarrow L^\sharp$ defined by:

$$f^\sharp = \alpha \circ f \circ \gamma$$

Theorem

For any two monotonic functions f, g on a complete lattice $(L, \sqsubseteq)$, if $f(x) \sqsubseteq g(x)$ for all $x \in L$ then $\texttt{lfp}(f) \sqsubseteq \texttt{lfp}(g)$.

In the previous example, $g^\sharp$ is better than $h^\sharp$!

Definition

For any monotonic function $f : L \to L$, the best abstraction of f is the monotonic function $f^\sharp : L^\sharp \to L^\sharp$ defined by:

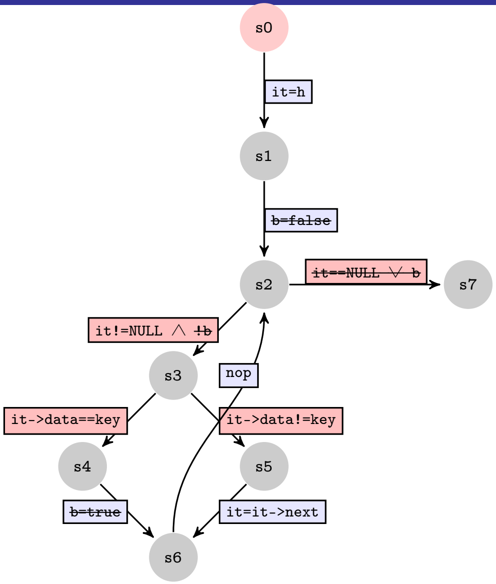
$$f^\sharp = \alpha \circ f \circ \gamma$$

But the best abstract transformer is difficult to compute!
$\longrightarrow$ *e.g.*, $\overline{\texttt{post}}^\sharp_{z=e*e}$ for sign abstraction needs to solve $e * e = 0$.

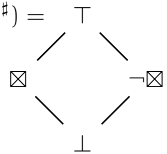
1. Design an abstract complete lattice $(L^\sharp, \sqsubseteq^\sharp)$, simpler than the concrete one $(L, \sqsubseteq)$, and formalise the "meaning" of abstract values by a Galois connection $(L, \sqsubseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (L^\sharp, \sqsubseteq^\sharp)$
   $\longrightarrow$ tests for equality with $\top^\sharp, \bot^\sharp$, algorithms for $\sqsubseteq^\sharp, \sqcup^\sharp, \ldots$

2. Design a sound abstract transformer $g^\sharp$ for each op $\in$ OP in ICFG.
   $\longrightarrow$ based on the natural semantics, try to be precise and efficient

3. Compute $\texttt{lfp}(g^\sharp)$ using the some algorithm (*e.g.*, workset) to obtain an over-approximation of $\overrightarrow{\mathrm{MFP}}$.
   $\longrightarrow$ generic algorithm with parameters $(L^\sharp, \sqsubseteq^\sharp)$, ICFG, and $g^\sharp$

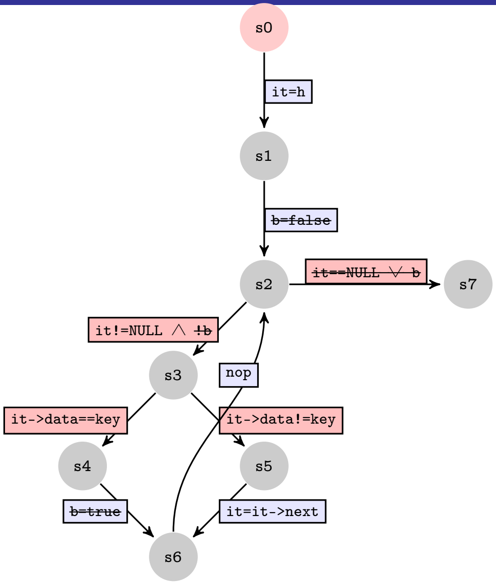


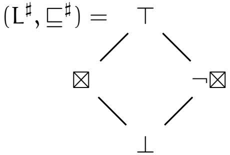
$(L^\sharp, \sqsubseteq^\sharp) =$



Initially:

CP	it	h	W
s0	$\bot$	$\top$	✓
s1	$\bot$	$\bot$	
s2	$\bot$	$\bot$	
s3	$\bot$	$\bot$	
s4	$\bot$	$\bot$	
s5	$\bot$	$\bot$	
s6	$\bot$	$\bot$	
s7	$\bot$	$\bot$	

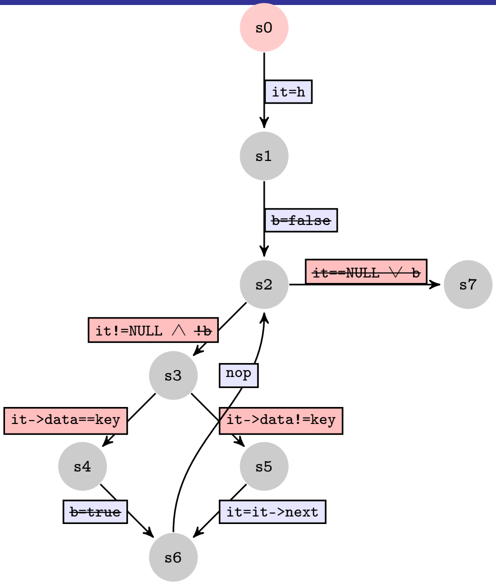




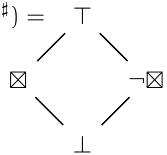
s0 extracted:

CP	it	h	W
s0	⊥	⊤	
s1	⊤	⊤	
s2	⊤	⊤	✓
s3	⊥	⊥	
s4	⊥	⊥	
s5	⊥	⊥	
s6	⊥	⊥	
s7	⊥	⊥	

$$\overline{\text{post}}^\sharp_{\texttt{it=h}}(\text{it} \mapsto v^\sharp, \text{h} \mapsto u^\sharp) = (\text{it} \mapsto u^\sharp, \text{h} \mapsto u^\sharp)$$



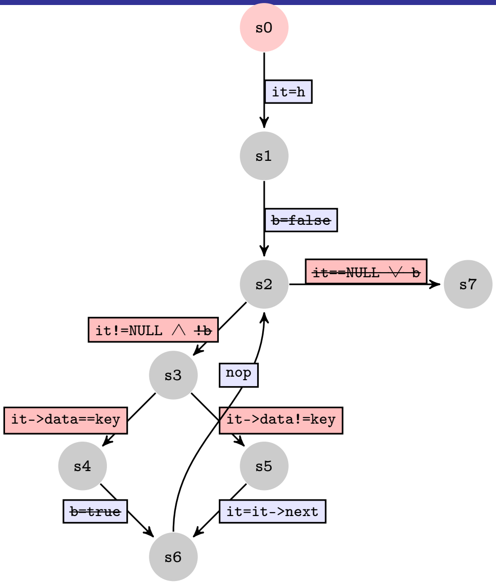
$(L^\sharp, \sqsubseteq^\sharp) =$



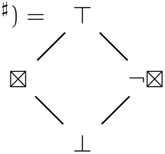
s2 extracted:

CP	it	h	W
s0	$\bot$	$\top$	
s1	$\top$	$\top$	
s2	$\top$	$\top$	
s3	$\neg\boxtimes$	$\top$	✓
s4	$\bot$	$\bot$	
s5	$\bot$	$\bot$	
s6	$\bot$	$\bot$	
s7	$\top$	$\top$	✓

$$\overline{\text{post}}^\sharp_{\texttt{it!=NULL}}(\text{it} \mapsto v^\sharp, \text{h} \mapsto u^\sharp) = (\text{it} \mapsto v^\sharp \sqcap^\sharp \neg\boxtimes, \text{h} \mapsto u^\sharp)$$

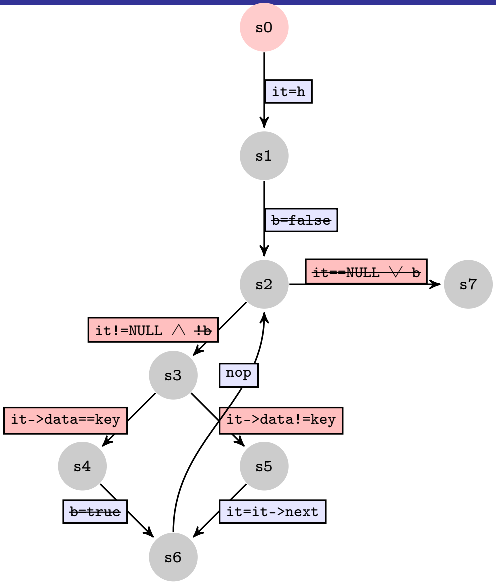


$(L^\sharp, \sqsubseteq^\sharp) =$

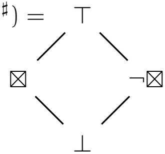


s3 extracted:

CP	it	h	W
s0	$\bot$	$\top$	
s1	$\top$	$\top$	
s2	$\top$	$\top$	
s3	¬⊠	$\top$	
s4	¬⊠	$\top$	✓
s5	¬⊠	$\top$	✓
s6	$\bot$	$\bot$	
s7	$\top$	$\top$	✓

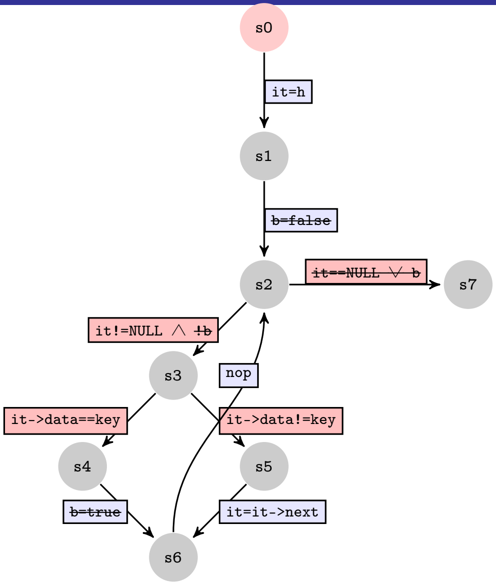


$(L^\sharp, \sqsubseteq^\sharp) =$

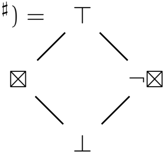


s4 extracted:

CP	it	h	W
s0	$\bot$	$\top$	
s1	$\top$	$\top$	
s2	$\top$	$\top$	
s3	¬⊠	$\top$	
s4	¬⊠	$\top$	
s5	¬⊠	$\top$	✓
s6	¬⊠	$\top$	✓
s7	$\top$	$\top$	✓



$(L^\sharp, \sqsubseteq^\sharp) =$

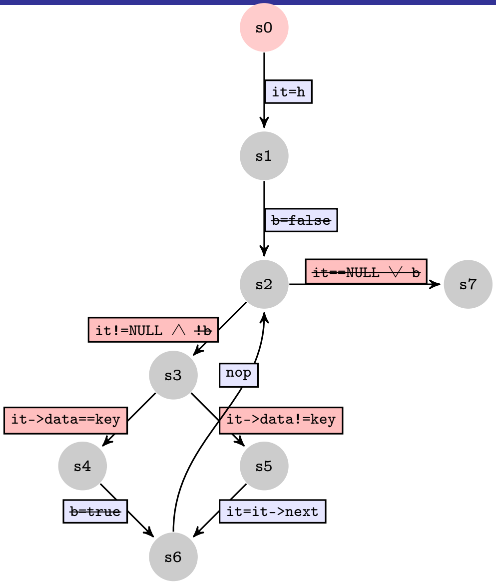


s5 extracted:

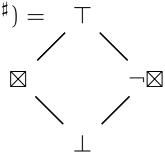
CP	it	h	W
s0	$\bot$	$\top$	
s1	$\top$	$\top$	
s2	$\top$	$\top$	
s3	$\neg\boxtimes$	$\top$	
s4	$\neg\boxtimes$	$\top$	
s5	$\neg\boxtimes$	$\top$	
s6	$\top$	$\top$	✓
s7	$\top$	$\top$	✓

$$\overline{\text{post}}^\sharp_{\texttt{it=it->next}}(\text{it} \mapsto v^\sharp, \text{h} \mapsto u^\sharp) = (\text{it} \mapsto \top, \text{h} \mapsto u^\sharp)$$

Example: Analysis of Null Aliasing [CC'77]



$(L^\sharp, \sqsubseteq^\sharp) =$



s6 extracted:

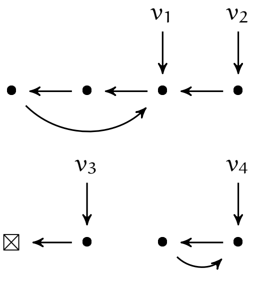
CP	it	h	W
s0	$\bot$	$\top$	
s1	$\top$	$\top$	
s2	$\top$	$\top$	
s3	¬⊠	$\top$	
s4	¬⊠	$\top$	
s5	¬⊠	$\top$	
s6	$\top$	$\top$	
s7	$\top$	$\top$	✓

Abstraction Idea

Partition the set of list variables (except NULL) such that:

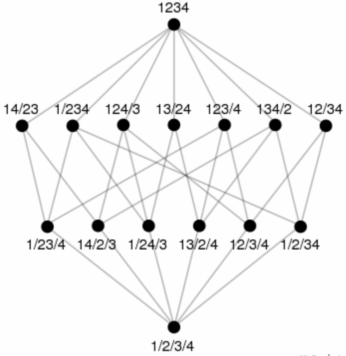
- v_1, v_2 belong to the same partition if $v_1 \xrightarrow{next^*} \cap\, v_2 \xrightarrow{next^*}$ may be non-empty,
- otherwise v_1, v_2 are in different partitions.

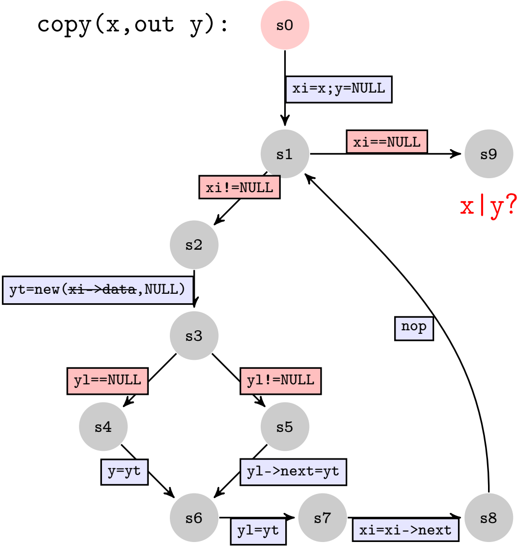
$\longrightarrow$ the abstraction keep track of relation between variables



12/34 or 12/3/4

$(P_4, \sqsubseteq^\sharp) =$





$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

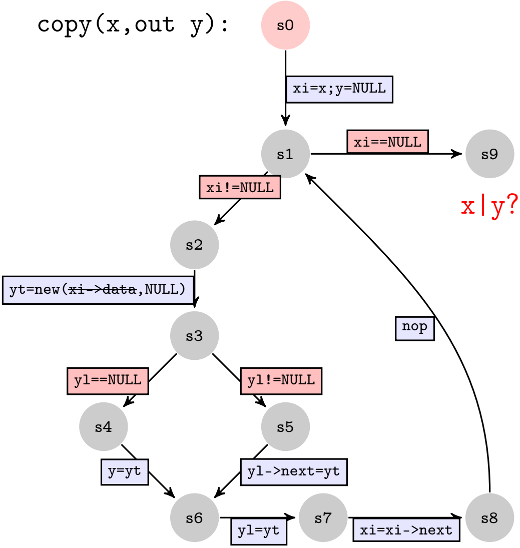
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

Initially:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	✓
s1	$\bot$	
s2	$\bot$	
s3	$\bot$	
s4	$\bot$	
s5	$\bot$	
s6	$\bot$	
s7	$\bot$	
s8	$\bot$	
s9	$\bot$	

Example: Analysis of Heap Separation [CC'77]

copy(x,out y):



$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

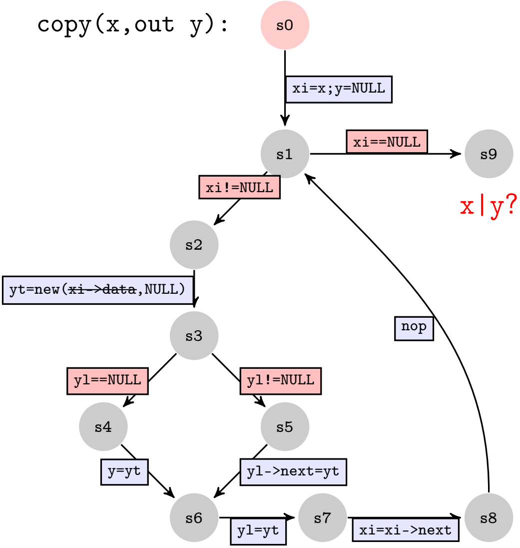
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s0 extracted:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y|yl yt	✓
s2	$\bot$	
s3	$\bot$	
s4	$\bot$	
s5	$\bot$	
s6	$\bot$	
s7	$\bot$	
s8	$\bot$	
s9	$\bot$	

$\overline{\texttt{post}}^\sharp_{\texttt{xi=x}}(\nu^\sharp) = \text{Extract}(\texttt{xi}, \nu^\sharp) \sqcup^\sharp \{\texttt{x}, \texttt{xi}\}$ $\overline{\texttt{post}}^\sharp_{\texttt{y=NULL}}(\nu^\sharp) = \text{Extract}(\texttt{y}, \nu^\sharp)$

Example: Analysis of Heap Separation [CC'77]



$v^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in v^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$v^\sharp \sqcup^\sharp u^\sharp$ based on union-find

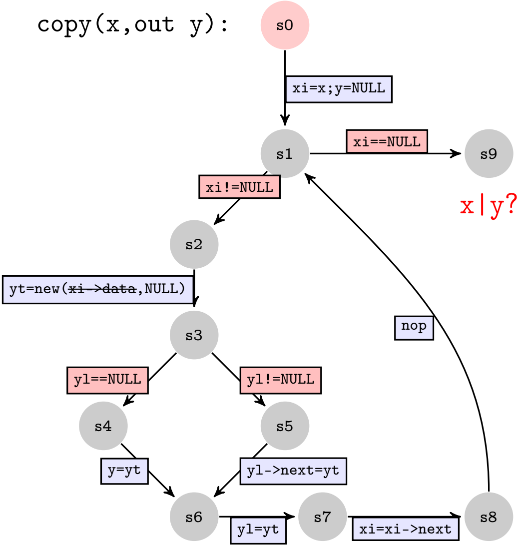
s1 extracted:

CP	$v^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y|yl yt	
s2	x xi|y|yl yt	✓
s3	$\bot$	
s4	$\bot$	
s5	$\bot$	
s6	$\bot$	
s7	$\bot$	
s8	$\bot$	
s9	x xi|y|yl yt	✓

$$\overline{\text{post}}^\sharp_{\texttt{xi==NULL}}(v^\sharp) = \overline{\text{post}}^\sharp_{\texttt{xi!=NULL}}(v^\sharp) = v^\sharp$$

Example: Analysis of Heap Separation [CC'77]

`copy(x,out y):`



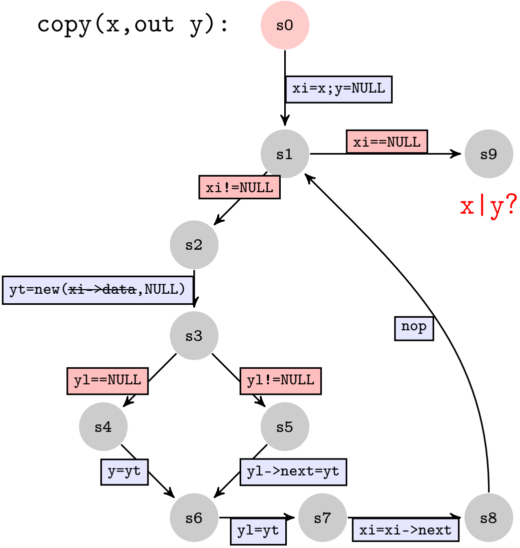
$v^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in v^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$v^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s2 extracted:

CP	$v^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y|yl yt	
s2	x xi|y|yl yt	
s3	x xi|y|yl|yt	✓
s4	$\bot$	
s5	$\bot$	
s6	$\bot$	
s7	$\bot$	
s8	$\bot$	
s9	x xi|y|yl yt	✓

$$\overline{\text{post}}^\sharp_{\texttt{yt=new...}}(v^\sharp) = \text{Extract}(\texttt{yt}, v^\sharp)$$

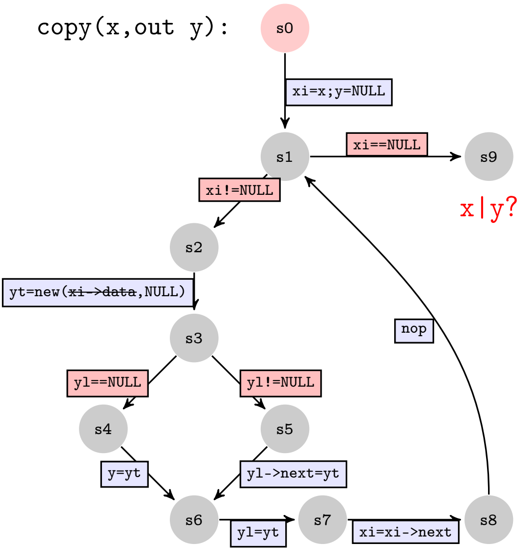


$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s3 extracted:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y|yl yt	
s2	x xi|y|yl yt	
s3	x xi|y|yl|yt	
s4	x xi|y|yl|yt	✓
s5	x xi|y|yl|yt	✓
s6	$\bot$	
s7	$\bot$	
s8	$\bot$	
s9	x xi|y|yl yt	✓



$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

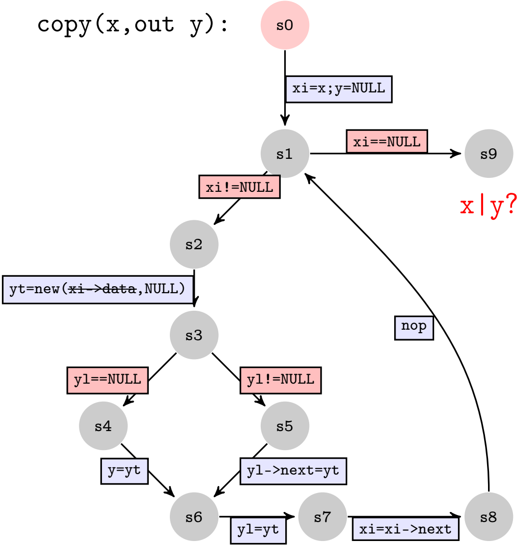
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s4 extracted:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y|yl yt	
s2	x xi|y|yl yt	
s3	x xi|y|yl|yt	
s4	x xi|y|yl|yt	
s5	x xi|y|yl|yt	✓
s6	x xi|y yt|yl	✓
s7	$\perp$	
s8	$\perp$	
s9	x xi|y|yl yt	✓

$$\overline{\text{post}}^\sharp_{\texttt{y=yt}}(\nu^\sharp) = \text{Extract}(y, \nu^\sharp) \sqcup^\sharp \{y, yt\}$$

copy(x,out y):



$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

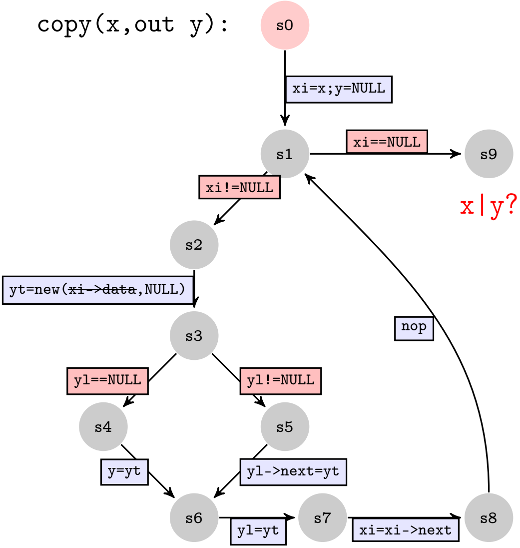
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s5 extracted:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y|yl yt	
s2	x xi|y|yl yt	
s3	x xi|y|yl|yt	
s4	x xi|y|yl|yt	
s5	x xi|y|yl|yt	
s6	x xi|y yt yl	✓
s7	$\perp$	
s8	$\perp$	
s9	x xi|y|yl yt	✓

$$\overline{\text{post}}^\sharp_{\texttt{yl->next=yt}}(\nu^\sharp) = \nu^\sharp \sqcup^\sharp \{\texttt{yl}, \texttt{yt}\}$$

Example: Analysis of Heap Separation [CC'77]



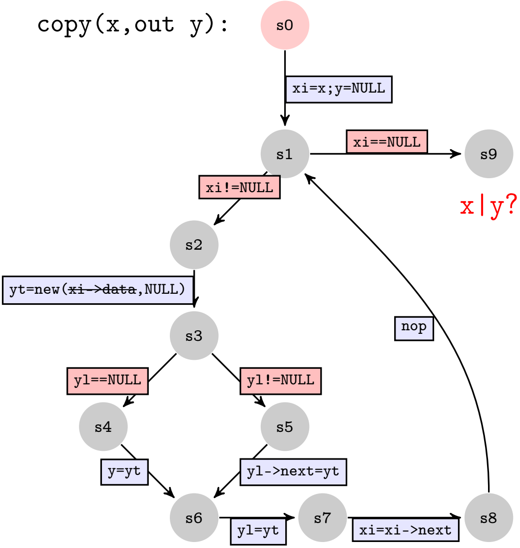
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s6 extracted:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y|yl yt	
s2	x xi|y|yl yt	
s3	x xi|y|yl|yt	
s4	x xi|y|yl|yt	
s5	x xi|y|yl|yt	
s6	x xi|y yt yl	
s7	x xi|y yt yl	✓
s8	$\perp$	
s9	x xi|y|yl yt	✓

`copy(x,out y):`

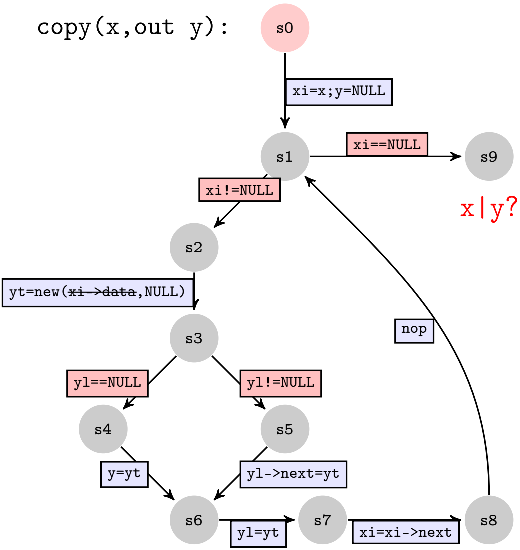


$v^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in v^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$v^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s7 extracted:

CP	$v^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y|yl yt	
s2	x xi|y|yl yt	
s3	x xi|y|yl|yt	
s4	x xi|y|yl|yt	
s5	x xi|y|yl|yt	
s6	x xi|y yt yl	
s7	x xi|y yt yl	
s8	x xi|y yt yl	✓
s9	x xi|y|yl yt	✓

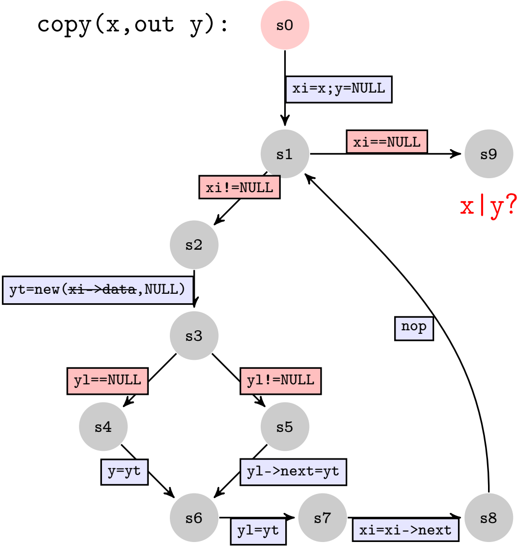


$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s8 extracted:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y yt yl	✓
s2	x xi|y|yl yt	
s3	x xi|y|yl|yt	
s4	x xi|y|yl|yt	
s5	x xi|y|yl|yt	
s6	x xi|y yt yl	
s7	x xi|y yt yl	
s8	x xi|y yt yl	
s9	x xi|y|yl yt	✓

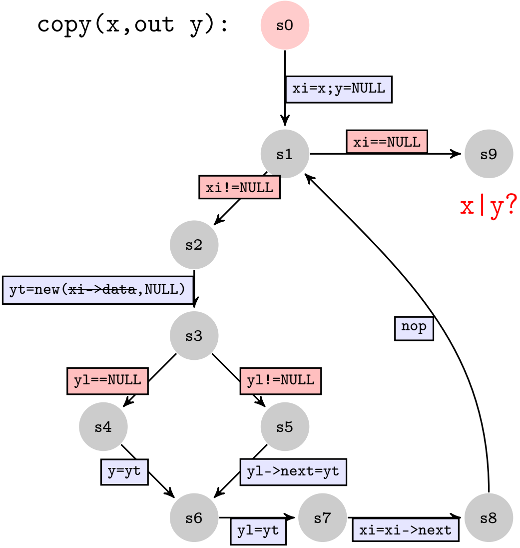


$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp .\ p \subseteq q$

$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s1 extracted:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y yt yl	
s2	x xi|y yt yl	✓
s3	x xi|y|yl|yt	
s4	x xi|y|yl|yt	
s5	x xi|y|yl|yt	
s6	x xi|y yt yl	
s7	x xi|y yt yl	
s8	x xi|y yt yl	
s9	x xi|y|yl yt	✓

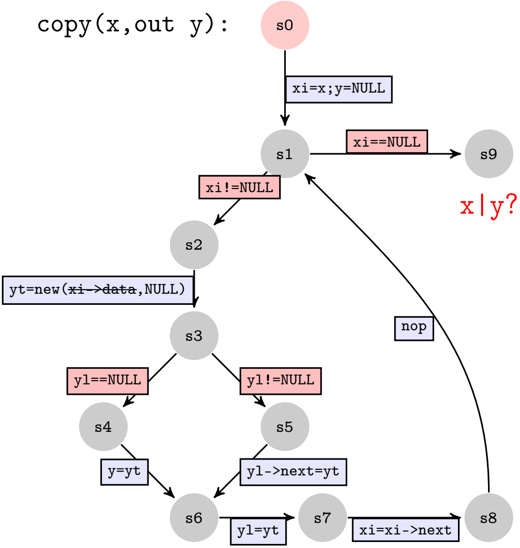


$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

… and a 2nd tour:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y yt yl	
s2	x xi|y yt yl	
s3	x xi|y yl|yt	
s4	x xi|y yl|yt	
s5	x xi|y yl|yt	
s6	x xi|y yt yl	
s7	x xi|y yt yl	
s8	x xi|y yt yl	
s9	x xi|y yl yt	✓



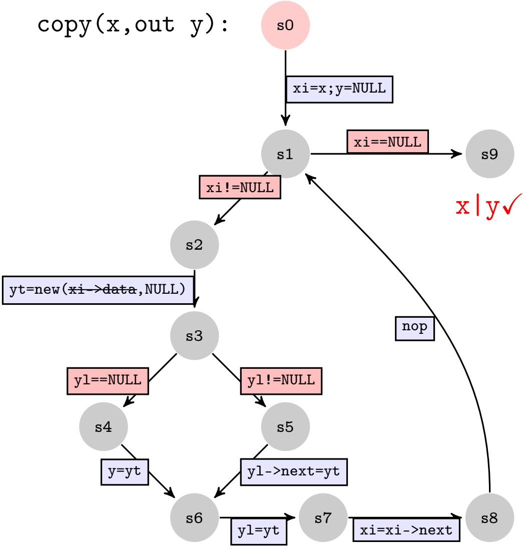
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s9 extracted:

CP	$\nu^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y yt yl	
s2	x xi|y yt yl	
s3	x xi|y yl|yt	
s4	x xi|y yl|yt	
s5	x xi|y yl|yt	
s6	x xi|y yt yl	
s7	x xi|y yt yl	
s8	x xi|y yt yl	
s9	x xi|y yl yt	

copy(x,out y):



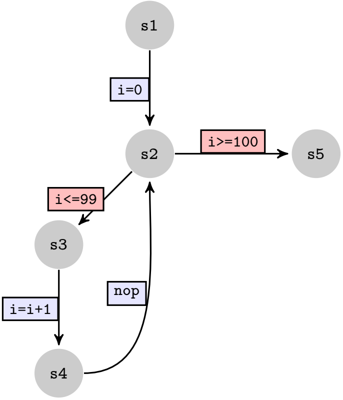
$v^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in v^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$v^\sharp \sqcup^\sharp u^\sharp$ based on union-find

CP	$v^\sharp$	W
s0	x y|xi yl yt	
s1	x xi|y yt yl	
s2	x xi|y yt yl	
s3	x xi|y yl|yt	
s4	x xi|y yl|yt	
s5	x xi|y yl|yt	
s6	x xi|y yt yl	
s7	x xi|y yt yl	
s8	x xi|y yt yl	
s9	x xi|y yl yt	

Recall:

Termination not guaranteed because $(\mathbb{Int}, \subseteq)$ does not satisfy a.c.c.!



$(\mathbb{Int}, \subseteq)$

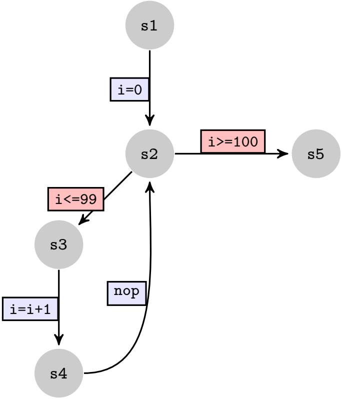
Initially:

CP	i	W
s1	$\top$	✓
s2	$\bot$	
s3	$\bot$	
s4	$\bot$	
s5	$\bot$	

Example: Numeric Analysis with Intervals

Recall:

Termination not guaranteed because $(\mathrm{Int}, \subseteq)$ does not satisfy a.c.c.!



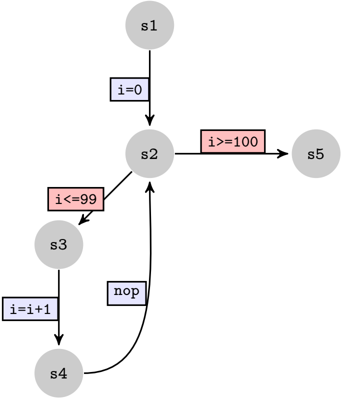
$(\mathrm{Int}, \subseteq)$

s1 extracted:

CP	i	W
s1	$\top$	
s2	(0,0)	✓
s3	$\bot$	
s4	$\bot$	
s5	$\bot$	

Recall:

Termination not guaranteed because (Int, ⊆) does not satisfy a.c.c.!



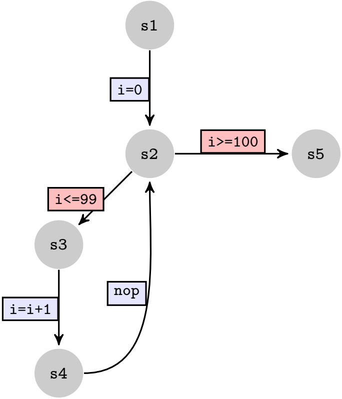
(Int, ⊆)

s2 extracted:

CP	i	W
s1	⊤	
s2	(0,0)	
s3	(0,0)	✓
s4	⊥	
s5	⊥	

Recall:

Termination not guaranteed because $(\text{Int}, \subseteq)$ does not satisfy a.c.c.!



$(\text{Int}, \subseteq)$

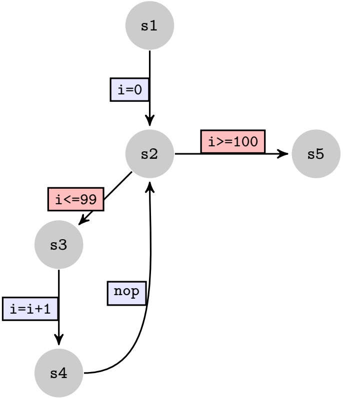
s3 extracted:

CP	i	W
s1	$\top$	
s2	(0,0)	
s3	(0,0)	
s4	(1,1)	✓
s5	$\bot$	

Example: Numeric Analysis with Intervals

Recall:

Termination not guaranteed because $(\mathrm{Int}, \subseteq)$ does not satisfy a.c.c.!



$(\mathrm{Int}, \subseteq)$

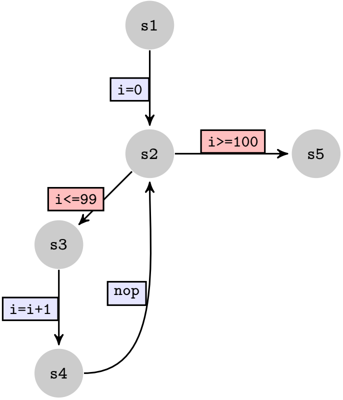
s4 extracted:

CP	i	W
s1	$\top$	
s2	(0,1)	✓
s3	(0,0)	
s4	(1,1)	
s5	$\bot$	

Recall: $(0,0) \sqcup^{\sharp} (1,1) = (0,1)$.

Recall:

Termination not guaranteed because $(\text{Int}, \subseteq)$ does not satisfy a.c.c.!



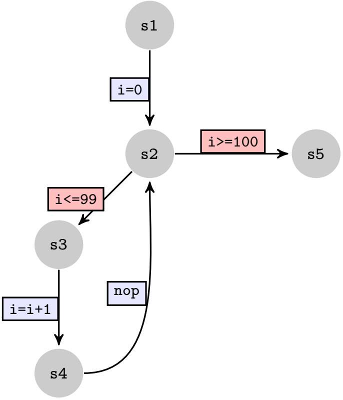
$(\text{Int}, \subseteq)$

s2 extracted:

CP	i	W
s1	$\top$	
s2	(0,1)	
s3	(0,1)	✓
s4	(1,1)	
s5	$\perp$	

Recall:

Termination not guaranteed because $(\text{Int}, \subseteq)$ does not satisfy a.c.c.!



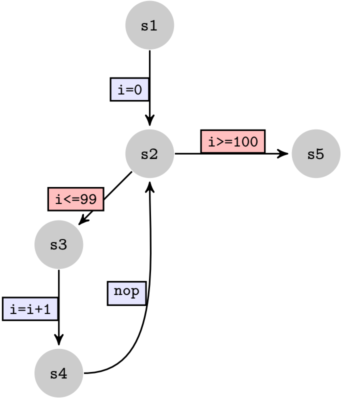
$(\text{Int}, \subseteq)$

s3 extracted:

CP	i	W
s1	$\top$	
s2	(0,1)	
s3	(0,1)	
s4	(1,2)	✓
s5	$\bot$	

Recall:

Termination not guaranteed because $(\mathrm{Int}, \subseteq)$ does not satisfy a.c.c.!



$(\mathrm{Int}, \subseteq)$

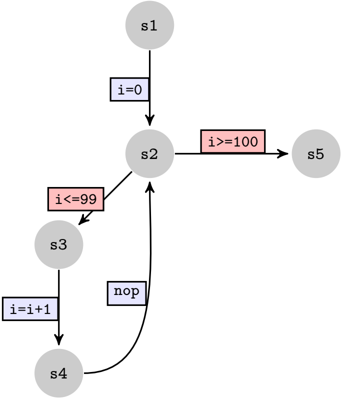
s4 extracted:

CP	i	W
s1	$\top$	
s2	(0,2)	✓
s3	(0,1)	
s4	(1,2)	
s5	$\bot$	

Example: Numeric Analysis with Intervals

Recall:

Termination not guaranteed because $(\mathrm{Int}, \subseteq)$ does not satisfy a.c.c.!



$(\mathrm{Int}, \subseteq)$

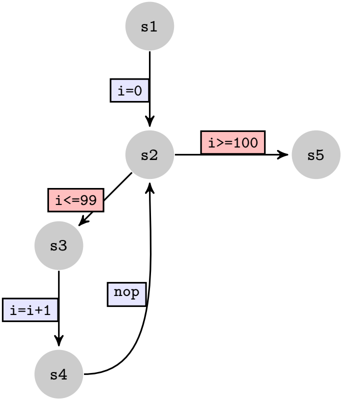
… 98 it. later ☹

CP	i	W
s1	$\top$	
s2	(0,100)	✓
s3	(0,99)	
s4	(1,100)	
s5	$\bot$	

Example: Numeric Analysis with Intervals

Recall:

Termination not guaranteed because $(\text{Int}, \subseteq)$ does not satisfy a.c.c.!



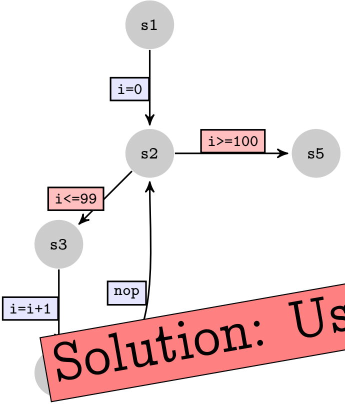
$(\text{Int}, \subseteq)$

s1 extracted:

CP	i	W
s1	$\top$	
s2	(0,100)	
s3	(0,99)	
s4	(1,100)	
s5	(100,100)	✓

Recall:

Termination not guaranteed because $(\text{Int}, \subseteq)$ does not satisfy a.c.c.!



$(\text{Int}, \subseteq)$

CP	i	W
s1	⊤	
s4	(1,100)	
s5	(100,100)	

Solution: Use widening! [Cousot&Cousot,79]

Widening for Intervals

Definition

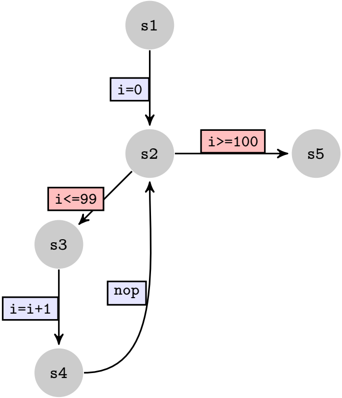
Given a complete lattice $(L, \sqsubseteq)$, a widening operator $\nabla : L \times L \to L$ satisfies

- $\forall x, y \in L.\ x \sqcup y \sqsubseteq x \nabla y$
- for all sequences $(l_n)_{n \in \mathbb{N}}$, the (ascending) chain $(w_n)_{n \in \mathbb{N}}$
  $w_0 = l_0, \quad w_{i+1} = w_i \nabla l_{i+1} \quad \text{for } i > 0$
  stabilises eventually.

Widening for $(Int, \subseteq)$: $(\ell_0, u_0) \nabla (\ell_1, u_1) = (\ell_2, u_2)$ where

$$\ell_2 = \begin{cases} \ell_0 & \text{if } \ell_0 \leqslant \ell_1 \\ -\infty & \text{otherwise} \end{cases} \quad u_2 = \begin{cases} u_0 & \text{if } u_0 \geqslant u_1 \\ +\infty & \text{otherwise} \end{cases}$$

Example of widening chain: $\bot \subseteq (0,0) \subseteq (0,0) \nabla (0,1) = (0, +\infty)$



$(\text{Int}, \subseteq)$

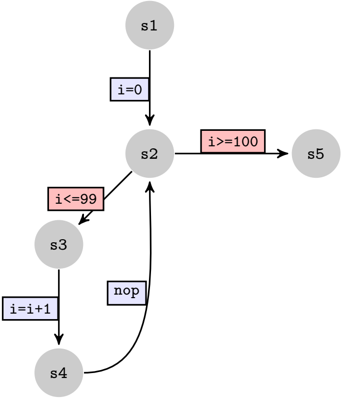
With $\sqcup$, in 100 it.:

CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

Initialy:

CP	i	W
s1	$\top$	✓
s2	$\bot$	
s3	$\bot$	
s4	$\bot$	
s5	$\bot$	

Solution (naive): Apply widening instead $\sqcup$ in the workset algorithm.



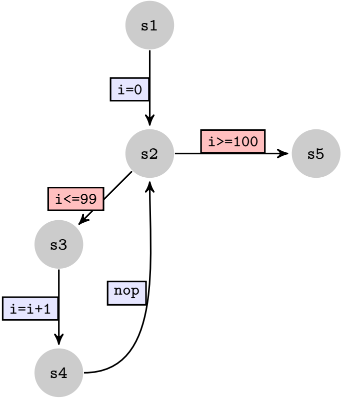
$(\text{Int}, \subseteq)$

With $\sqcup$, in 100 it.:

CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

s1 extracted:

CP	i	W
s1	$\top$	
s2	(0,0)	✓
s3	$\bot$	
s4	$\bot$	
s5	$\bot$	



$(\text{Int}, \subseteq)$

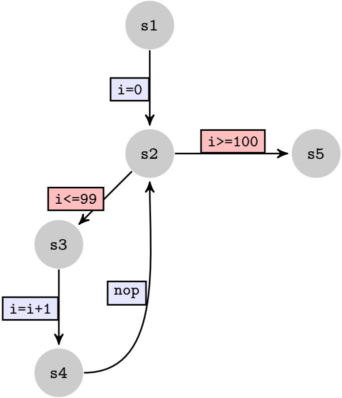
With $\sqcup$, in 100 it.:

CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

s2 extracted:

CP	i	W
s1	$\top$	
s2	(0,0)	
s3	(0,0)	✓
s4	$\bot$	
s5	$\bot$	

Example: Numeric Analysis with Intervals



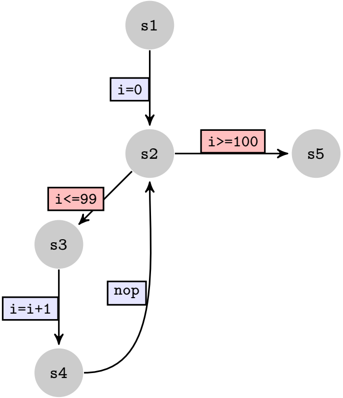
$(\text{Int}, \subseteq)$

With $\sqcup$, in 100 it.:

CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

s3 extracted:

CP	i	W
s1	$\top$	
s2	(0,0)	
s3	(0,0)	
s4	(1,1)	✓
s5	$\bot$	



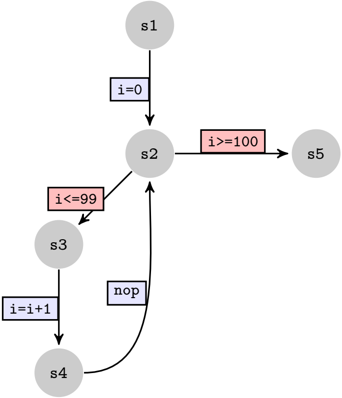
(Int, ⊆)

With $\sqcup$, in 100 it.:

CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

s4 extracted:

CP	i	W
s1	$\top$	
s2	$(0,+\infty)$	✓
s3	(0,0)	
s4	(1,1)	
s5	$\bot$	



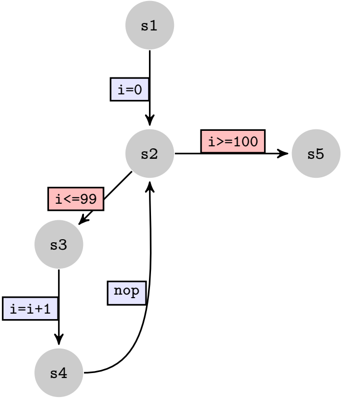
$(\text{Int}, \subseteq)$

With $\sqcup$, in 100 it.:

CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

s2 extracted:

CP	i	W
s1	$\top$	
s2	$(0,+\infty)$	
s3	$(0,+\infty)$	✓
s4	(1,1)	
s5	$(100,+\infty)$	✓



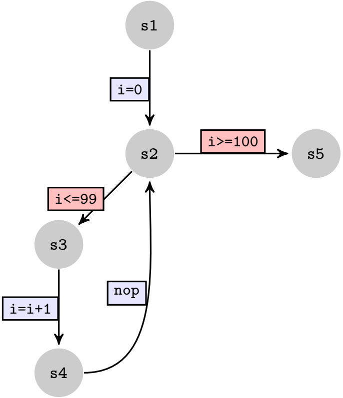
$(\text{Int}, \subseteq)$

With $\sqcup$, in 100 it.:

CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

s3 extracted:

CP	i	W
s1	$\top$	
s2	$(0,+\infty)$	
s3	$(0,+\infty)$	
s4	$(1,+\infty)$	✓
s5	$(100,+\infty)$	



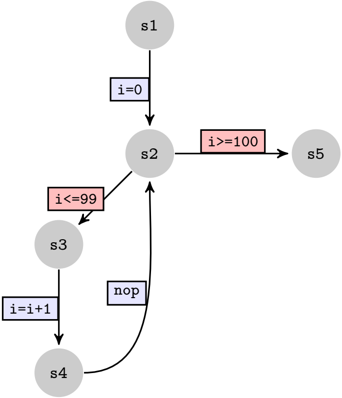
$(\text{Int}, \subseteq)$

With $\sqcup$, in 100 it.:

CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

s4 extracted:

CP	i	W
s1	$\top$	
s2	$(0,+\infty)$	✓
s3	$(0,+\infty)$	
s4	$(1,+\infty)$	
s5	$(100,+\infty)$	



$(\text{Int}, \subseteq)$

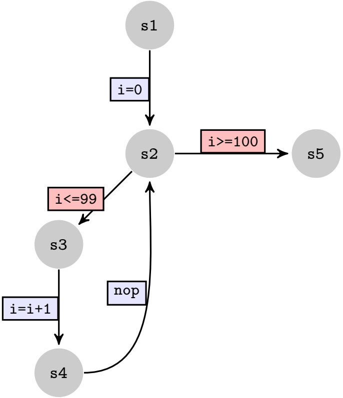
With $\sqcup$, in 100 it.:

CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

With ∇, in 2 it. ☹

CP	i	W
s1	$\top$	
s2	$(0,+\infty)$	
s3	$(0,+\infty)$	
s4	$(1,+\infty)$	
s5	$(100,+\infty)$	

Example: Numeric Analysis with Intervals



$(\text{Int}, \subseteq)$

With $\sqcup$, in 100 it.:

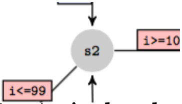
CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

With ∇, in 2 it.

CP	i	W
s1	$\top$	
s2	$(0,+\infty)$	
s3	$(0,+\infty)$	
s4	$(1,+\infty)$	
s5	$(100,+\infty)$	

Conclusion: Widening speeds up termination with some precision loss!

Widening on "loop separators": *i.e.*, one point by ICFG cycle, *e.g.*



Widening "up to": take into account constants in boolean conditions

⟶[Jeannet *et al*,11]

Delayed widening: apply ∇ after $k \geqslant 2$ iterations

Narrowing: iterate again from the result obtained by widening

Accelerate: introduce iteration variables for loops

⟶[Gonnord & Halbwachs,06]

...

Intervals: $\bigwedge_{x \in V} \pm x \leqslant c$ [Cousot&Cousot,77] $O(n)$

Differences: $\bigwedge_{x,y \in V} x - y \leqslant c$

Octagons: $\bigwedge_{x,y \in V} \pm x \pm y \leqslant c$ [Miné'01] $O(n^3)$

Polyhedra: $\bigwedge \Sigma_{x_i \in V} c_i x_i \leqslant c$ [Cousot& Halbwachs,78] $O(2^n)$

More precision leads to higher costs (in $n = |V|$) of lattice operations.

Numerical Abstract Lattices

Intervals: $\bigwedge_{x \in V} \pm x \leqslant c$ [Cousot&Cousot,77] $O(n)$

Differences: $\bigwedge_{x,y \in V} x - y \leqslant c$ $O(n^3)$

Octagons: $\bigwedge_{x,y \in V} \pm x \pm y \leqslant c$ [Miné'01] $O(n^3)$

Polyhedra: $\bigwedge \Sigma_{x_i \in V} c_i x_i \leqslant c$ [Cousot& Halbwachs,78] $O(2^n)$

More precision leads to higher costs (in $n = |V|$) of lattice operations.

Some Free Implementations

Apron: numerical domains (C, C++, and Ocaml), normalised interface, wrapper PPL, tools for arithmetical expressions
⟶ `apron.cri.ensmp.fr`, [Jeannet& Miné,09]

Fixpoint: engine for computing `lfp` from ICFG and lattice
⟶ `pop-art.inrialpes.fr/people/bjeannet`

Interproc: on-line analyser for a simple language
⟶ `pop-art.inrialpes.fr/people/bjeannet`

Frama-C: platform for verifying and analysing C programs
⟶ `frama-c.com`

PPL: numerical domains (C++)
⟶ `bugseng.com/products/ppl/`, [Bagnara *et al*,08]

General Interface for Abstract Domains

```
module AbsDom is
  type D;
  operations
   copy : D -> D
   size : D -> int

   minimize : D -> unit
   hash     : D -> unit

   bot, top : int -> D
   of_itv   : itv -> D

   is_bot, is_top : D -> bool
   is_leq, is_eq  : D -> D -> bool
   sat_itv : D -> itv -> bool

   itv_of_var : D -> V -> itv
   to_itv     : D -> itv

   meet, join : D -> D -> D

   post_bexp, pre_bexp : D -> bexpr -> D
   post_astmt, pre_astmt : D -> var -> expr -> D

   widen      : D -> D -> D
   widen_upto : D -> D -> expr -> D

   add_dim, prj_dim : D -> var list -> D

   print :  D -> ostream -> unit
end module
```

≈ Apron [Jeannet& Miné,09]

Some References

- F. Nielson, H. R. Nielson, and C. Hankin,
  Principles of Program Analysis.
  Springer, 1999

- P. Cousot and R. Cousot,
  Systematic design of program analysis frameworks.
  In Proc. 6th ACM Symp. Principles of Programming Languages, San Antonio, TX, USA, pages 269-282. ACM Press, 1979

- VTSA Talks: G. Sutre – 2008, D. Monniaux – 2012, M. Müller-Olm – 2010

Outline

Programs with Lists and Data

— Invariant Synthesis by Abstract Interpretation —

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

CAV'10, PLDI'11

Motivation: Example

```
struct list { int data; list* next; };
/* @assume: n≥2 */
list* fibList(int n) {
  int k = 1; list* lf = newList(1,NULL);
  lf = pushList(lf,1);
  while(k<n) {
    lf = pushList(lf,lf->data+lf->next->dt);
    k = k+1;
  }
  return lf;
}
```

Motivation: Example

```
struct list { int data; list* next; };
/* @assume:  n≥2 */
list* fibList(int n) {
  int k = 1; list* lf = newList(1,NULL);
  lf = pushList(lf,1);
  while(k<n) {
    lf = pushList(lf,lf->data+lf->next->dt);
    k = k+1;
  }
  return lf;
}
```

Motivation: Example

```
struct list { int data; list* next; };
/* @assume:  n≥2 */
list* fibList(int n) {
  int k = 1; list* lf = newList(1,NULL);
  lf = pushList(lf,1);
  while(k<n) {
    lf = pushList(lf,lf->data+lf->next->dt);
    k = k+1;
  }
  return lf;
} /* @assert:lseg(lf,NULL,n) ∧...∧ ∀i.0≤i<n-2 ⟹ l̂f[i]=l̂f[i+1]+l̂f[i+2] */
```

Motivation: Example

```
struct list { int data; list* next; };
/* @assume:  n≥2 */
list* fibList(int n) {
  int k = 1; list* lf = newList(1,NULL);
  lf = pushList(lf,1);
  while(k<n) {
    lf = pushList(lf,lf->data+lf->next->dt);
    k = k+1;
  }
  return lf;
} /* @assert:lseg(lf,NULL,n) ∧...∧ ∀i.0≤i<n-2 ⟹ lf^[i]=lf^[i+1]+lf^[i+2] */
```

CELIA tool:

- ✓ prove the program and the correct access to the memory
- ✓ infer automatically the annotations given

Motivation: Example

```
struct list { int data; list* next; };
/* @assume:  n≥2 */
list* fibList(int n) {
  int k = 1; list* lf = newList(1,NULL);
  lf = pushList(lf,1);
  while(k<n) {
    lf = pushList(lf,lf->data+lf->next->dt);
    k = k+1;
  }
  return lf;
} /* @assert:lseg(lf,NULL,n) ∧...∧ ∀i.0≤i<n-2 ⟹ l̂f[i]=l̂f[i+1]+l̂f[i+2] */
```

Other tools:

- TVLA [Sagiv *et al*, 07, 11]
  ⟶ fixed set of data constraints, no size constraints
- SLAyer, Infer [Berdine, Cook & Ishtiaq, 11]
  ⟶ no data or size constraints

```
/* @assume:  n≥1 */
int fib(int n) {
  int fp = 1; int fl = 1; int i = 1;
  /* n≥i≥1 ∧ 1≤i,fp≤fl */
  while(i<n) {
    /* 1≤i<n ∧ i,fp≤fl */
    int t = fp+fl;
    /* 1≤i<n ∧ 1≤i,fp≤fl≤t */
    fp = fl;/* 1≤i<n ∧ 1,i≤fp=fl≤t */
    fl = t; /* 1≤i<n ∧ 1≤i,fp≤fl=t */
    i = i+1;/* 1≤i≤n ∧ 1≤i,fp≤fl*/
  }
  /* 1≤n=i ∧ 1≤i,fp≤fl */
  return fl;
} /* @assert:  fib(n)≥n≥1 */
```

$(L^\sharp, \sqsubseteq^\sharp)$

Octogonal constraints:
$\bigwedge \pm x \pm y \leqslant c$

Tool

✓ Interproc & Apron

Recall the formal model of IMPR:

Control points	$\mathbf{CP} \ni \ell, \ell'$
	$\ni$ *start*$_{\mathrm{P}}$, *end*$_{\mathrm{P}}$ for each procedure $\mathrm{P} \in \mathbf{P}$
Stack	$\mathtt{Stacks} \triangleq [\left(\mathbf{CP} \times \mathbf{P} \times (\mathbf{DV} \nrightarrow \mathbb{D} \;\cup\; \mathbf{RV} \nrightarrow \mathbb{L})\right)^{*}] \ni \mathtt{S}$
Heap	$\mathtt{Heaps} \triangleq [(\mathbb{L} \times \mathbf{FS}) \nrightarrow (\mathbb{D} \;\cup\; \mathbb{L})] \ni \mathtt{H}$
Memory	$\mathtt{Mem} \triangleq \mathtt{Stacks} \times \mathtt{Heaps} \ni \mathtt{m}$
Configurations	$\mathtt{Config} \triangleq \mathbf{CP} \times (\mathtt{Mem} \cup \{\mathtt{merr}\}) \ni \mathtt{C}$

Consider IMPR without recursive procedures, then:

Control points	$\mathbf{CP} \ni \ell, \ell'$
Stack	$\mathtt{Stacks} \triangleq \mathbf{DV} \nrightarrow \mathbb{D} \ \cup \ \mathbf{RV} \nrightarrow \mathbb{L} \ni \mathtt{S}$
Heap	$\mathtt{Heaps} \triangleq [(\mathbb{L} \times \mathbf{FS}) \nrightarrow (\mathbb{D} \ \cup \ \mathbb{L})] \ni \mathtt{H}$
Memory	$\mathtt{Mem} \triangleq \mathtt{Stacks} \times \mathtt{Heaps} \ni \mathtt{m}$
Configurations	$\mathtt{Config} \triangleq \mathbf{CP} \times (\mathtt{Mem} \cup \{\mathtt{merr}\}) \ni \mathtt{C}$

Mem is represented by deterministic labeled graphs, *i.e.*:

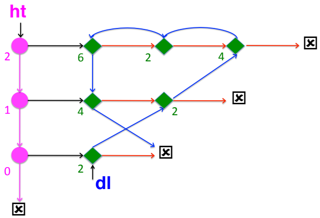
Definition

A heap graph is a tuple $\langle \mathcal{N}, \mathcal{E}, \mathcal{L} \rangle$ where $\mathcal{N} = \mathbb{L}$ is a set of (typed) graph *nodes*, $\mathcal{E} : \mathcal{N} \times \mathbf{RF} \nrightarrow \mathcal{N}$ is a set of (reference field) *labeled edges*, and $\mathcal{L} : (\mathcal{N} \to \mathcal{P}(\mathbf{RV})) \ \cup \ (\mathcal{N} \times \mathbf{DF} \nrightarrow \mathbb{D})$ is *node labeling* function.

Consider IMPR without recursive procedures, then:

Control points	$\mathbf{CP} \ni \ell, \ell'$
Stack	$\mathtt{Stacks} \triangleq \mathbf{DV} \nrightarrow \mathbb{D} \ \cup \ \mathbf{RV} \nrightarrow \mathbb{L} \ni \mathtt{S}$
Heap	$\mathtt{Heaps} \triangleq [(\mathbb{L} \times \mathbf{FS}) \nrightarrow (\mathbb{D} \ \cup \ \mathbb{L})] \ni \mathtt{H}$
Memory	$\mathtt{Mem} \triangleq \mathtt{Stacks} \times \mathtt{Heaps} \ni \mathtt{m}$
Configurations	$\mathtt{Config} \triangleq \mathbf{CP} \times (\mathtt{Mem} \cup \{\mathtt{merr}\}) \ni \mathtt{C}$

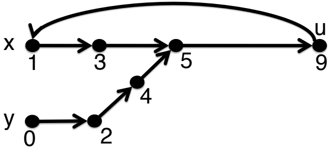
Mem is represented by deterministic labeled graphs, *e.g.*:



Heap Graphs for Lists of Integers (1/2)

For programs with lists of integers, *i.e.*,
$\mathbf{RT} = \{\texttt{list}\}$ with $\mathbf{RF} = \{\texttt{next}\}$ and $\mathbf{DF} = \{\texttt{data}\}$,
the heap graph model is a labeled functional graph, *i.e.*, $\langle \mathcal{N}, \mathcal{E}, \mathcal{L} \rangle$ where:

- $\mathcal{N} = \mathbb{L}$
- $\mathcal{E} : \mathcal{N} \setminus \{\boxtimes\} \to \mathcal{N}$
- $\mathcal{L} : \big(\mathcal{N} \to \mathcal{P}(\mathbf{RV})\big) \cup \big(\mathcal{N} \setminus \{\boxtimes\} \to \mathbb{D}\rangle$.



Particular case

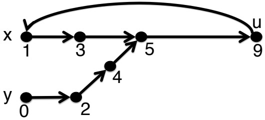
A heap graph without data labels on nodes is called pure heap graph.

Heap Graphs for Lists of Integers (2/2)

Furthermore, a precise abstraction of these heap graphs keeps only cut nodes and a set of integer words.

Definition

A cut node is a node labelled by a variable or having at least two incoming edges. A node which not a cut node is called anonymous.

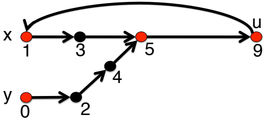


Heap Graphs for Lists of Integers (2/2)

Furthermore, a precise abstraction of these heap graphs keeps only cut nodes and a set of integer words.

Definition

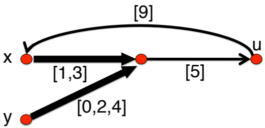
A cut node is a node labelled by a variable or having at least two incoming edges. A node which not a cut node is called anonymous.



Furthermore, a precise abstraction of these heap graphs keeps only cut nodes and a set of integer words.

Definition

A cut node is a node labelled by a variable or having at least two incoming edges. A node which not a cut node is called anonymous.

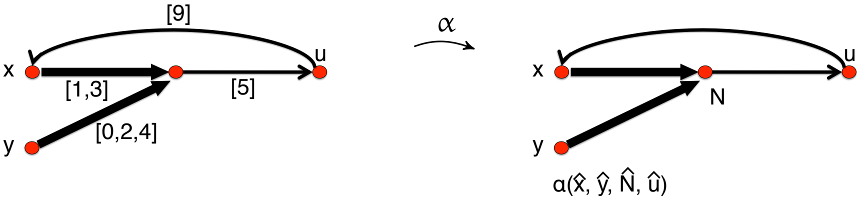


Property

The number of pure heap graphs with only cut nodes is finite (for finite **RV**). $\longrightarrow$ Idea: reverse edges!

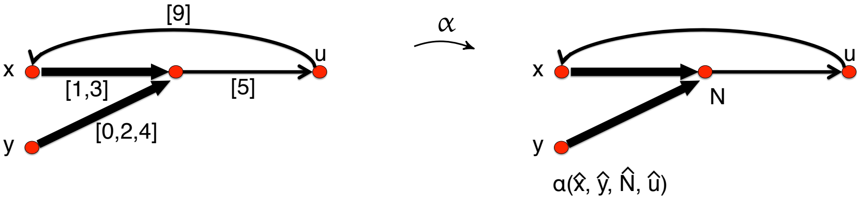
Abstract Heap Graphs: Idea

The only source of infinity are the integer words $\longrightarrow$ abstract them!



where α on integer words may be, *e.g.*:

The only source of infinity are the integer words $\longrightarrow$ abstract them!

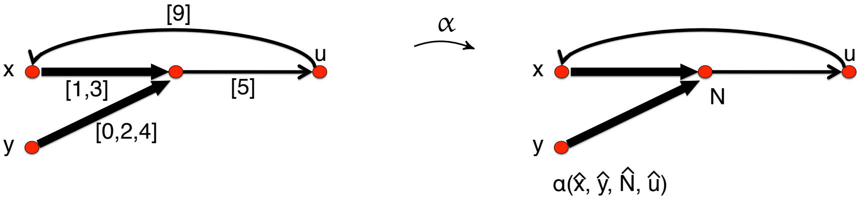


where α on integer words may be, *e.g.*:

Universal constraints abstraction

$$
\begin{aligned}
\alpha(\widehat{x}, \widehat{y}, \widehat{N}, \widehat{u}) \quad = \quad & |\widehat{u}| = 1 \ \wedge \ |\widehat{N}| = 1 \ \wedge \ |\widehat{x}| \leqslant |\widehat{y}| \ \wedge \\
& \widehat{u}[0] \geqslant \widehat{N}[0] \ \wedge \ \widehat{x}[0] \geqslant 1 \ \wedge \ \widehat{y}[0]\%2 = 0 \ \wedge \\
& \forall i.\ 0 < i < |\widehat{x}| \Longrightarrow \widehat{x}[i] \geqslant 1 \ \wedge \\
& \forall i.\ 0 < i < |\widehat{y}| \Longrightarrow \widehat{y}[i]\%2 = 0
\end{aligned}
$$

The only source of infinity are the integer words $\longrightarrow$ abstract them!

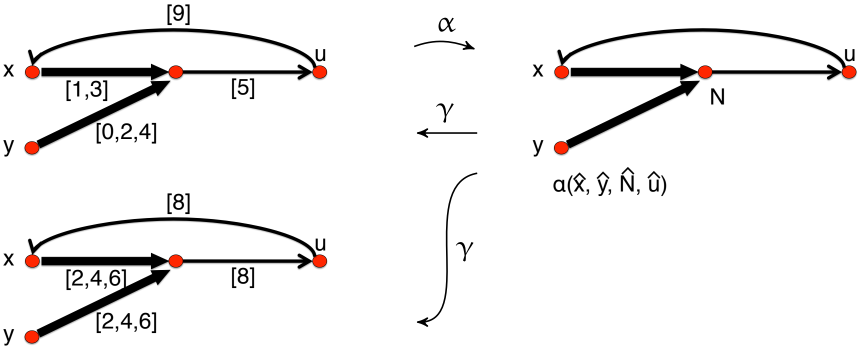


where α on integer words may be, *e.g.*:

Sum constraints abstraction

$$\alpha(\widehat{x}, \widehat{y}, \widehat{N}, \widehat{u}) \quad = \quad \Sigma(\widehat{N}) - \Sigma(\widehat{x}) \geq 1 \ \wedge \ \Sigma(\widehat{y}) \leqslant \Sigma(\widehat{u}) \ \wedge$$
$$|\widehat{x}| \leqslant |\widehat{y}| \ \wedge \ |\widehat{x}| \leqslant \Sigma(\widehat{N})$$

γ is defined using models of integer words constraints.



$$\alpha(\widehat{x}, \widehat{y}, \widehat{N}, \widehat{u}) \quad = \quad \widehat{u}[0] \geqslant \widehat{N}[0] \ \wedge \ |\widehat{x}| \leqslant |\widehat{y}| \ \wedge \ldots \ \wedge$$
$$\forall i.\ 0 < i < |\widehat{x}| \Longrightarrow \widehat{x}[i] \geqslant 1 \ \wedge$$
$$\forall i.\ 0 < i < |\widehat{y}| \Longrightarrow \widehat{y}[i]\%2 = 0$$

Definition

Given $\mathcal{A}_{\mathbb{W}}$ an abstract domain on words, the domain of abstract heaps is:

$$\mathcal{A}_{\mathbb{H}}(\mathcal{A}_{\mathbb{W}}) = (\mathrm{L}^{\mathbb{H}}, \sqsubseteq^{\mathbb{H}}, \sqcup^{\mathbb{H}}, \sqcap^{\mathbb{H}}, \top^{\mathbb{H}}, \bot^{\mathbb{H}})$$

where

$$\forall (\mathrm{G}, W) \in \mathrm{L}^{\mathbb{H}} \implies \mathrm{G} \in \text{pure heap graph without anonymous}, \ W \in Ł^{\mathbb{W}}$$

$$(\mathrm{G}_1, W_1) \sqsubseteq^{\mathbb{H}} (\mathrm{G}_2, W_2) \text{ iff } \mathrm{G}_1 \approx_{\mathrm{iso}} \mathrm{G}_2 \text{ and } W_1 \sqsubseteq^{\mathbb{W}} W_2$$

$$(\mathrm{G}_1, W_1) \sqcup^{\mathbb{H}} (\mathrm{G}_2, W_2) = \begin{cases} (\mathrm{G}_1, W_1 \sqcup^{\mathbb{W}} W_2) & \text{if } \mathrm{G}_1 \approx_{\mathrm{iso}} \mathrm{G}_2 \\ \top^{\mathbb{H}} & \text{otherwise} \end{cases}$$

and $\sqcap^{\mathbb{H}}$ defined similarly. $\top^{\mathbb{H}}$ and $\bot^{\mathbb{H}}$ are special values such that

$$\forall v^{\mathbb{H}} \in \mathrm{L}^{\mathbb{H}}. \ \bot^{\mathbb{H}} \sqsubseteq^{\mathbb{H}} v^{\mathbb{H}} \sqsubseteq^{\mathbb{H}} \top^{\mathbb{H}}$$

Abstract Heap Set Domain $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{W}})$

Definition

The abstract heap set domain $\mathcal{A}_{\mathbb{HS}}(\mathcal{A}_{\mathbb{W}})$ is the power-set domain of $\mathcal{A}_{\mathbb{H}}(\mathcal{A}_{\mathbb{W}})$ such that for any $\nu^{\mathbb{HS}} \in \mathrm{L}^{\mathbb{HS}}$, $\nu^{\mathbb{HS}}$ does not contain two abstract heaps with isomorphic pure heap graphs.

Main logics for specifying heap graphs:

- FO(G)+TC [Immerman *et al*, 87, 04]
  $\longrightarrow$ decidable fragment LRP [Yorsh *et al*, 06]
  $\longrightarrow$ decidable fragment CSL [Bouajjani *et al*, 09]

- Calculus of reachability [Nelson, 93]
  $\longrightarrow$ conjunction of reachability predicates $p \xrightarrow[x]{nxt} q$ is decidable
  $\longrightarrow$ extension for well-founded lists [Lahiri & Quadeer, 06]

- Separation Logic [Reynolds *et al*, 99]
  $\longrightarrow$ decidable if no quantification [Calcagno, Yang & O'Hearn, 01]

Assertions: $\exists\overrightarrow{X}.\ \Pi \wedge \Sigma$ where

E, F	$::=$	$x \mid X$	$x \in \mathbf{RV}$, X logical variable
Π	$::=$	$E = F \mid E \neq F \mid \Pi \wedge \Pi$	pure formulas
Σ	$::=$	$emp \mid E \mapsto \{(f_i, F_i)\}_i \mid \Sigma * \Sigma$	spatial formulas

No Aliasing: $(S, H) \models E \neq F$ iff $S(E) \neq S(F)$

Empty heap: $(S, H) \models emp$ iff $\mathrm{dom}(H) = \emptyset$

An allocated cell: $(S, H) \models E \mapsto \{(f_i, F_i)\}_i$
iff $\mathrm{dom}(H) = S(E)$, $H(S(E), f_i) = S(F_i)$ for any i

Separating conjunction: $(S, H) \models \Sigma_1 * \Sigma_2$ iff
$H = H_1 \cup H_2$ s.t. $\mathrm{dom}(H_1) \cap \mathrm{dom}(H_2) = \emptyset$ and
$(S, H_1) \models \Sigma_1$, $(S, H_2) \models \Sigma_2$

To specify unbounded heap graphs, use inductive predicates defined by a set of rules of the form $P(\overrightarrow{E}) \triangleq \exists \overrightarrow{X}.\ \Pi \wedge \Sigma$

$$
\begin{array}{llll}
E, F & ::= & x \mid X & x \in \mathbf{RV},\ X \text{ logical var.} \\
\Pi & ::= & E = F \mid E \neq F \mid \Pi \wedge \Pi & \text{pure formulas} \\
\Sigma & ::= & emp \mid E \mapsto \{(f_i, F_i)\}_i \mid \Sigma * \Sigma \mid P(\overrightarrow{E}) & \text{spatial formulas}
\end{array}
$$

Examples:

$$
\begin{array}{lll}
\mathtt{ls}(E, F) & \triangleq & E = F \wedge emp \\
\mathtt{ls}(E, F) & \triangleq & \exists X.\ E \neq F * E \mapsto \{(next, X)\} * \mathtt{ls}(X, F) \\
\\
\mathtt{ls}^{+}(E, F) & \triangleq & E \neq F \wedge E \mapsto \{(next, F)\} \\
\mathtt{ls}^{+}(E, F) & \triangleq & \exists X.\ E \neq F * E \mapsto \{(next, X)\} * \mathtt{ls}^{+}(X, F) \\
\\
\mathtt{nll}(E, F, B) & \triangleq & E = F \wedge emp \\
\mathtt{nll}(E, F, B) & \triangleq & \exists X, Y.\ E \neq F * E \mapsto \{(next, X), (s, Y)\} * \mathtt{ls}^{+}(Y, B) * \mathtt{nll}(X, F, B)
\end{array}
$$

The fragment of SL using only $\texttt{ls}^+$ (also true for $\texttt{ls}$) has good properties:

1. Compositional reasoning due to separation conjunction $*$

$$\frac{\{P\}\ stmt\ \{Q\}}{\{P * R\}\ stmt\ \{Q * R\}}$$

2. Satisfiability and entailment are in PTIME [Cook *et al*, 11]
3. Closure under post image of IMPR due to logic variables
4. Efficient symbolic representation: functional graphs
5. Not closed under $\neg$, not stably infinite
   $\longrightarrow$ combination with other logic theories is difficult

Logical View: Array Properties

Some decidable logics for specifying arrays of integers:

1. Quantifier free with permutation predicate [Suzuki & Jefferson, 80]

$$\mathrm{store}(a, i, v) \wedge a[j] = b[j] \quad \Longrightarrow \quad \mathrm{perm}(a, b)$$

2. Array Property Fragment [Bradley, Manna & Sipma, 06]

$$n \geqslant 0 \ \wedge \ a[\ell] \leqslant k \ \wedge \ \forall i_1, i_2.\ n \leqslant i_1 \leqslant i_2 < \ell \Longrightarrow \varphi(a[i_2], a[i_1], a[\ell])$$

3. Logic of Integer Arrays [Habermehl, Iosif & Vojnar, 08]

$$\forall \vec{i}.\ i \equiv_2 0 \ \Longrightarrow \ a[i] - a[i+1] \leqslant k$$

Any element of $L^{\mathbb{HS}}$ has a logical representation by a formula:

$$\bigvee_i \exists \overrightarrow{X}_i.\ (\varphi^i_{SL} \wedge \psi^i_{AL})$$

where:

- $\varphi^i_{SL} \in SL(\mathtt{ls}^+)$ whose Gaifman graph is a pure heap graph
- for any i, j, Gaifman graph of φ^i_{SL} and φ^j_{SL} are not isomorphic
- ψ^i_{AL} are formulas in some logic over arrays AL

Any element of $L^{\mathbb{HS}}$ has a logical representation by a formula:

$$\bigvee_i \exists \overrightarrow{X}_i.\ (\varphi^i_{SL} \wedge \psi^i_{AL})$$

where:

- $\varphi^i_{SL} \in SL(\mathtt{ls}^+)$ whose Gaifman graph is a pure heap graph
- for any i, j, Gaifman graph of φ^i_{SL} and φ^j_{SL} are not isomorphic
- ψ^i_{AL} are formulas in some logic over arrays AL

Decidability Issues

$$\forall i.\ 0 \le i < n - 2 \Longrightarrow \widehat{\mathtt{lf}}[i] = \widehat{\mathtt{lf}}[i+1] + \widehat{\mathtt{lf}}[i+2]$$

is not in a decidable array logic!
Solution: use sound but incomplete procedures for satisfiability (*i.e.*, $= \bot^{AL}$) and entailment (*i.e.*, $\sqsubseteq^{AL}$)

Domain of Array Logic for $\mathcal{A}_{\mathbb{W}}$

- “all the values in n are greater than 3”

$$\forall i.\ 0 \leq i < |\widehat{n}| \Longrightarrow \widehat{n}[i] \geqslant 3$$

- “n contains a Fibonacci sequence”

$$\forall i_1, i_2, i_3.\ 0 \leq i_1, i_2, i_3 < |\widehat{n}| \wedge i_1 <_1 i_2 <_1 i_3 \Longrightarrow \widehat{n}[i_3] = \widehat{n}[i_2] + \widehat{n}[i_1]$$

$$\forall i_1, i_2.\ 0 \leq i_1, i_2 < |\widehat{n}| \wedge i_1 < i_2 \Longrightarrow \widehat{n}[i_2] - \widehat{n}[i_1] \geqslant i_2 - i_1$$

- “lists n and m have the same content”

$$|\widehat{n}| = |\widehat{m}| \wedge \forall i, i'.\ 0 \leq i < |\widehat{n}| \wedge 0 \leq i' < |\widehat{m}| \wedge i = i' \Longrightarrow \widehat{n}[i] = \widehat{m}[i']$$

$$E(\overrightarrow{N}) \wedge \bigwedge_{g \in \mathcal{G}} \forall \overrightarrow{i}.\ g(\overrightarrow{i}, \overrightarrow{N}) \Longrightarrow U(\overrightarrow{N}, \overrightarrow{i})$$

$$\mathcal{A}_{\mathbb{U}} = (A^{\mathbb{U}}, \sqsubseteq^{\mathbb{U}}, \sqcup^{\mathbb{U}}, \sqcap^{\mathbb{U}}, \top^{\mathbb{U}}, \bot^{\mathbb{U}})$$

$$A^{\mathbb{U}} \ni E(\overrightarrow{N}) \wedge \bigwedge_{g \in \mathcal{G}} \forall \overrightarrow{i}.\ g(\overrightarrow{i}, \overrightarrow{N}) \Longrightarrow U(\overrightarrow{N}, \overrightarrow{i})$$

is parameterised by

- a set of guard patterns $\mathcal{G}$, *e.g.*, $\mathcal{G} = \{g_{all}, g_{<}, g_{+1}\}$ with

$$\begin{array}{lcl} g_{all}(i, \hat{n}) & ::= & 0 < i < |\hat{n}| \\ g_{<}(i_1, i_2, \hat{n}) & ::= & 0 < i_1 < i_2 < |\hat{n}| \\ g_{+1}(i_1, i_2, i_3, \hat{n}) & ::= & 0 < i_1 <_1 i_2 <_1 i_3 < |\hat{n}| \end{array}$$

- a numerical abstract domain $\mathcal{A}_{\mathbb{Z}}$, *e.g.*, polyhedra, octagons, ...

$$\mathcal{A}_{\mathbb{Z}} \ni E(\overrightarrow{N}), U(\overrightarrow{N}, \overrightarrow{i})$$

$$\mathcal{A}_{\mathbb{U}} = (A^{\mathbb{U}}, \sqsubseteq^{\mathbb{U}}, \sqcup^{\mathbb{U}}, \sqcap^{\mathbb{U}}, \top^{\mathbb{U}}, \bot^{\mathbb{U}})$$

$$A^{\mathbb{U}} \ni E(\overrightarrow{N}) \wedge \bigwedge_{g \in \mathcal{G}} \forall \overrightarrow{i}.\ g(\overrightarrow{i}, \overrightarrow{N}) \Longrightarrow U(\overrightarrow{N}, \overrightarrow{i})$$

with sound but incomplete lattice operations:

- $E^1 \wedge (g_i \mapsto U_i^1)_i \ \sqsubseteq^{\mathbb{U}} \ E^2 \wedge (g_i \mapsto U_i^2)_i$
  if $E^1 \sqsubseteq^{\mathbb{Z}} E^2$ and for any i, $(E^1 \wedge U_i^1) \sqsubseteq^{\mathbb{Z}} (E^2 \wedge U_i^2)$
- $E^1 \wedge (g_i \mapsto U_i^1)_i \ \sqcup^{\mathbb{U}} \ E^2 \wedge (g_i \mapsto U_i^2)_i$
  gives $E^1 \sqcup^{\mathbb{Z}} E^2$ and for any i, $g_i \mapsto (E^1 \wedge U_i^1) \sqcup^{\mathbb{Z}} (E^2 \wedge U_i^2)$
- $E \wedge (g_i \mapsto U_i)_i \ = \ \top^{\mathbb{U}}$
  if $E = \top^{\mathbb{Z}}$ and for any i, $U_i = \top^{\mathbb{Z}}$
- $E \wedge (g_i \mapsto U_i)_i \ = \ \bot^{\mathbb{U}}$
  if $E = \bot^{\mathbb{Z}}$ or exists i such that $E \wedge U_i = \bot^{\mathbb{Z}}$

sound, but weaker than sat testing $\longrightarrow$ may delay termination!

Recall: Widening not needed for pure heap graphs (finite lattice)!

Like for lattice operations, $\nabla^{\mathbb{H}}$ is applied for values with isomorphic graphs, *i.e.*

$$(G_0, W_0)\ \nabla^{\mathbb{H}}\ (G_1, W_1)\ \triangleq\ (G_1, W_0 \nabla^{\mathbb{U}} W_1) \qquad \text{if } G_0 \approx_{\text{iso}} G_1$$

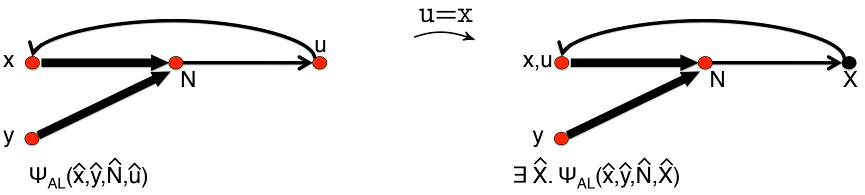
where

$$\left(E^0 \wedge (g_i \mapsto U_i^0)_i\right)\ \nabla^{\mathbb{U}}\ \left(E^1 \wedge (g_i \mapsto U_i^1)_i\right)\ \triangleq\ \left(E^2 \wedge (g_i \mapsto U_i^2)_i\right)$$

and

$$\begin{aligned} E^2 &\triangleq E^0\ \nabla^{\mathbb{Z}}\ E^1 \\ U_i^2 &\triangleq (E^0 \wedge U_i^0) \nabla^{\mathbb{Z}}\ (E^1 \wedge U_i^1) \end{aligned}$$

Abstract Transformers: Issues



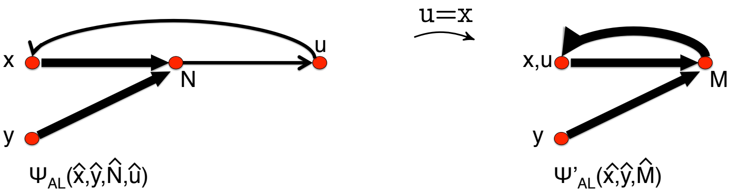
Issue 1:

Define a procedure for existential quantifier elimination in AL.

Required by Issue 1:

Define a procedure for concatenation of array properties in AL (`fold`).

Abstract Transformers: Issues

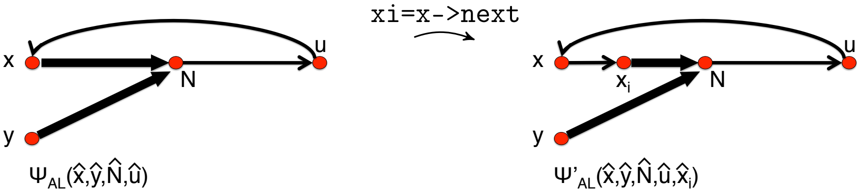


Issue 1:

Define a procedure for existential quantifier elimination in AL.

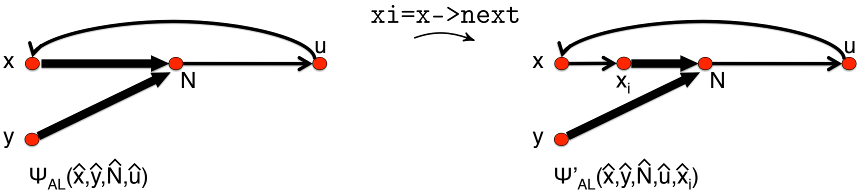
Required by Issue 1:

Define a procedure for concatenation of array properties in AL (`fold`).



Issue 2:

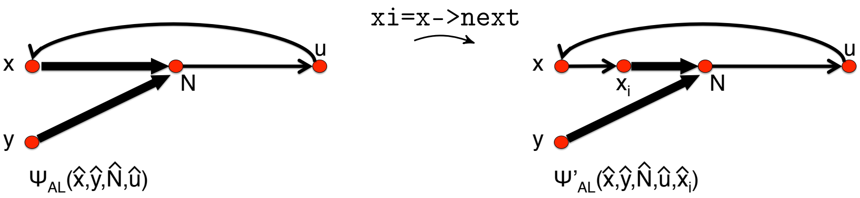
Define a procedure for universal formula unfolding at $i = 0$ (`unfold`).



Easy case: only $g_{all}(i, \widehat{n}) \ ::= \ 0 < i < |\widehat{n}|$

$$\begin{aligned}\Psi_{AL}(\widehat{x}, \widehat{y}, \widehat{N}, \widehat{u}) \quad = \quad & \widehat{x}[0] \geqslant 1 \ \wedge \ \widehat{y}[0]\%2 = 0 \ \wedge \ 2 \leqslant |\widehat{x}| \leqslant |\widehat{y}| \ \wedge \\ & \forall i.\ 0 < i < |\widehat{x}| \Longrightarrow \widehat{x}[i] \geqslant 1 \ \wedge \\ & \forall i.\ 0 < i < |\widehat{y}| \Longrightarrow \widehat{y}[i]\%2 = 0\end{aligned}$$

Abstract Transformers: unfold

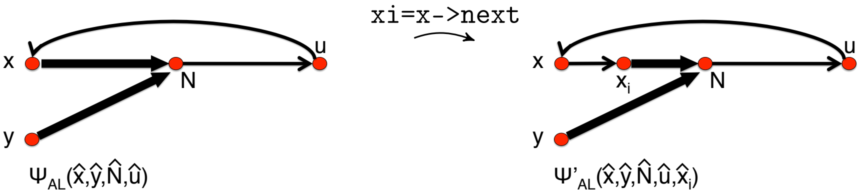


Easy case: only $g_{all}(i, \widehat{n}) \ ::= \ 0 < i < |\widehat{n}|$

$$\begin{aligned}\Psi_{AL}(\widehat{x}, \widehat{y}, \widehat{N}, \widehat{u}) \quad = \quad & \widehat{x}[0] \geqslant 1 \ \wedge \ \widehat{y}[0]\%2 = 0 \ \wedge \ 2 \leqslant |\widehat{x}| \leqslant |\widehat{y}| \ \wedge \\ & \forall i.\ 0 < i < |\widehat{x}| \Longrightarrow \widehat{x}[i] \geqslant 1 \ \wedge \\ & \forall i.\ 0 < i < |\widehat{y}| \Longrightarrow \widehat{y}[i]\%2 = 0\end{aligned}$$

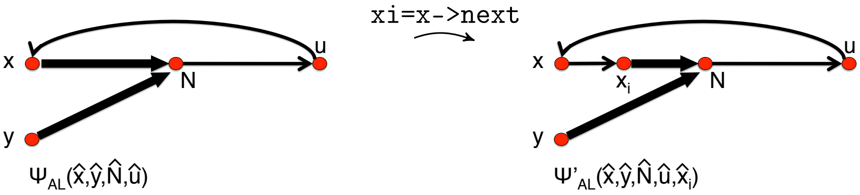
$$\begin{aligned}\Psi'_{AL}(\widehat{x}, \widehat{y}, \widehat{N}, \widehat{u}, \widehat{x_i}) \quad = \quad & \widehat{x}[0] \geqslant 1 \ \wedge \ \widehat{y}[0]\%2 = 0 \ \wedge \ |\widehat{x}| = 1 \ \wedge \ 2 \leqslant 1 + |\widehat{x_i}| \leqslant |\widehat{y}| \ \wedge \\ & \widehat{x_i}[0] \geqslant 1 \ \wedge \\ & \forall i.\ 0 < i < |\widehat{x_i}| \Longrightarrow \widehat{x_i}[i] \geqslant 1 \ \wedge \\ & \forall i.\ 0 < i < |\widehat{y}| \Longrightarrow \widehat{y}[i]\%2 = 0\end{aligned}$$

Abstract Transformers: unfold



Less easy case: only g_{all}, $g_{\leqslant}(i_1, i_2, \hat{n})$ ::= $0 < i_1 \leqslant i_2 < |\hat{n}|$

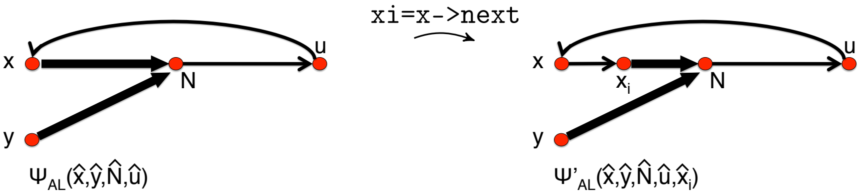
$$\begin{aligned}\Psi_{AL}(\hat{x},\hat{y},\hat{N},\hat{u}) \quad = \quad & \hat{x}[0] \geqslant 1 \ \wedge \ \hat{y}[0]\%2 = 0 \ \wedge \ 2 \leqslant |\hat{x}| \leqslant |\hat{y}| \ \wedge \\ & \forall i.\ 0 < i < |\hat{x}| \Longrightarrow \hat{x}[0] \geqslant \hat{x}[i] \ \wedge \\ & \forall i.\ 0 < i_1 \leqslant i_2 < |\hat{x}| \Longrightarrow \hat{x}[i_1] \geqslant \hat{x}[i_2] \ \wedge \\ & \forall i.\ 0 < i < |\hat{y}| \Longrightarrow \hat{y}[i]\%2 = 0\end{aligned}$$



Less easy case: only g_{all}, $g_{\leqslant}(i_1, i_2, \hat{n}) \ ::= \ 0 < i_1 \leqslant i_2 < |\hat{n}|$

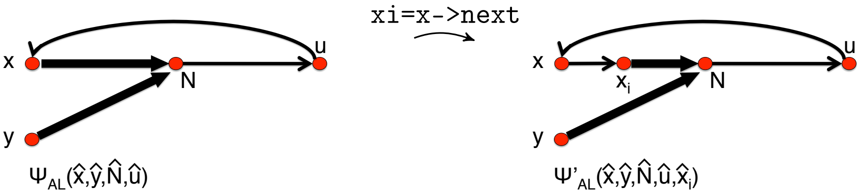
$$\begin{aligned}\Psi_{AL}(\hat{x},\hat{y},\hat{N},\hat{u}) \quad = \quad & \hat{x}[0] \geqslant 1 \ \wedge \ \hat{y}[0]\%2 = 0 \ \wedge \ 2 \leqslant |\hat{x}| \leqslant |\hat{y}| \ \wedge \\ & \forall i.\ 0 < i < |\hat{x}| \Longrightarrow \hat{x}[0] \geqslant \hat{x}[i] \ \wedge \\ & \forall i.\ 0 < i_1 \leqslant i_2 < |\hat{x}| \Longrightarrow \hat{x}[i_1] \geqslant \hat{x}[i_2] \ \wedge \\ & \forall i.\ 0 < i < |\hat{y}| \Longrightarrow \hat{y}[i]\%2 = 0\end{aligned}$$

$$\begin{aligned}\Psi'_{AL}(\hat{x},\hat{y},\hat{N},\hat{u},\hat{x_i}) \quad = \quad & \hat{x}[0] \geqslant 1 \ \wedge \ \hat{y}[0]\%2 = 0 \ \wedge \ |\hat{x}| = 1 \ \wedge \ 2 \leqslant 1 + |\hat{x_i}| \leqslant |\hat{y}| \ \wedge \\ & \hat{x}[0] \geqslant \hat{x_i}[0] \ \wedge \forall i.\ 0 < i < |\hat{x_i}| \Longrightarrow \hat{x_i}[0] \geqslant \hat{x_i}[i] \ \wedge \\ & \forall i.\ 0 < i_1 \leqslant i_2 < |\hat{x_i}| \Longrightarrow \hat{x_i}[i_1] \geqslant \hat{x_i}[i_2] \ \wedge \\ & \forall i.\ 0 < i < |\hat{y}| \Longrightarrow \hat{y}[i]\%2 = 0\end{aligned}$$



Difficult case: only g_{all}, $g_{+1}(i_1, i_2, \widehat{n}) \;::=\; 0 < i_1 <_1 i_2 < |\widehat{n}|$

$$
\begin{aligned}
\Psi_{AL}(\widehat{x}, \widehat{y}, \widehat{N}, \widehat{u}) \quad = \quad & \widehat{x}[0] \geqslant 1 \;\wedge\; \widehat{y}[0]\%2 = 0 \;\wedge\; 2 \leqslant |\widehat{x}| \leqslant |\widehat{y}| \;\wedge \\
& \forall i.\ 0 < i_1 <_1 i_2 < |\widehat{x}| \Longrightarrow \widehat{x}[i_1] + 2 = \widehat{x}[i_2] \;\wedge \\
& \forall i.\ 0 < i < |\widehat{y}| \Longrightarrow \widehat{y}[i]\%2 = 0
\end{aligned}
$$



Difficult case: only g_{all}, $g_{+1}(i_1, i_2, \hat{n}) ::= 0 < i_1 <_1 i_2 < |\hat{n}|$

$$\begin{aligned}\Psi_{AL}(\hat{x},\hat{y},\hat{N},\hat{u}) \quad = \quad & \hat{x}[0] \geqslant 1 \ \wedge\ \hat{y}[0]\%2 = 0 \ \wedge\ 2 \leqslant |\hat{x}| \leqslant |\hat{y}| \ \wedge \\ & \forall i.\ 0 < i_1 <_1 i_2 < |\hat{x}| \implies \hat{x}[i_1] + 2 = \hat{x}[i_2] \ \wedge \\ & \forall i.\ 0 < i < |\hat{y}| \implies \hat{y}[i]\%2 = 0\end{aligned}$$

$$\begin{aligned}\Psi'_{AL}(\hat{x},\hat{y},\hat{N},\hat{u},\hat{x}_i) \quad = \quad & \hat{x}[0] \geqslant 1 \ \wedge\ \hat{y}[0]\%2 = 0 \ \wedge\ |\hat{x}| = 1 \ \wedge\ 2 \leqslant 1 + |\hat{x}_i| \leqslant |\hat{y}| \ \wedge \\ & ??? \\ & \forall i.\ 0 < i_1 <_1 i_2 < |\hat{x}_i| \implies \hat{x}_i[i_1] + 2 = \hat{x}_i[i_2] \ \wedge \\ & \forall i.\ 0 < i < |\hat{y}| \implies \hat{y}[i]\%2 = 0\end{aligned}$$

Closure of $\mathcal{G}$

For each $g(i_1, i_2, \ldots, \overrightarrow{N}) \in \mathcal{G}$, add to $\mathcal{G}$ a new guard, $g' \equiv g(1, i_2 + 1, \ldots, \overrightarrow{N})$ which collects informations about unfolding of g ... and so on for g'!

Example: for $g_{+1}(i_1, i_2, \widehat{n}) \ ::= \ 0 < i_1 <_1 i_2 < |\widehat{n}|$

$\longrightarrow$ add $g_1(i, \widehat{n}) \ ::= \ 0 < i_1 = 1 < |\widehat{n}|$

$$\begin{aligned} \Psi_{AL}(\widehat{x}, \widehat{y}, \widehat{N}, \widehat{u}) \quad = \quad & \widehat{x}[0] \geqslant 1 \ \wedge \ \widehat{y}[0]\%2 = 0 \ \wedge \ 2 \leqslant |\widehat{x}| \leqslant |\widehat{y}| \ \wedge \\ & \forall i.\ 0 < i_1 <_1 i_2 < |\widehat{x}| \Longrightarrow \widehat{x}[i_1] + 2 = \widehat{x}[i_2] \ \wedge \\ & \forall i.\ 0 < i < |\widehat{y}| \Longrightarrow \widehat{y}[i]\%2 = 0 \end{aligned}$$

gives

$$\begin{aligned} \Psi'_{AL}(\widehat{x}, \widehat{y}, \widehat{N}, \widehat{u}, \widehat{x_i}) \quad = \quad & \widehat{x}[0] \geqslant 1 \ \wedge \ \widehat{y}[0]\%2 = 0 \ \wedge \ |\widehat{x}| = 1 \ \wedge \ 2 \leqslant 1 + |\widehat{x_i}| \leqslant |\widehat{y}| \ \wedge \\ & \forall i.\ 0 < i = 1 < |\widehat{x_i}| \Longrightarrow \widehat{x_i}[0] + 2 = \widehat{x_i}[i] \ \wedge \\ & \forall i.\ 0 < i_1 <_1 i_2 < |\widehat{x_i}| \Longrightarrow \widehat{x_i}[i_1] + 2 = \widehat{x_i}[i_2] \ \wedge \\ & \forall i.\ 0 < i < |\widehat{y}| \Longrightarrow \widehat{y}[i]\%2 = 0 \end{aligned}$$

Experimental Results

Program	$\mathcal{A}_{\mathbb{W}}$	$\mathcal{A}_{\mathbb{Z}}$	k	*property*	SEC
dispatch	$\mathbb{U}$	poly	1	$g_{all}(i, \widehat{grt}) \Longrightarrow \widehat{grt}[i] \geqslant 3$	0.4
	Σ	poly	0	$\Sigma(\widehat{grt}) \geqslant 3 \times \lvert\widehat{grt}\rvert$	0.4
	$\mathbb{M}$	poly	0	$ms(\widehat{grt}) + ms(\widehat{less}) = ms(\widehat{head})$	1
initFibo	$\mathbb{U}$	poly	1	$g_{<}(i, i', \widehat{n}) \Longrightarrow \widehat{n}[i'] - \widehat{n}[i] \geqslant i' - i$	1
	$\mathbb{U}$	poly	3	$g_{+1}(i_1, i_2, i_3, \widehat{n}) \Longrightarrow \widehat{n}[i_3] = \widehat{n}[i_1] + \widehat{n}[i_2]$	0.5
	Σ	poly	0	$\Sigma_{i=1,N} F_i = 2 \times F_N + F_{N-1} - 1$	0.4
init2N	$\mathbb{U}$	poly	1	$g_{all}(i, \widehat{n}) \Longrightarrow \widehat{n}[i] = 2 \times i$	0.4
	Σ	poly	0	$\Sigma(\widehat{n}) \geqslant 2 \times \lvert\widehat{n}\rvert - 2$	0.5
bubbleSort	$\mathbb{M}$	poly	0	$ms(\widehat{n}) = ms\_init$	0.4
	$\mathbb{U}$	oct	1	$g_{all}(i, \widehat{n}) \Longrightarrow \widehat{n}[i] \geqslant \widehat{n}[0]$	0.6
	$\mathbb{U}$	oct	2	$g_{<}(i_1, i_2, \widehat{n}) \Longrightarrow \widehat{n}[i_1] \leqslant \widehat{n}[i_2]$	2
insertSort	$\mathbb{M}$	poly	0	$ms(\widehat{n}) = ms\_init$	0.4
	$\mathbb{U}$	oct	1	$g_{all}(i, \widehat{n}) \Longrightarrow \widehat{n}[i] \geqslant \widehat{n}[0]$	5
	$\mathbb{U}$	oct	2	$g_{<}(i_1, i_2, \widehat{n}) \Longrightarrow \widehat{n}[i_1] \leqslant \widehat{n}[i_2]$	36
copyReverse	$\mathbb{M}$	poly	0	$ms(\widehat{rev}) = ms(\widehat{n})$	0.4
	Σ	poly	0	$\Sigma(\widehat{rev}) = \Sigma(\widehat{n})$	0.03

Related Works on Array Analysis

- predicate abstraction [Flanagan & Qadeer, 02],[Lahiri *et al*, 03]
  $\longrightarrow$ only fixed properties for data constraints

- abstract interpretation [Blanchet *et al*, 03], [Gopan *et al*, 04-07]
  $\longrightarrow$ *e.g.*, [Halbwachs & Péron, 08] infers

$$\forall i \in I.\ \varphi(a_1[i+k_1], ..., a_m[i+k_m], k)$$

- symbol elimination in loop body [Kovacs *et al*, 09-11]
  $\longrightarrow$ slides at VTSA'14

Conclusion of Intra-procedural Shape Analysis

- Abstract interpretation principles work for more complex constraints.

- Building new abstract domains may be a challenging task.

- Free numerical domains exists with a clear interface.

- Experimental results are good for realistic programs.

Outline

Programs with Lists and Data

— Static Analysis for Decision Procedures —

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

VMCAI'12

$$\text{SLAD} \triangleq \text{Separation Logic}(\mathtt{ls}^{+}) + \text{Array Logic}(\mathbb{Z})$$

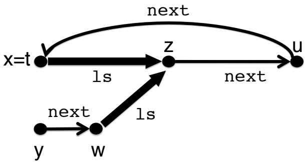
Corollary

Satisfiability and entailment in SLAD are undecidable.

Theorem

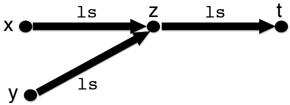
$\exists$ a sound procedure for checking satisfiability and entailment, which is complete when Array Logic has only $\leqslant$-constraints in $g(\overrightarrow{i})$.

Main idea: Apply compositionally a syntactic check and, if it fails, strengthen the array formulas using program analysis with an abstraction given by SLAD formulas, and apply the syntactic check.



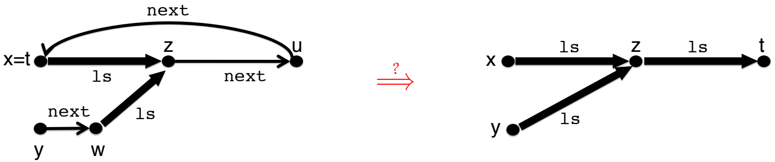
$$\hat{x}[0] \geq 2 \wedge \mathrm{sorted}_{<}(\hat{x}) \wedge$$
$$\hat{z}[0] = 0 \wedge \hat{u}[0] = 1 \wedge$$
$$\hat{y}[0] + 1 = \hat{w}[0] \geq 2 \wedge \mathrm{succ}(\hat{w})$$

$$\stackrel{?}{\Longrightarrow}$$



$$\hat{x}[0] \geq 1 \wedge \mathrm{sorted}_{\leqslant}(\hat{x}) \wedge$$
$$\mathrm{succ}(\hat{z}) \wedge$$
$$\mathrm{all}_{\geq 1}(\hat{y})$$

Main idea: Apply compositionally a syntactic check and, if it fails, strengthen the array formulas using program analysis with an abstraction given by SLAD formulas, and apply the syntactic check.

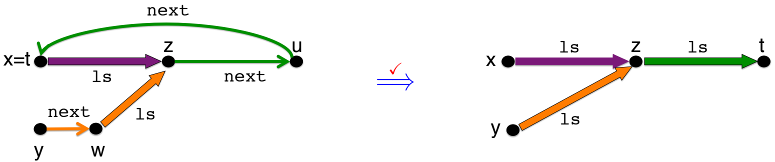


$\widehat{x}[0] \geq 2 \land \text{sorted}_{<}(\widehat{x}) \land$ $\stackrel{?}{\Longrightarrow}$ $\widehat{x}[0] \geq 1 \land \text{sorted}_{\leqslant}(\widehat{x}) \land$

$\widehat{z}[0] = 0 \land \widehat{u}[0] = 1 \land$ $\stackrel{?}{\Longrightarrow}$ $\text{succ}(\widehat{z}) \land$

$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \land \text{succ}(\widehat{w})$ $\stackrel{?}{\Longrightarrow}$ $\text{all}_{\geq 1}(\widehat{y})$

Main idea: Apply compositionally a syntactic check and, if it fails, strengthen the array formulas using program analysis with an abstraction given by SLAD formulas, and apply the syntactic check.

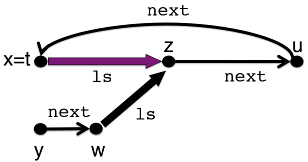


$$\widehat{x}[0] \geq 2 \land \mathrm{sorted}_{<}(\widehat{x}) \land \quad \stackrel{?}{\Longrightarrow} \quad \widehat{x}[0] \geq 1 \land \mathrm{sorted}_{\leqslant}(\widehat{x}) \land$$

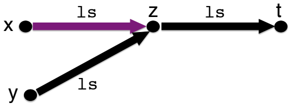
$$\widehat{z}[0] = 0 \land \widehat{u}[0] = 1 \land \quad \stackrel{?}{\Longrightarrow} \quad \mathrm{succ}(\widehat{z}) \land$$

$$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \land \mathrm{succ}(\widehat{w}) \quad \stackrel{?}{\Longrightarrow} \quad \mathrm{all}_{\geq 1}(\widehat{y})$$

Main idea: Apply compositionally a syntactic check and, if it fails, strengthen the array formulas using program analysis with an abstraction given by SLAD formulas, and apply the syntactic check.



$\overset{\checkmark}{\Longrightarrow}$



$\widehat{x}[0] \geq 2 \wedge \text{sorted}_{<}(\widehat{x}) \wedge \quad \overset{\checkmark}{\Longrightarrow} \quad \widehat{x}[0] \geq 1 \wedge \text{sorted}_{\leqslant}(\widehat{x}) \wedge$

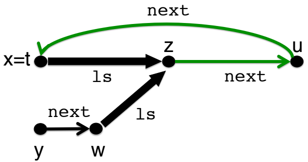
$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge \quad \overset{?}{\Longrightarrow} \quad \text{succ}(\widehat{z}) \wedge$

$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \text{succ}(\widehat{w}) \quad \overset{?}{\Longrightarrow} \quad \text{all}_{\geq 1}(\widehat{y})$

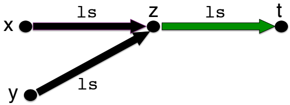
$$\text{sorted}_{<}(\widehat{x}) \equiv \forall i_1, i_2.\ 0 \leqslant i_1 < i_2 < \text{len}(\widehat{x}) \Longrightarrow \widehat{x}[i_1] < \widehat{x}[i_2]$$

$$\text{sorted}_{\leqslant}(\widehat{x}) \equiv \forall i_1, i_2.\ 0 \leqslant i_1 < i_2 < \text{len}(\widehat{x}) \Longrightarrow \widehat{x}[i_1] \leqslant \widehat{x}[i_2]$$

Main idea: Apply compositionally a syntactic check and, if it fails, strengthen the array formulas using program analysis with an abstraction given by SLAD formulas, and apply the syntactic check.



$\overset{\checkmark}{\Longrightarrow}$



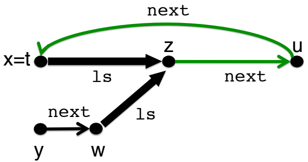
$\widehat{x}[0] \geq 2 \wedge \mathrm{sorted}_<(\widehat{x}) \wedge \quad \overset{\checkmark}{\Longrightarrow} \quad \widehat{x}[0] \geq 1 \wedge \mathrm{sorted}_\leqslant(\widehat{x}) \wedge$

$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge \quad \overset{?}{\Longrightarrow} \quad \mathrm{succ}(\widehat{z}) \wedge$

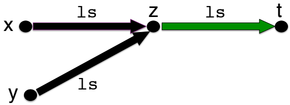
$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathrm{succ}(\widehat{w}) \quad \overset{?}{\Longrightarrow} \quad \mathrm{all}_{\geq 1}(\widehat{y})$

$$\mathrm{succ}(\widehat{z}) \quad \equiv \quad \forall i_1, i_2.\ 0 \leqslant i_1, i_2 < \mathrm{len}(\widehat{z}) \wedge i_1 + 1 = i_2 \Longrightarrow \widehat{z}[i_1] + 1 = \widehat{z}[i_2]$$

Main idea: Apply compositionally a syntactic check and, if it fails, strengthen the array formulas using program analysis with an abstraction given by SLAD formulas, and apply the syntactic check.



$\overset{\checkmark}{\Longrightarrow}$



$\widehat{x}[0] \geq 2 \wedge \text{sorted}_<(\widehat{x}) \wedge$ $\overset{\checkmark}{\Longrightarrow}$ $\widehat{x}[0] \geq 1 \wedge \text{sorted}_\leqslant(\widehat{x}) \wedge$

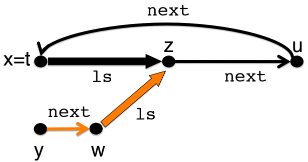
$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge \text{succ}(\widehat{z \cdot u}) \wedge$ $\overset{\checkmark}{\Longrightarrow}$ $\text{succ}(\widehat{z}) \wedge$

$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \text{succ}(\widehat{w})$ $\overset{?}{\Longrightarrow}$ $\text{all}_{\geq 1}(\widehat{y})$

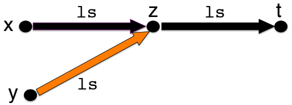
$$\text{succ}(\widehat{z}) \equiv \forall i_1, i_2.\ 0 \leqslant i_1, i_2 < \text{len}(\widehat{z}) \wedge i_1 + 1 = i_2 \Longrightarrow \widehat{z}[i_1] + 1 = \widehat{z}[i_2]$$

Entailment Procedure in a Nutshell

Main idea: Apply compositionally a syntactic check and, if it fails, strengthen the array formulas using program analysis with an abstraction given by SLAD formulas, and apply the syntactic check.



$\stackrel{\checkmark}{\Longrightarrow}$



$\widehat{x}[0] \geq 2 \wedge \mathsf{sorted}_{<}(\widehat{x}) \wedge \quad \stackrel{\checkmark}{\Longrightarrow} \quad \widehat{x}[0] \geq 1 \wedge \mathsf{sorted}_{\leqslant}(\widehat{x}) \wedge$

$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge \mathsf{succ}(\widehat{z \cdot u}) \wedge \quad \stackrel{\checkmark}{\Longrightarrow} \quad \mathsf{succ}(\widehat{z}) \wedge$

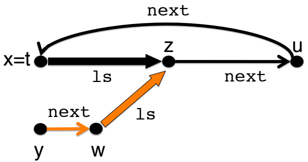
$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathsf{succ}(\widehat{w}) \quad \stackrel{?}{\Longrightarrow} \quad \mathsf{all}_{\geq 1}(\widehat{y})$

$$\mathsf{succ}(\widehat{w}) \equiv \forall i_1, i_2.\ 0 \leqslant i_1, i_2 < \mathsf{len}(\widehat{w}) \wedge i_1 + 1 = i_2 \Longrightarrow \widehat{w}[i_1] + 1 = \widehat{w}[i_2]$$

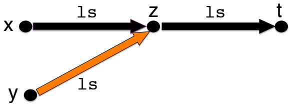
$$\mathsf{all}_{\geq 1}(\widehat{y}) \equiv \forall i.\ 0 \leqslant i < \mathsf{len}(\widehat{y}) \Longrightarrow \widehat{y}[i] \geq 1$$

Entailment Procedure in a Nutshell

Main idea: Apply compositionally a syntactic check and, if it fails, strengthen the array formulas using program analysis with an abstraction given by SLAD formulas, and apply the syntactic check.



$\overset{\checkmark}{\Longrightarrow}$



$\widehat{x}[0] \geq 2 \wedge \mathrm{sorted}_{<}(\widehat{x}) \wedge \quad \overset{\checkmark}{\Longrightarrow} \quad \widehat{x}[0] \geq 1 \wedge \mathrm{sorted}_{\leqslant}(\widehat{x}) \wedge$

$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge \mathrm{succ}(\widehat{z \cdot u}) \wedge \quad \overset{\checkmark}{\Longrightarrow} \quad \mathrm{succ}(\widehat{z}) \wedge$

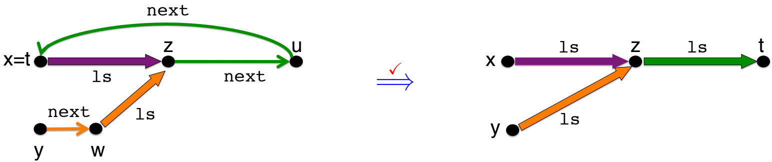
$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathrm{succ}(\widehat{w}) \quad \overset{\checkmark}{\Longrightarrow} \quad \mathrm{all}_{\geq 1}(\widehat{y})$

$\wedge\ \mathrm{all}_{\geq 1}(\widehat{y \cdot w})$

$$\mathrm{succ}(\widehat{w}) \equiv \forall i_1, i_2.\ 0 \leqslant i_1, i_2 < \mathrm{len}(\widehat{w}) \wedge i_1 + 1 = i_2 \Longrightarrow \widehat{w}[i_1] + 1 = \widehat{w}[i_2]$$

$$\mathrm{all}_{\geq 1}(\widehat{y}) \equiv \forall i.\ 0 \leqslant i < \mathrm{len}(\widehat{y}) \Longrightarrow \widehat{y}[i] \geq 1$$

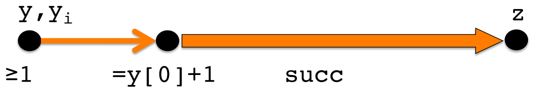
Main idea: Apply compositionally a syntactic check and, if it fails, strengthen the array formulas using program analysis with an abstraction given by SLAD formulas, and apply the syntactic check.



$\widehat{x}[0] \geq 2 \wedge \mathrm{sorted}_{<}(\widehat{x}) \wedge$	$\stackrel{\checkmark}{\Longrightarrow}$	$\widehat{x}[0] \geq 1 \wedge \mathrm{sorted}_{\leqslant}(\widehat{x}) \wedge$
$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge \mathrm{succ}(\widehat{z \cdot u}) \wedge$	$\stackrel{\checkmark}{\Longrightarrow}$	$\mathrm{succ}(\widehat{z}) \wedge$
$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathrm{succ}(\widehat{w})$ $\wedge\ \mathrm{all}_{\geq 1}(\widehat{y \cdot w})$	$\stackrel{\checkmark}{\Longrightarrow}$	$\mathrm{all}_{\geq 1}(\widehat{y})$

Main idea: Do analysis presented before on each list segment using as $\mathcal{A}_{\mathbb{H}}$ the SLAD formulas! The guards in universal formulas, size and data constraints fix the abstract domain of integer words.

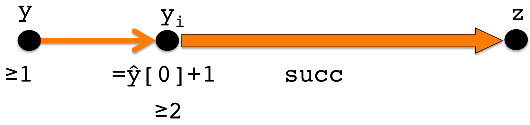
Main idea: Do analysis presented before on each list segment using as $\mathcal{A}_{\mathbb{H}}$ the SLAD formulas! The guards in universal formulas, size and data constraints fix the abstract domain of integer words.




```
/* @assume: y->w * ls(w,z),
            ŷ[0]+1=ŵ[0]≥2, succ(ŵ) */
yi=y;
/* @inv: ls(y,yi) * ls(yi,z), …
         ∀i 0≤i<len(y.w) => ?   */
while(yi!= z)
  yi=yi->next;
```

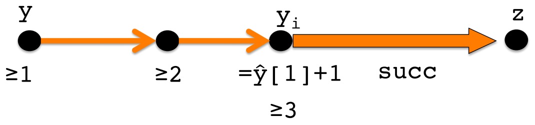
in G, used in all≥1

Main idea: Do analysis presented before on each list segment using as $\mathcal{A}_{\mathbb{H}}$ the SLAD formulas! The guards in universal formulas, size and data constraints fix the abstract domain of integer words.



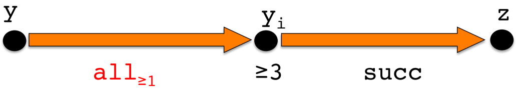

```
/* @assume: y->w * ls(w,z),
            ŷ[0]+1=ŵ[0]≥2, succ(ŵ) */
yi=y;
/* @inv: ls(y,yi) * ls(yi,z), …
         ∀i 0≤i<len(ŷ) => ŷ[i]≥1 */
while(yi!= z)
  yi=yi->next;
```

Main idea: Do analysis presented before on each list segment using as $\mathcal{A}_{\mathbb{H}}$ the SLAD formulas! The guards in universal formulas, size and data constraints fix the abstract domain of integer words.



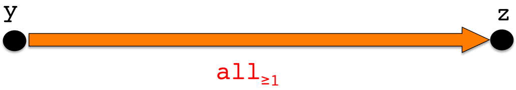

```
/* @assume: y->w * ls(w,z),
            ŷ[0]+1=ŵ[0]≥2, succ(ŵ) */
yi=y;
/* @inv: ls(y,yi) * ls(yi,z), …
        ∀i 0≤i<len(ŷ)=>ŷ[i]≥1 ⊔ ŷ[i]≥2*/
while(yi!= z)
  yi=yi->next;
```

Main idea: Do analysis presented before on each list segment using as $\mathcal{A}_{\mathbb{H}}$ the SLAD formulas! The guards in universal formulas, size and data constraints fix the abstract domain of integer words.




```
/* @assume: y->w * ls(w,z),
             ŷ[0]+1=ŵ[0]≥2, succ(ŵ) */
yi=y;
/* @inv: ls(y,yi) * ls(yi,z),…
         ∀i 0≤i<len(ŷ)=> ŷ[i]≥1 */
while(yi!= z)
  yi=yi->next;
```

Main idea: Do analysis presented before on each list segment using as $\mathcal{A}_{\mathbb{H}}$ the SLAD formulas! The guards in universal formulas, size and data constraints fix the abstract domain of integer words.




```
/* @assume: y->w * ls(w,z),
            ŷ[0]+1=ŵ[0]≥2, succ(ŵ) */
yi=y;
/* @inv: ls(y,yi) * ls(yi,z), …
         ∀i 0≤i<len(ŷ)=> ŷ[i]≥1 */
while(yi!= z)
  yi=yi->next;
```

Outline

Formal Semantics for Procedure Call

Stack	$\mathtt{Stacks} \triangleq [(\mathbf{CP} \times \mathbf{P} \times (\mathbf{DV} \nrightarrow \mathbb{D} \;\cup\; \mathbf{RV} \nrightarrow \mathbb{L}))^{*}] \ni \mathtt{S}$
Memory	$\mathtt{Mem} \triangleq \mathtt{Stacks} \times \mathtt{Heaps} \ni \mathtt{m}$
Configurations	$\mathtt{Config} \triangleq \mathbf{CP} \times (\mathtt{Mem} \cup \{\mathtt{merr}\}) \ni \mathtt{C}$

$$\frac{\forall v_i \in \overrightarrow{vin}\,.(\mathtt{S},\mathtt{H}) \vdash v_i \rightsquigarrow c_i \neq \mathtt{merr}}{(\ell,(\mathtt{S},\mathtt{H})) \vdash v{=}P(\overrightarrow{vin},\overrightarrow{vout}) \rightsquigarrow (start_P,(push(\mathtt{S},\ell+1,P,v,\overrightarrow{vout},\overrightarrow{c_i},\mathtt{lv}_P),\mathtt{H}))}$$

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

Another source of infinity is the unbounded stack that usually stores locations in the heap.

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

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.

Functional Context-Sensitive

Aim

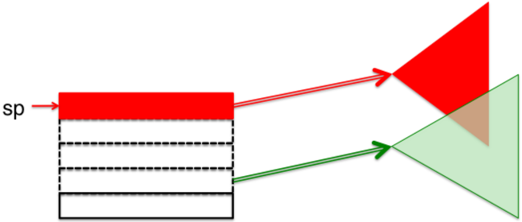
Compute a function $summary_P : \mathbf{CP} \nrightarrow (\mathcal{A}_{\mathbb{H}} \rightarrow \mathcal{A}_{mH})$ mapping each control point of the procedure $q \in \mathbf{CP}$ to a function which associates every (G_0, W_0) abstract heap reachable at $start_P$ to the abstract heap (G_q, W_q) reachable at q.

```
   int length(list* l) {
1:   int len = 0;
2:   if (l == NULL)
3:     len=0;
4:   else {
5:     len=1+length(l->next);
6:   }
7:   return len;
8: }
```

q	(G_0, W_0)	(G_q, W_q)
2	$l = \boxtimes$	$l = \boxtimes$
	$\mathtt{ls}^+(l, \boxtimes)$	$\mathtt{ls}^+(l, \boxtimes)$
...	...	...
8	$l = \boxtimes$	$\$ret = 0 \land l = \boxtimes$
	$\mathtt{ls}^+(l, \boxtimes)$	$\$ret \geqslant 1 \land \mathtt{ls}^+(l, \boxtimes)$

Problem

The local heap of a procedure may be accessed from the stack bypassing the actual parameters.

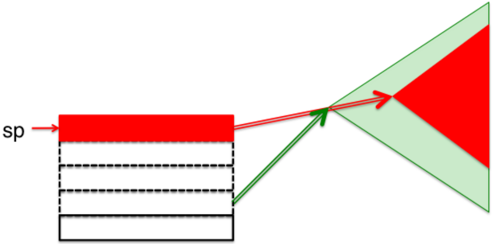


Bad Consequence

Context sensitive analyses shall track also these interferences!

Observation

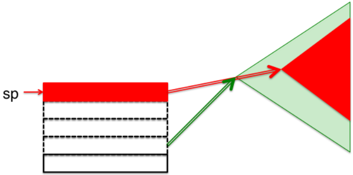
In a large class of programs with procedure calls, the local heap is reachable from the stack by passing through the actual parameters.

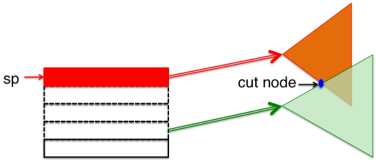


Consequence

For this class, the computation of summaries is compositional.

[Rinetzky *et al*,05]





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 $\mathtt{start_P}$ over a new vocabulary V^0
  $\longrightarrow$ similar to `old` notation in JML
- H is the heap at the control point q of the procedure over $V \cup \{\$ret\}$

Definition

A concrete procedure summary is the set $\{(H^0, H_{end_P})\}$.

Outline

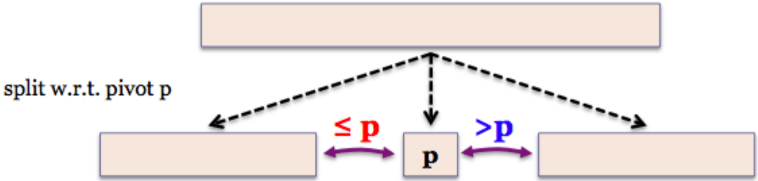
Programs with Lists and Data

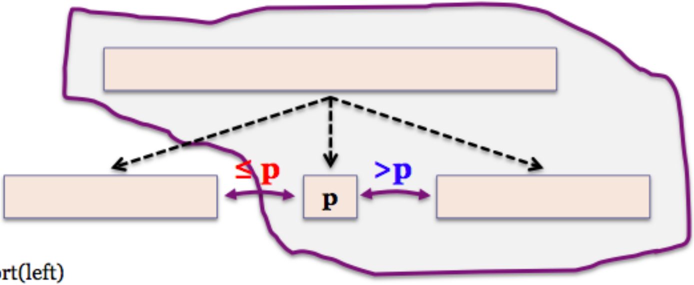
— Inter-procedural Analysis —

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

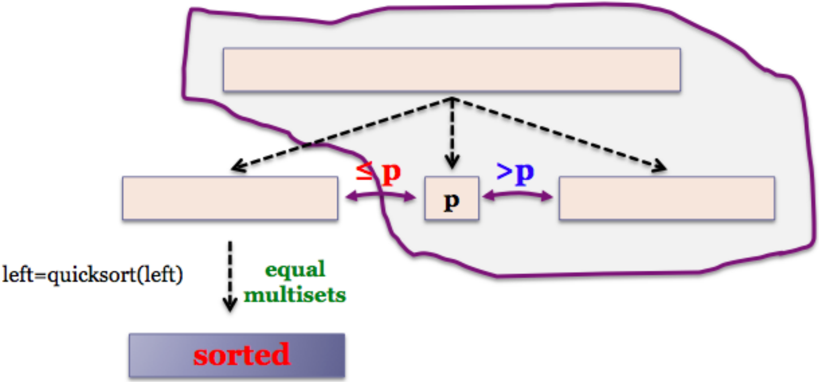
PLDI'11

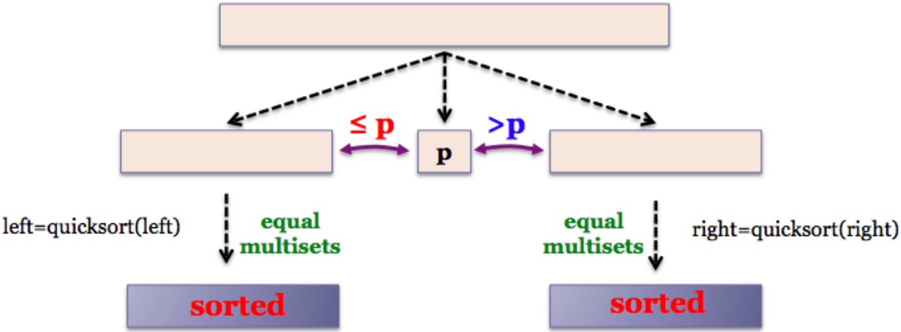
Running Example: Quicksort on Lists (with Copy)



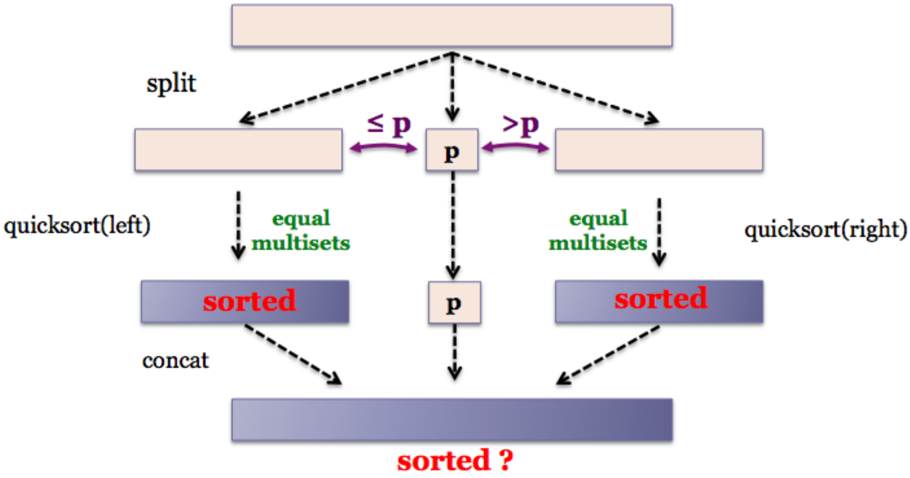


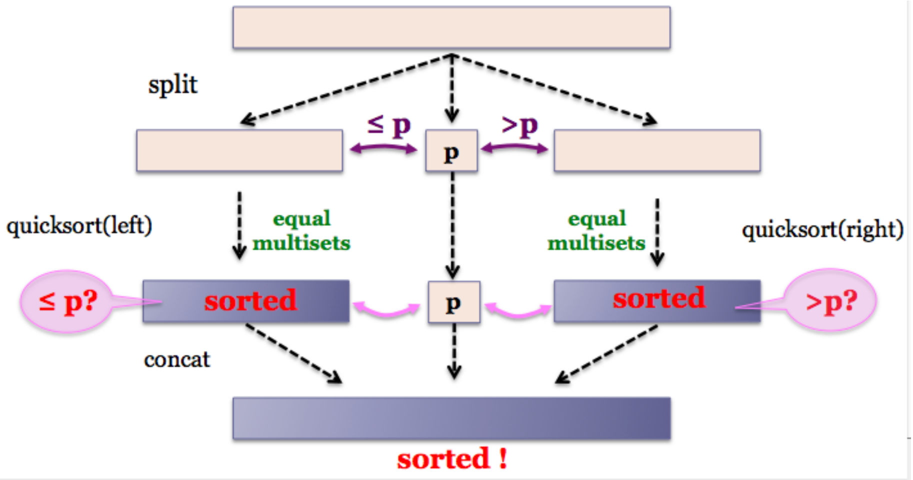
left=quicksort(left)


≤p
p
>p
left=quicksort(left)
equal
multisets
sorted

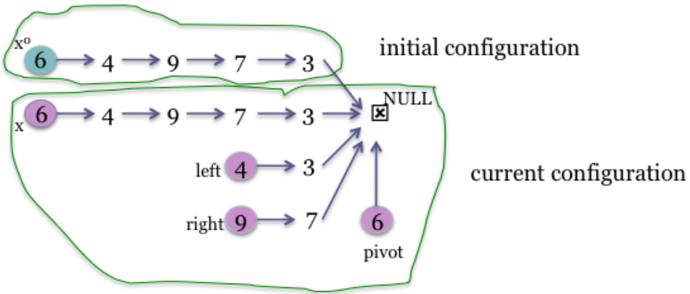

≤ p
p
>p
left=quicksort(left)
equal
multisets
equal
multisets
right=quicksort(right)
sorted
sorted

Running Example: Quicksort on Lists (with Copy)



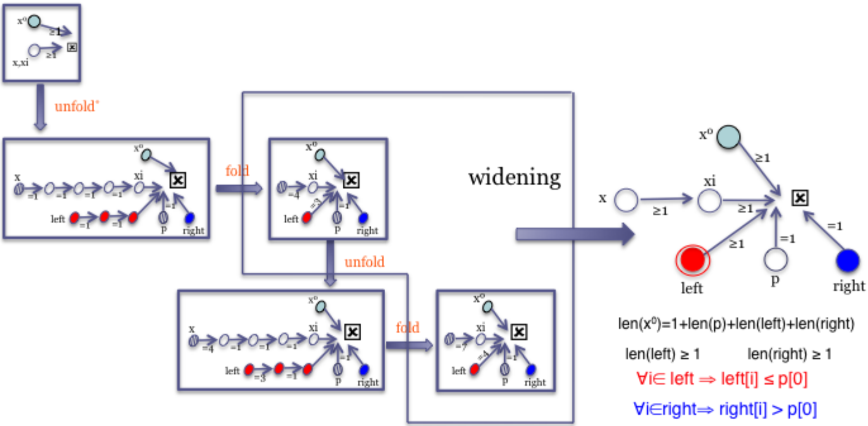

split
≤ p
p
>p
quicksort(left)
equal
multisets
equal
multisets
quicksort(right)
≤ p?
sorted
p
sorted
>p?
concat
sorted !

Representing Summaries in $\mathcal{A}_{\mathrm{HS}}$

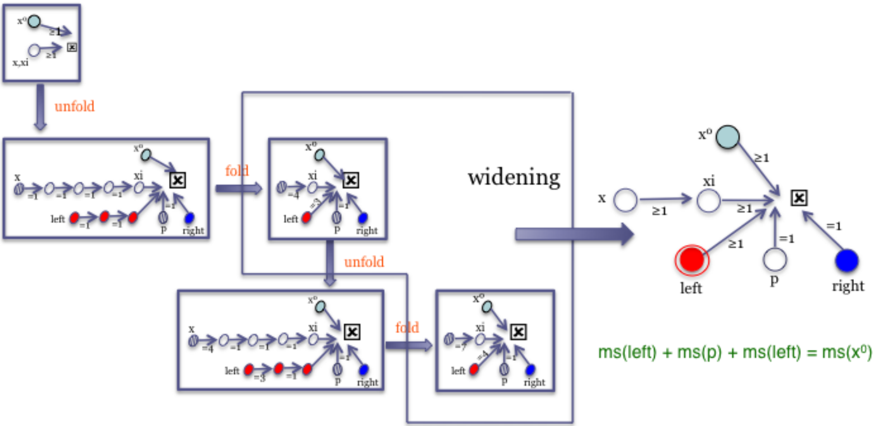


$$\alpha\big((H^0, H)\big) \triangleq (G^0 * G,\ W^0 \wedge W)$$

Analysing Quicksort: $\mathcal{A}^{\mathbb{U}}$ Domain

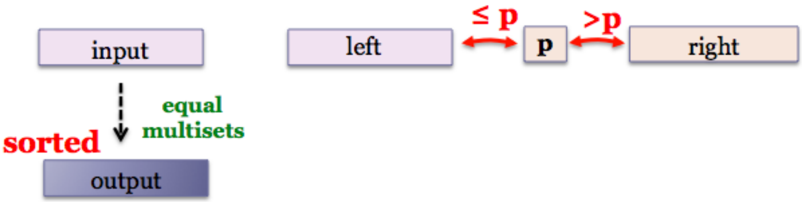


ANALYSING QUICKSORT: $\mathcal{A}^{\mathrm{M}}$ DOMAIN

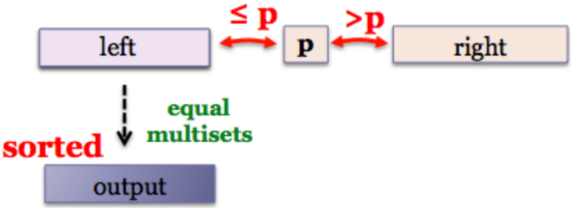


ms(left) + ms(p) + ms(left) = ms(x^0)

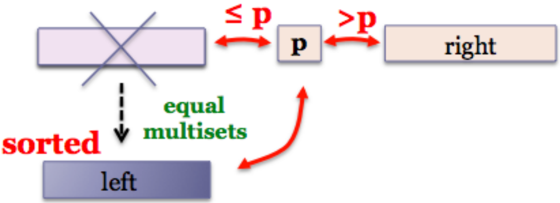
left = quicksort(left)

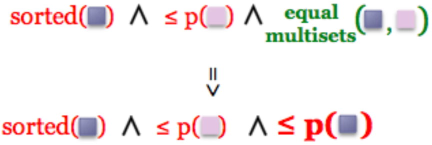


left = quicksort(left)



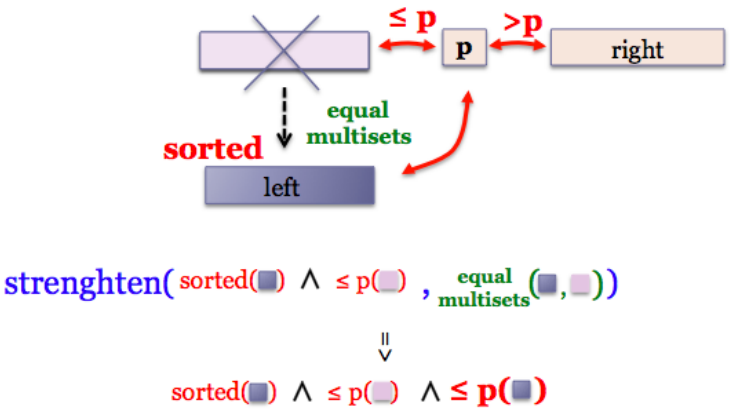
left = quicksort(left)



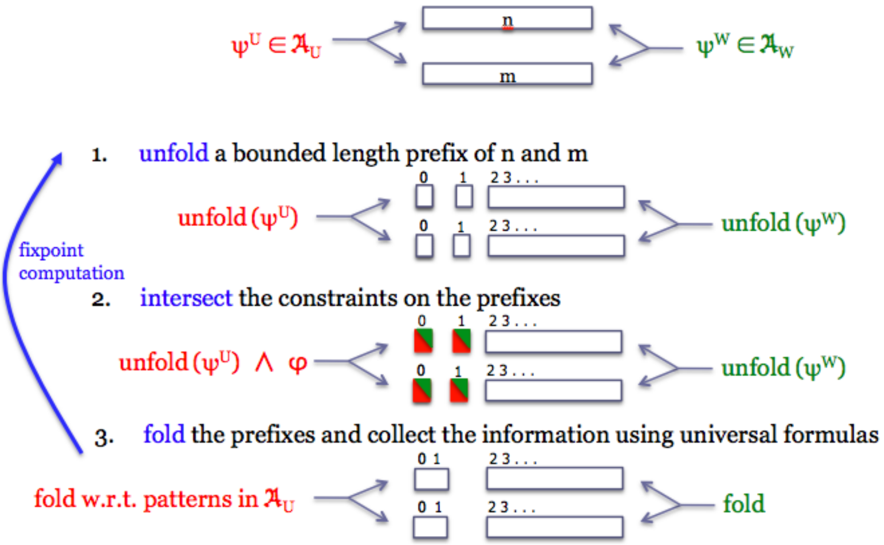


Quicksort: Loss of Precision

left = quicksort(left)

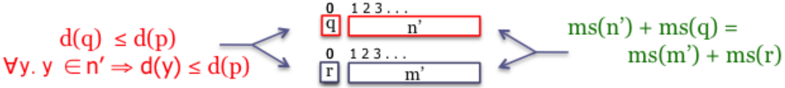


Strengthen Procedure



Strengthen Procedure: Example

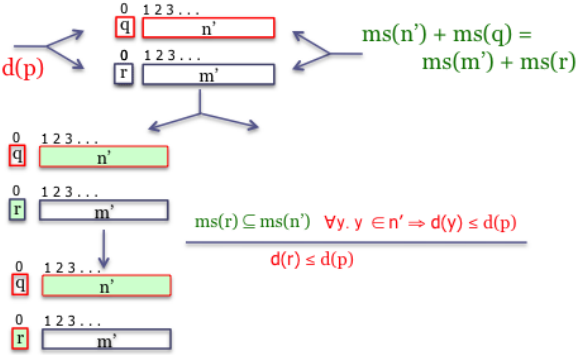
1. unfold a bounded length prefix of n and m



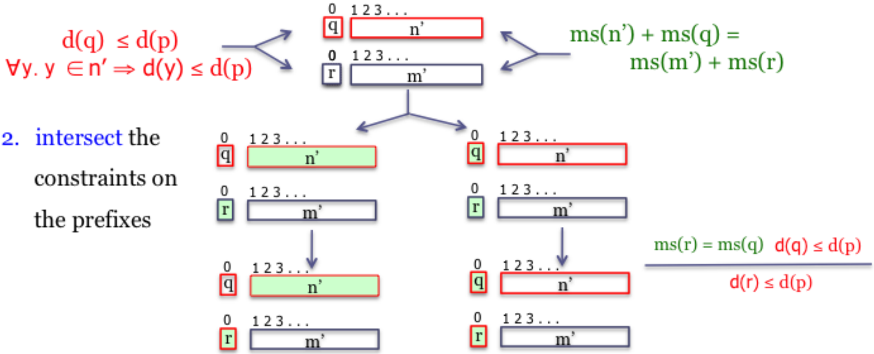
$d(q) \leq d(p)$

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

2. intersect the constraints on the prefixes

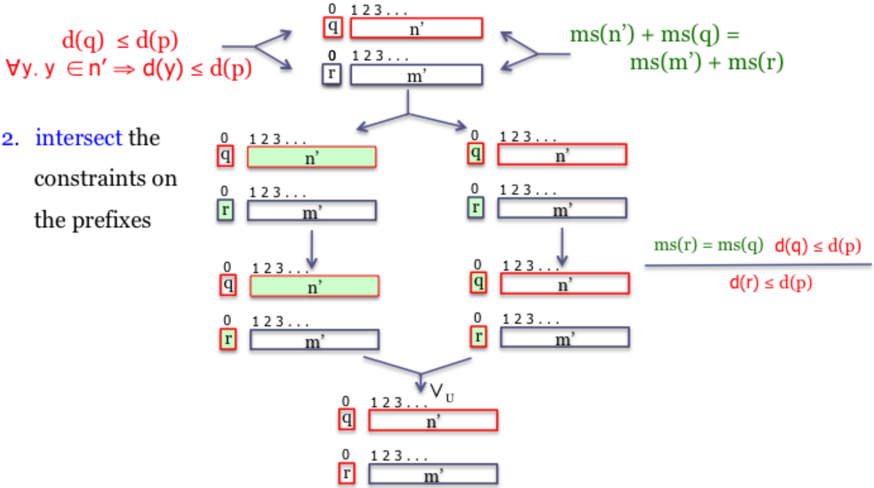


2. intersect the constraints on the prefixes



Strengthen Procedure: Example

2. intersect the constraints on the prefixes



class	fun	nesting (loop,rec)	$\mathcal{A}_{\mathbb{M}}$ t (s)	$\mathcal{A}_{\mathbb{U}}$ $\mathcal{P}$	t (s)	Examples of summaries synthesized
	create	$(0,-)$	<1	$P_=, P_1$	<1	
	addfst	–	<1	$P_=$	<1	
sll	*addlst*	$(0,1)$	<1	$P_=$	<1	$\rho^{\#}_{\mathbb{U}}(create(\&x,\ell))$: $\mathtt{hd}(x)=0 \wedge \mathtt{len}(x)=\ell \wedge \forall y \in \mathtt{tl}(x) \Rightarrow x[y]=0$
	delfst	–	<1	$P_=$	<1	
	dellst	$(0,1)$	<1	$P_=$	<1	
	init(v)	$(0,1)$	<1	$P_=, P_1$	<1	$\rho^{\#}_{\mathbb{U}}(init(v,x))$: $\mathtt{len}(x^0)=\mathtt{len}(x) \wedge \mathtt{hd}(x)=v \wedge \forall y \in \mathtt{tl}(x).\ x[y]=v$
map	*initSeq*	$(0,1)$	<1	$P_=, P_1$	<1	$\rho^{\#}_{\mathbb{U}}(add(v,x))$: $\mathtt{len}(x^0)=\mathtt{len}(x) \wedge \mathtt{hd}(x)=\mathtt{hd}(x^0)+v \wedge$
	add(v)	$(0,1)$	<1	$P_=$	<1	$\forall y_1 \in \mathtt{tl}(x), y_2 \in \mathtt{tl}(x^0).\ y_1=y_2 \Rightarrow x[y_1]=x^0[y_2]+v$
map2	*add*(v)	$(0,1)$	<1	$P_=$	<1	$\rho^{\#}_{\mathbb{U}}(add(v,x,z))$: $\mathtt{len}(x^0)=\mathtt{len}(x) \wedge \mathtt{len}(z^0)=\mathtt{len}(z) \wedge eq(x,x^0) \wedge$
	copy	$(0,1)$	<1	$P_=$	<1	$\forall y_1 \in \mathtt{tl}(x), y_2 \in \mathtt{tl}(z).\ y_1=y_2 \Rightarrow x[y_1]+v=z[y_2]$
	*del*Pred	$(0,1)$	<1	$P_=, P_1$	<1	$\rho^{\#}_{\mathbb{M}}(split(v,x,\&l,\&u))$: $\mathtt{ms}(x)=\mathtt{ms}(x^0)=\mathtt{ms}(l) \cup \mathtt{ms}(u)$
fold	*max*	$(0,1)$	<1	$P_=, P_1$	<1	$\rho^{\#}_{\mathbb{U}}(split(v,x,\&l,\&u))$: $equal(x,x^0) \wedge \mathtt{len}(x)=\mathtt{len}(l)+\mathtt{len}(u) \wedge$
	clone	$(0,1)$	<1	$P_=$	<1	$l[0] \le v \wedge \forall y \in \mathtt{tl}(l) \Rightarrow l[y] \le v \wedge$
	split	$(0,1)$	<1	$P_=, P_1$	<1	$u[0] > v \wedge \forall y \in \mathtt{tl}(u) \Rightarrow u[y] > v$
	equal	$(0,1)$	<1	$P_=$	<1	$\rho^{\#}_{\mathbb{M}}(merge(x,z,\&r))$: $\mathtt{ms}(x) \cup \mathtt{ms}(z)=\mathtt{ms}(r) \wedge \mathtt{ms}(x^0)=\mathtt{ms}(x) \wedge \ldots$
fold2	*concat*	$(0,1)$	<1	$P_=, P_1, P_2$	<3	$\rho^{\#}_{\mathbb{U}}(merge(x,z,\&r))$: $equal(x,x^0) \wedge equal(z,z^0) \wedge sorted(x^0) \wedge sorted(z^0) \wedge$
	merge	$(0,1)$	<1	$P_=, P_1, P_2$	<3	$sorted(r) \wedge \mathtt{len}(x)+\mathtt{len}(z)=\mathtt{len}(r)$
	bubble	$(1,-)$	<1	$P_=, P_1, P_2$	<3	
sort	*insert*	$(1,-)$	<1	$P_=, P_1, P_2$	<3	$\rho^{\#}_{\mathbb{M}}(quicksort(x))$: $\mathtt{ms}(x)=\mathtt{ms}(x^0)=\mathtt{ms}(res)$
	quick	$(-,2)$	<2	$P_=, P_1, P_2$	<4	$\rho^{\#}_{\mathbb{U}}(quicksort(x))$: $equal(x,x^0) \wedge sorted(res)$
	merge	$(-,2)$	<2	$P_=, P_1, P_2$	<4	

Reasoning about Composite Data Structures

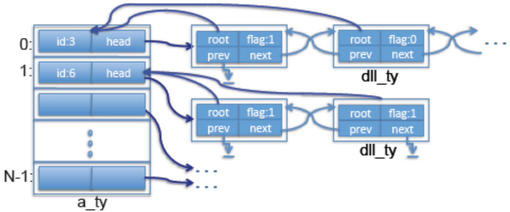
— using a FO Logic Framework —

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

CONCUR'09

Properties of Complex Data Structures

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



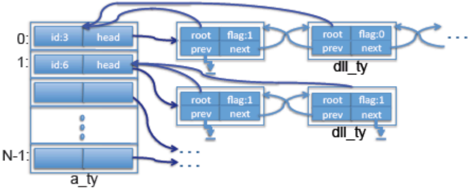
Structure: "the array contains in each cell a reference to an acyclic doubly linked list"

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`"

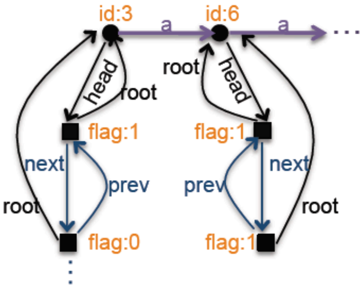
Heaps are represented as labeled directed graphs called heap graphs

```
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

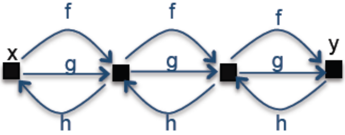
- assume $\mathbb{D}$ the data domain where data fields take values
- assume $\mathsf{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$ a first order logic on $\mathbb{D}$, with operations in $\mathbb{O}$ and predicates in $\mathbb{P}$

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

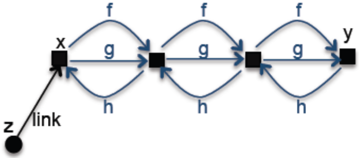
$$\text{gCSL} = \text{FO} \; + \; \text{reachability} \; + \; \begin{array}{c}\text{arithmetical}\\ \text{constraints}\end{array} \; + \; \text{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$$

Reachability Predicates

$$x \xrightarrow{\{f,g,\overline{h}\}} y$$

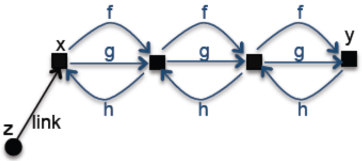


$$x \xrightarrow{\{f,g,\overline{h}\}} y \qquad link(z) = x$$



$$x \xrightarrow{\{f,g,\overline{h}\},l} y$$

$$x \xrightarrow{\{f,g,\overline{h}\}} y \qquad link(z) = x$$



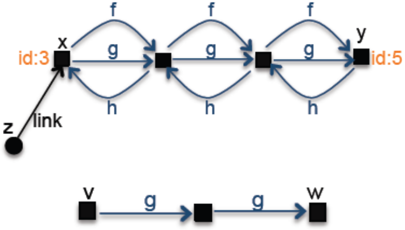
$$id(x) = 3$$

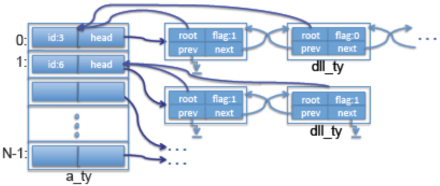
$$\exists c.\ id(x) + id(y) + c \geq 9$$

$$x \xrightarrow{\{f,g,\overline{h}\},l} y \wedge l = 3 \wedge v \xrightarrow{\{g\},l'} w$$

$$l' < l \wedge l + l' \geq 4$$

$$x \xrightarrow{\{f,g,\overline{h}\}} y \qquad link(z) = x$$





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

$$\forall i\ \exists x, y.\ x = \mathtt{head}(a[i]) \wedge x \xrightarrow{\{\mathtt{next},\overline{\mathtt{prev}}\}} y$$

Sizes: "the array is sorted w.r.t. the lengths of lists stored"

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

$$x \xrightarrow{\{\mathtt{next}\},l} \mathtt{null} \wedge x' \xrightarrow{\{\mathtt{next}\},l'} \mathtt{null} \wedge l' \leq l)$$

Data: "the array is sorted w.r.t. the values of the field id"

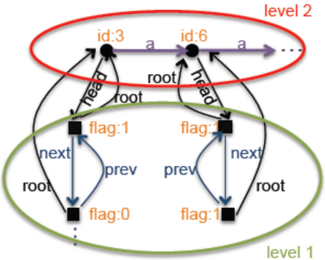
$$\forall i, j.\ i < j \implies \mathtt{id}(a[i]) < \mathtt{id}(a[j])$$

The satisfiability problem of gCSL 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

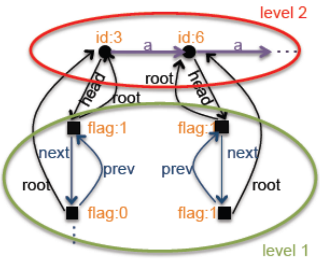
An ordered partition over $\mathcal{RT}$ is a mapping $\sigma : \mathcal{RT} \rightarrow \{1, \ldots, N\}$

- a type $R \in \mathcal{RT}$ is of *level* k iff $\sigma(R) = k$



An ordered partition over $\mathcal{RT}$ is a mapping $\sigma : \mathcal{RT} \to \{1, \ldots, N\}$

- a type $R \in \mathcal{RT}$ 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:

$$\exists^*_{\leq k}\ \forall^*_k\ \ \exists^*_{\leq k-1}\ \forall^*_{k-1}\ \ \ldots\ \ \exists^*_1\ \forall^*_1\ \ \{\exists_d, \forall_d\}^*.\ \phi$$

ϕ is a quantifier-free formula in gCSL

CSL Fragment

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:

$$\exists^*_{\leq k} \ \forall^*_k \ \ \exists^*_{\leq k-1} \ \forall^*_{k-1} \ \ldots \ \exists^*_1 \ \forall^*_1 \ \ \{\exists_d, \forall_d\}^* . \ \phi$$

ϕ 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 $j = j'$

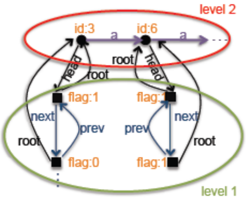
  YES $\quad \forall j, j' . \ j < j' \Rightarrow data(a[j]) < data(a[j'])$

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

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

  YES $\exists x, x', l_1 \exists z, z', l_2 . \ x \xrightarrow{\{f\}, l_1} x' \wedge z \xrightarrow{\{f\}, l_2} z' \wedge l_1 \geq 4 \wedge l_2 \geq 0$

  NO $\exists x, x', l_1 \exists z, z', l_2 . \ x \xrightarrow{\{f\}, l_1} x' \wedge z \xrightarrow{\{f\}, l_2} z' \wedge l_1 + l_2 \geq 0$



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

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

Sizes: "the array is sorted w.r.t. the lengths of lists stored"

$$\forall j, j'.\ j < j' \Longrightarrow \exists x, x', l, l'.\ (x = \mathtt{head}(a[j]) \wedge x' = \mathtt{head}(a[j']) \wedge$$

$$x \xrightarrow{\{\mathtt{next}\}, l} \mathtt{null} \wedge x' \xrightarrow{\{\mathtt{next}\}, l'} \mathtt{null} \wedge l' \leq l)$$

Data: "the array is sorted w.r.t. the values of the field id"

$$\forall i, j.\ i < j \Longrightarrow \mathtt{id}(a[i]) < \mathtt{id}(a[j])$$

Theorem

The satisfiability of CSL is decidable if the satisfiability of the underlying first order logic $\mathrm{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$ *is decidable*

Let

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

1. compute φ_{k-1} equi-satisfiable to φ_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 φ to the satisfiability of a formula in $\mathrm{FO}(\mathbb{D}, \mathbb{O}, \mathbb{P})$

Theorem

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

Let

$$\varphi = \exists_1^* \mathbf{x} \ \forall_1^* \mathbf{y} \ \ \{\exists_d, \forall_d\}^*. \ \phi''$$

1. compute the set of small models for the reachability and size constraints

2. for each small model, build a $FO(\mathbb{D}, \mathbb{O}, \mathbb{P})$ formula ψ

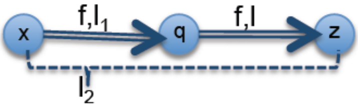
   If one of ψ is satisfiable then φ is satisfiable.

Computing Small Models

$$\varphi = \exists x, q, z \;.\; x \xrightarrow{\{f\}} q \wedge x \xrightarrow{\{f\}} z \;\wedge\; q \neq z$$

- φ has two small models of size three

$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z \ \wedge\ q \neq z$$
$$\wedge\ l_1 + l_2 \geq 8$$



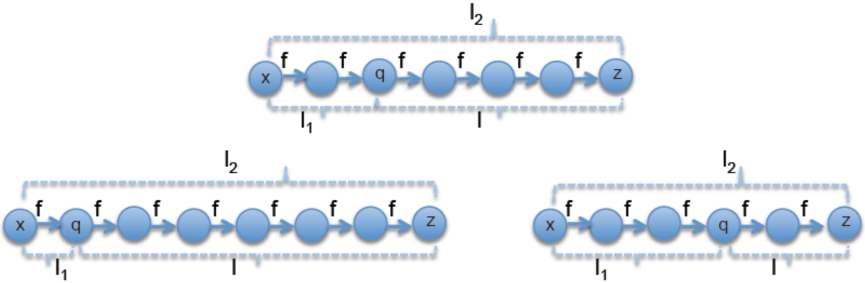
$$l_1 + l = l_2 \wedge l_1 + l_2 \geq 8$$

$$\varphi = \exists x, q, z \; \exists l_1, l_2.\; x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z \;\wedge\; q \neq z$$
$$\wedge\; l_1 + l_2 \geq 8$$

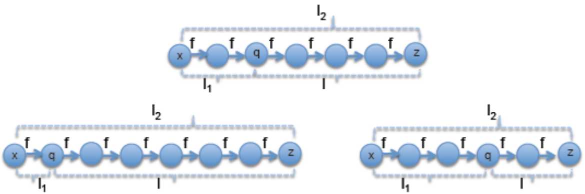
- minimal solutions (l_1, l_2, l) for $l_1 + l = l_2 \wedge l_1 + l_2 \geq 8$

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

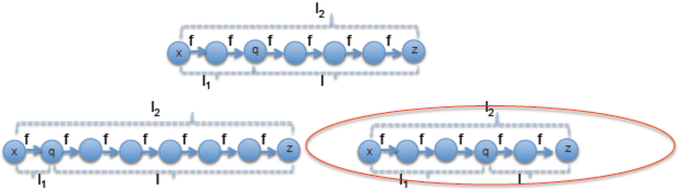
- small models for φ



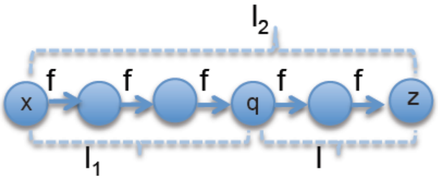
$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\},l_1} q \wedge x \xrightarrow{\{f\},l_2} z\ \wedge\ q \neq z$$
$$\wedge l_1 + l_2 \geq 8$$
$$\wedge\ g(x) = 0 \wedge g(q) = 2$$
$$\wedge\ \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$



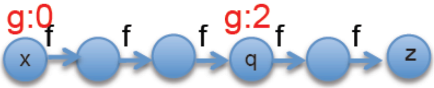
$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\},l_1} q \land x \xrightarrow{\{f\},l_2} z \ \land\ q \neq z$$
$$\land l_1 + l_2 \geq 8$$
$$\land\ g(x) = 0 \land g(q) = 2$$
$$\land\ \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$



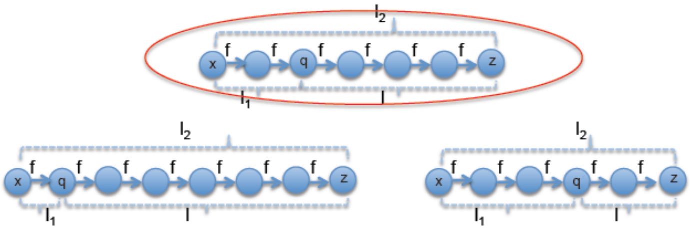
$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\},l_1} q \wedge x \xrightarrow{\{f\},l_2} z\ \wedge\ q \neq z$$
$$\wedge l_1 + l_2 \geq 8$$
$$\wedge\, g(x) = 0 \wedge g(q) = 2$$
$$\wedge\, \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$



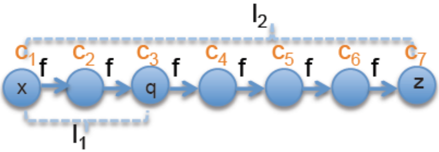
$$\varphi = \exists x, q, z\ \exists l_1, l_2.\ x \xrightarrow{\{f\}, l_1} q \wedge x \xrightarrow{\{f\}, l_2} z\ \wedge\ q \neq z$$
$$\wedge l_1 + l_2 \geq 8$$
$$\wedge\ g(x) = 0 \wedge g(q) = 2$$
$$\wedge\ \forall y, y'.\ (y \xrightarrow{\{f\}} y' \Rightarrow g(y) < g(y'))$$



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



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



$$\psi_1 = \exists c_1, c_2, c_3, c_4, c_5, c_5, c_6, c_7.\ \bigwedge_{i \neq j} c_i \neq c_j$$
$$true \wedge true\ \wedge\ true\ \wedge true$$
$$\wedge\ c_1 = 0 \wedge c_3 = 2$$
$$\wedge\ \bigwedge_{1 \leq i < j \leq 7} c_i < c_j$$

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

$$\psi_1 = \exists c_1, c_2, c_3, c_4, c_5, c_5, c_6, c_7.\ \bigwedge_{i \neq j} c_i \neq c_j$$
$$\mathit{true} \wedge \mathit{true}\ \wedge\ \mathit{true}\ \wedge \mathit{true}$$
$$\wedge\ c_1 = 0 \wedge c_3 = 2$$
$$\wedge\ \bigwedge_{1 \leq i < j \leq 7} c_i < c_j$$

1. choose a small model for the reachability and size constraints; if there are no models then φ is unsatisfiable
2. build a FO($\mathbb{D}, \mathbb{O}, \mathbb{P}$) formula ψ for the selected small model
3. check the satisfiability of ψ

Remark

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

Theorem

If the satisfiability of the underlying first order logic $FO(\mathbb{D}, \mathbb{O}, \mathbb{P})$ *is decidable, then the satisfiability of CSL is decidable*

Theorem

For any basic statement S *and any CSL formula* φ*, we can compute in polynomial time a formula* $\mathtt{post}(S, \varphi)$ *describing the strongest post-condition of* φ *by* S*.*

Outline

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:

- 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:

$$\begin{aligned}
\mathtt{dll}(E, L, P, F) &\triangleq (E = F \land L = P \land emp) \lor \big(E \neq F \land L \neq P \land \\
&\quad \exists X.\ E \mapsto \{(\mathtt{nxt}, X), (\mathtt{prv}, P)\} * \mathtt{dll}(X, L, E, F)\ \big) && (1)\\
\mathtt{btree}(E) &\triangleq (E = \boxtimes \land emp) \lor \big(E \neq \boxtimes \land \\
&\quad \exists X, Y.\ E \mapsto \{(\mathtt{lson}, X), (\mathtt{rson}, Y)\} * \mathtt{btree}(X) * \mathtt{btree}(Y)\big) && (2)\\
\mathtt{tll}(R, P, E, F) &\triangleq (R = E \land R \mapsto \{(\mathtt{lson}, \boxtimes), (\mathtt{rson}, \boxtimes), (\mathtt{parent}, P), (\mathtt{nxt}, F)\})\ \lor && (3)\\
&\quad \big(R \neq E \land \exists X, Y, Z.\ \ R \mapsto \{(\mathtt{lson}, X), (\mathtt{rson}, Y), (\mathtt{parent}, P), (\mathtt{nxt}, Z)\} * \\
&\qquad \mathtt{tll}(X, R, E, Z) * \mathtt{tll}(Y, R, Z, F)\big)
\end{aligned}$$

The fragment allowing these specifications has good theoretical properties:

- decidability of satisfiability [Brotherston *et al*, 14]
  $\longrightarrow$ by reduction boolean equations
- decidability of the entailment [Iosif *et al*, 13]
  $\longrightarrow$ by reduction to MSO on graphs with bounded width

Separation Logic Solvers

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]

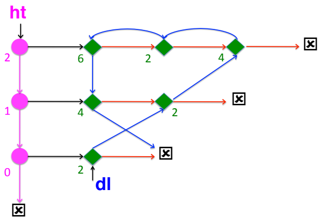
Follow them on SL-COMP competition:

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

`www.liafa.univ-paris-diderot.fr/slcomp`

- 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]



$$\texttt{nll}_\beta(h, \boxtimes, \boxtimes) \circledast \texttt{ls}_\delta(dl, \boxtimes) \wedge \beta(\lozenge) = \delta(\lozenge)$$

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.