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

Static Analysis

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 

Some Industrial Success Stories

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

Static Analysis Interface

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

Main Ingredient: Property-Preserving Abstraction

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.

Challenge

Find abstractions with

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

Static Analysis Example: Null Pointer Aliasing

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:

$$x = null$$

$$x \neq null$$

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;
}
```

Static Analysis Example: Null Pointer Aliasing

```
list* search(list* h, int key) {
                h = null  ⋁  h ≠ null
  list* it = h;

  bool b = false;


  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) {

    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) {
```
$h = null \quad \bigvee \quad h \neq null$
```
  list* it = h;
```
$h = null \ \wedge \ it = null \quad \bigvee \quad h \neq null \ \wedge \ it \neq null$
```
  bool b = false;
```
$h = null \ \wedge \ it = null \quad \bigvee \quad h \neq null \ \wedge \ it \neq null$
$\bigvee \quad h \neq null \ \wedge \ it = null$
```
  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;
```
$h \neq null \ \wedge \ it \neq null \quad \bigvee \quad h \neq null \ \wedge \ it = null$
```
  }


  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;
```
$h = null \ \wedge \ it = null \quad \bigvee \quad h \neq null \ \wedge \ it \neq null$
$\bigvee \quad h \neq null \ \wedge \ it = null$
```
  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;
```
$h \neq null \ \wedge \ it \neq null \quad \bigvee \quad h \neq null \ \wedge \ it = null$
```
  }
```
$h = null \ \wedge \ it = null \quad \bigvee \quad h \neq null \ \wedge \ it \neq null$
$\bigvee \quad h \neq null \ \wedge \ it = null$
```
  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

$\longrightarrow$ shape analysis [Larus&Hilfinger,88, Horwitz&al,89,...]

Motivation for Shape Analysis

```
/* @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: $\mathrm{list}(l, \mathrm{null})$, $\mathrm{list}(r, \mathrm{null})$, $r \xrightarrow{next}\!\!\!\!\!\rightarrow l$, ...

Motivation for Shape Analysis

```
/* @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: $\mathrm{list}(\mathrm{h}, \mathrm{null})$ of data $0..\mathrm{d}$

Abstractions Techniques for Shape Analysis

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... ☹

This lecture

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.

Basic Types

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

Record types

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

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

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

Class of Programs: Variables and Procedures

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;
start_P:       // sequence of statements
                ...; v = ...; ...
                return v;
end_P:        }
```

Class of Programs: Expressions and Statements

Boolean and numeric expressions

Fixed evaluation order of arguments; no side effects.

$$be ::= bcst \mid bv \mid r(\overrightarrow{de}) \mid rv_1 == rv_2 \mid !be \mid be \wedge be \mid be \vee be$$

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

Reference expressions

No arithmetics on references!

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

Statements

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

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

Example Revisited

```
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 $\quad \texttt{Mem} \triangleq \texttt{Stacks} \times \texttt{Heaps} \ni \texttt{m}$

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.*

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

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

Then, the heap formal model is:

$$\texttt{H} \in \texttt{Heaps} \triangleq [(\mathbb{L} \times \mathbf{FS}) \rightarrowtail (\mathbb{D} \cup \mathbb{L})]$$

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

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	$\mathtt{Stacks} \triangleq [(\mathbf{CP} \times \mathbf{P} \times (\mathbf{DV} \rightarrow \mathbb{D} \cup \mathbf{RV} \rightarrow \mathbb{L}))^*] \ni \mathtt{S}$
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

$$\mathtt{C} \vdash \mathit{stmt} \rightsquigarrow \mathtt{C}'$$
$$\mathtt{m} \vdash \mathit{astmt} \rightsquigarrow \mathtt{m}' \mid \mathtt{merr}$$
$$\mathtt{m} \vdash \mathit{be} \rightsquigarrow \mathtt{b} \mid \mathtt{merr}$$
$$\mathtt{m} \vdash \mathit{de} \rightsquigarrow \mathtt{c} \mid \mathtt{merr}$$
$$\mathtt{m} \vdash \mathit{re} \rightsquigarrow \mathtt{a} \mid \mathtt{merr}$$

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

Natural Semantics: Rules (Some)

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

$$\frac{\exists i.\ \mathtt{m} \vdash \mathtt{de}_i \rightsquigarrow \mathtt{merr}}{\mathtt{m} \vdash \mathtt{r}(\mathtt{de}_1, \ldots, \mathtt{de}_n) \rightsquigarrow \mathtt{merr}}$$

Natural Semantics: Rules (Some)

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

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

Natural Semantics: Rules (Some)

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

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

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

Natural Semantics: Rules (Some)

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

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

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

No garbage detection

Natural Semantics: Rules (Some)

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

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

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

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

Natural Semantics: Rules (Some)

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

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

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

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

Infinite heap

By default initialisation

Natural Semantics: Rules (Some)

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

See procedure call in the second part!

$$\frac{m(rv) = \quad\quad = c}{m \vdash rv{\rightarrow}df \rightsquigarrow c} \qquad \frac{m(rv) = \boxtimes}{m \vdash rv{\rightarrow}df \rightsquigarrow \texttt{merr}}$$

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

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

Infinite heap

By default initialisation

Definition

An inter-procedural control flow graph (ICFG) over a set of operations $\mathtt{Op}$ is a tuple $\langle V, \mathtt{Op}, \rightarrow, \mathit{start}, \mathit{end} \rangle$ 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*,
- $\rightarrow \in V \times \mathtt{Op} \times V$ is a finite set of *edges*.

For our class of program

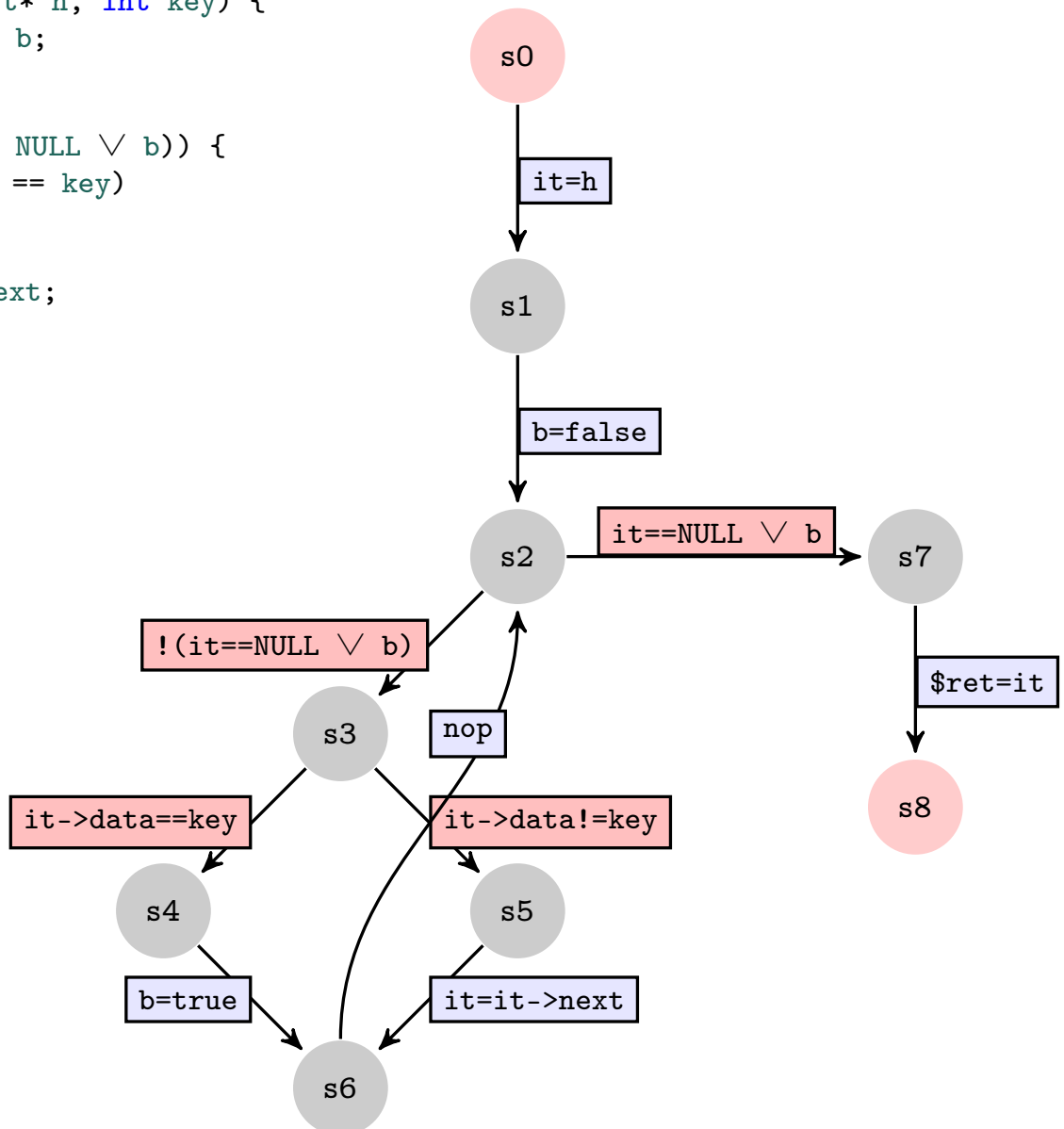
$$\mathtt{Op} \ni \mathrm{op} \ ::= \ \mathrm{be} \mid \mathrm{astmt} \mid \mathtt{call}\ v = \mathrm{P}(\ldots) \mid \mathtt{return}\ v = \mathrm{P}(\ldots)$$

where (recall)

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

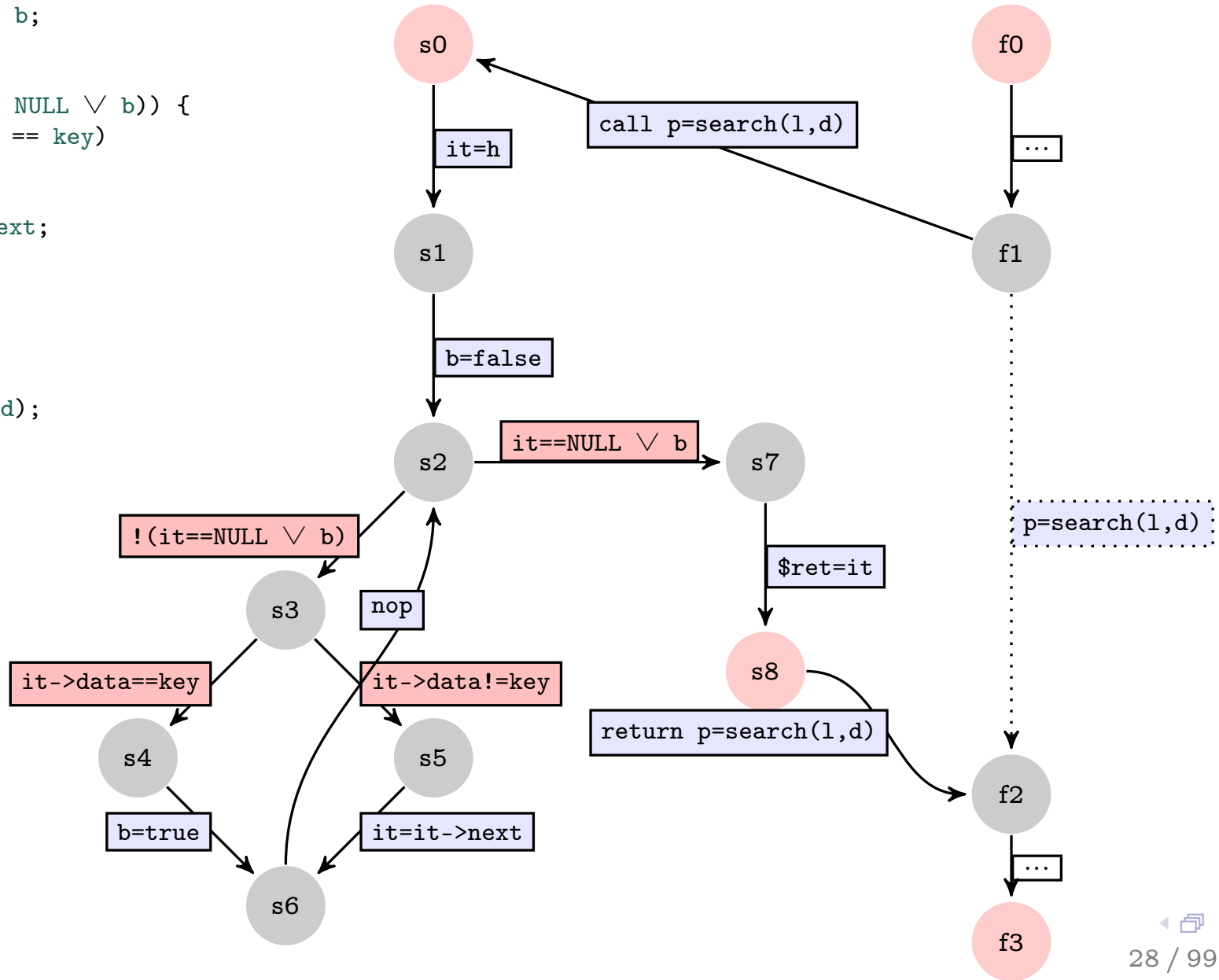
ICFG for search

```
    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: }
```



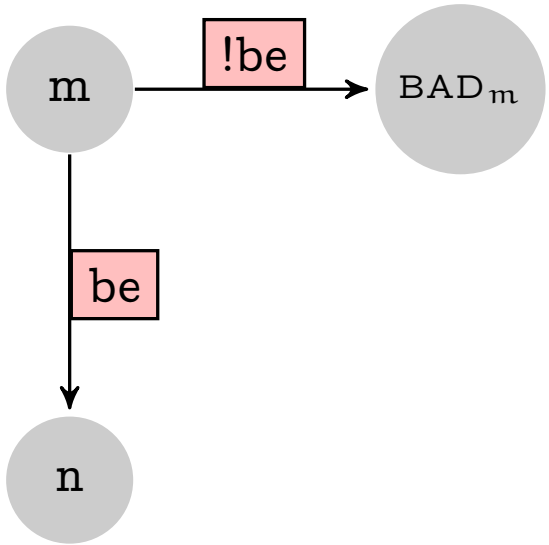
ICFG for search and foo

```
    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: }
```

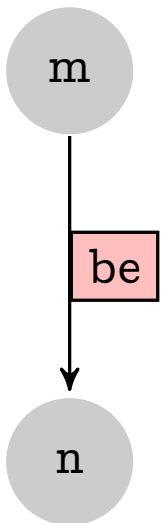


ICFG for User Assertions

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




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



ICFG and Labeled Transition Systems

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 $\langle C, \mathrm{Init}, \mathrm{Out}, \Sigma, \rightarrow \rangle$ where:

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

LTS is an infinite, semantic object.

Control Paths, Execution Paths, Runs

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:

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

Outline

Reformulate Our Goal

Goal

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

The exact set of reachable configurations is:

$$\begin{aligned} \mathrm{Post}^* &= \bigcup_{\sigma \,:\, \mathrm{run}} \{(q, m) \mid (q, m) \text{ occurs in } \sigma\} \\ &= \bigcup_{q_{\mathrm{Init}} \xrightarrow{op_0} \ldots \xrightarrow{op_k} q} \{q\} \times \left(\mathtt{post}_{op_k} \circ \ldots \circ \mathtt{post}_{op_0}\right)(m_{\mathrm{Init}}) \end{aligned}$$

where

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

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

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

Reformulate Our Goal

Goal

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

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

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

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

and

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

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

Reformulate our Goal

$\overline{\text{Post}}^*$ 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:

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

Reformulate our Goal

$\overrightarrow{\overline{\text{Post}}^*}$ 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… system of in-equations over ICFG n…

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

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

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.

Knaster-Tarski Fixpoint Theorem

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 $f(x) = x$.

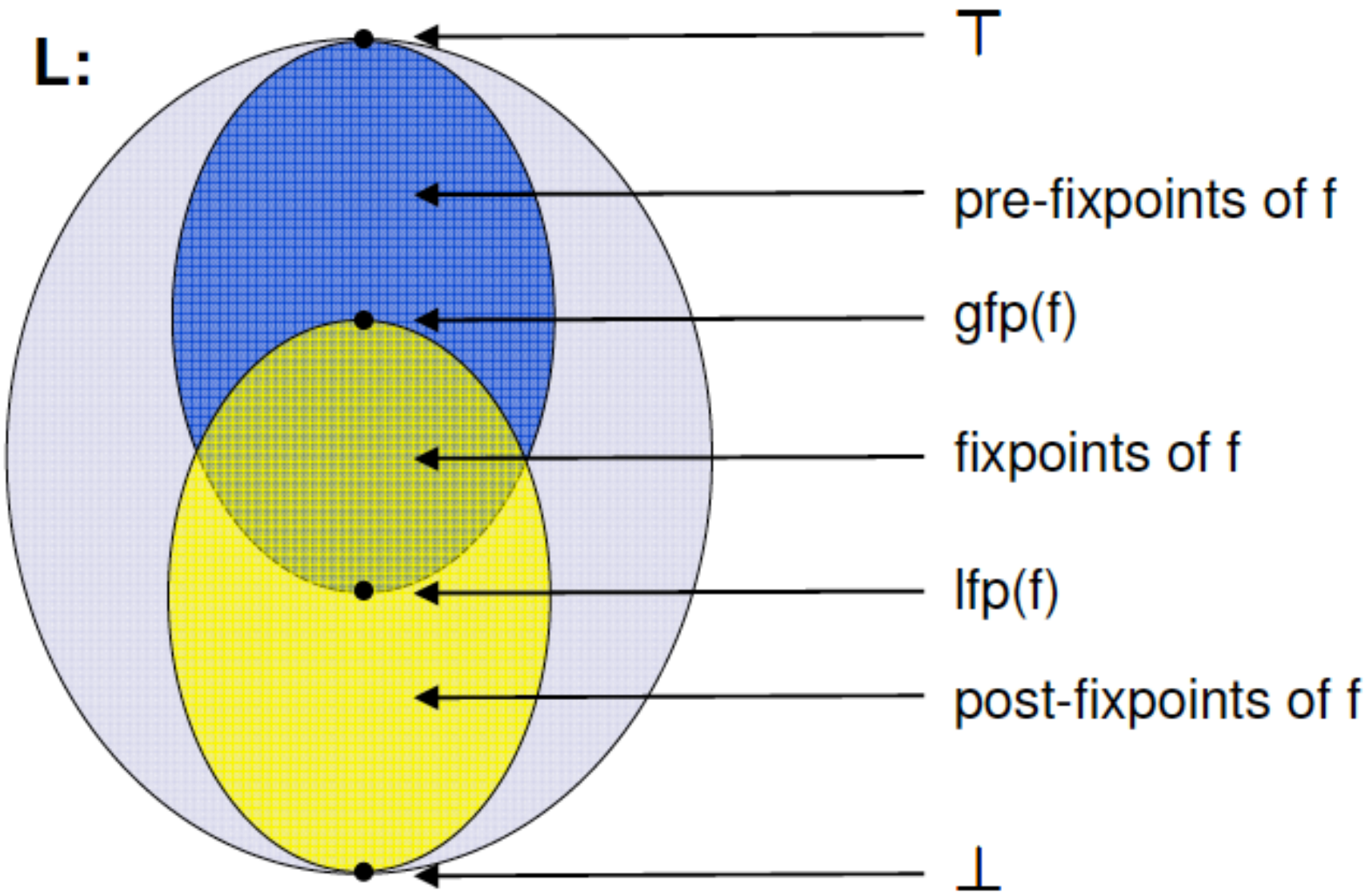
Theorem Knaster-Tarski

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:

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

Lattice of Fixpoints



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:

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

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 $\texttt{lfp}(f)$ 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 $\texttt{lfp}(f)$.

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

```
1:  W = ∅;
2:  for (all vertex q) { P[q] = ⊥; W = Add(W,q); }
3:  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 */
4:  while (W != ∅) {
5:  q = Extract(W);
6:  for (all edge (q,op,r)) {
7:    t = post_op (P[q]);
8:    if (! (t ⊑ P[r])) {
9:      P[r] = P[r] ⊔ t;
10:     W = Add(W,r);
11: } }
12: } /* ∀q. P[q] ⊑ MFP→(q) ∧ P solution ⟹ P = MFP→ */
```

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.

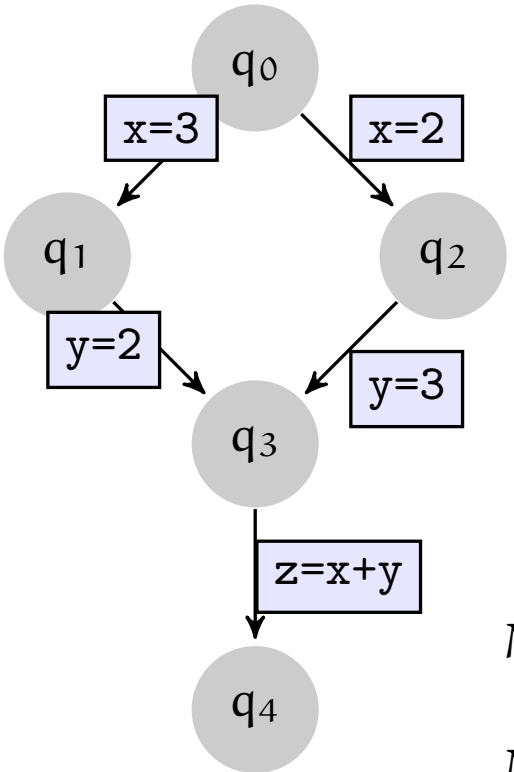
Otherwise, change computation at line 9 to obtain convergence of $(P^i[r])_{i \in \mathbb{N}}$ for any r, *e.g.* widening (see later).

Variants

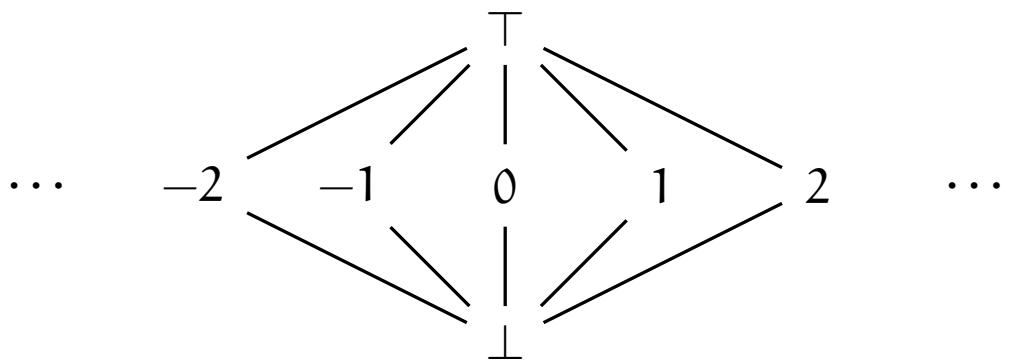
Different iteration strategies are obtained by changing the selection of the visited vertices (`Extract`) and edges (line 6).

Precision of MFP

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



$(L, \sqsubseteq)$:



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

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

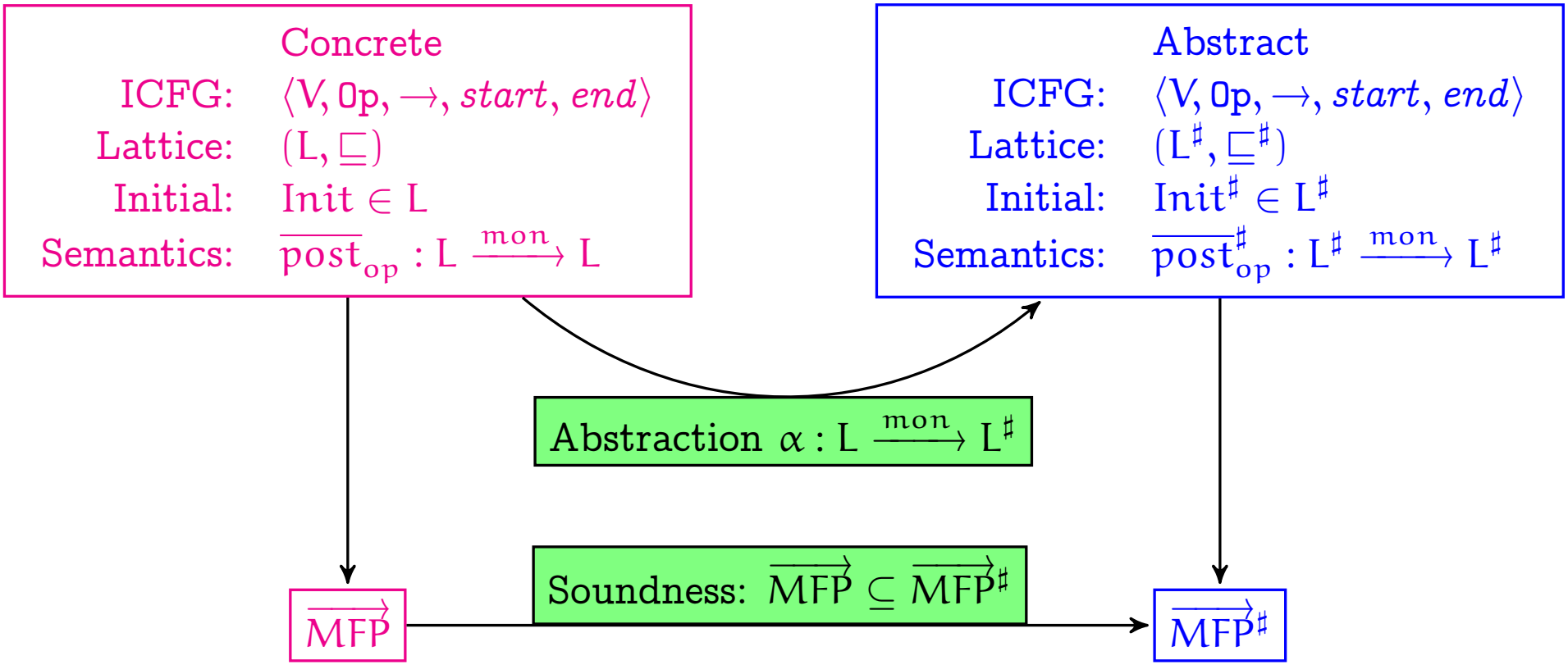
$$f \sqsubseteq g \quad \text{if} \quad \forall x \in S.\ f(x) \sqsubseteq g(x)$$

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

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

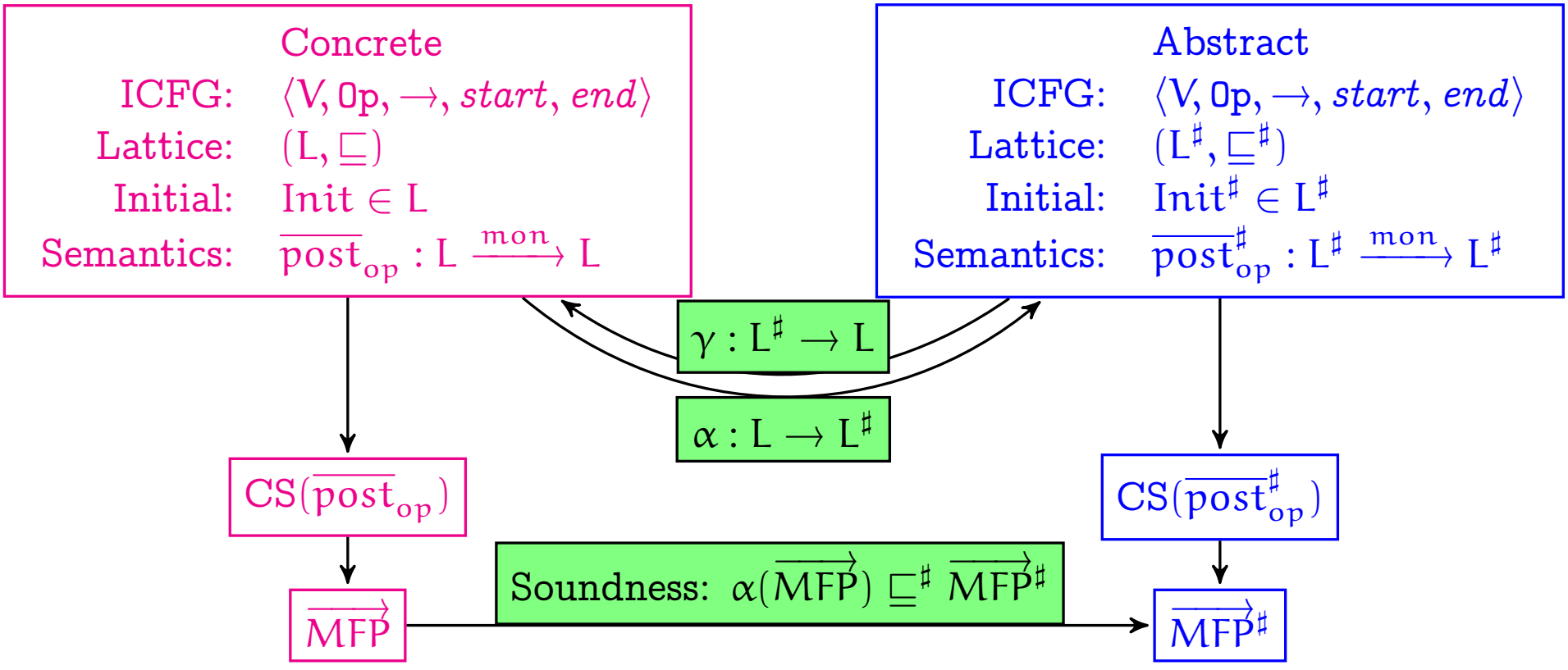
If S is finite and $(L, \sqsubseteq)$ satisfies a.c.c. then $(S \to L, \sqsubseteq)$ satisfies a.c.c.

Abstraction Principle



How to systematically ensure the correctness of this principle?

$\longrightarrow$ Abstract Interpretation [Cousot&Cousot,79].



Conditions

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

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 $y^\sharp \in L^\sharp$:

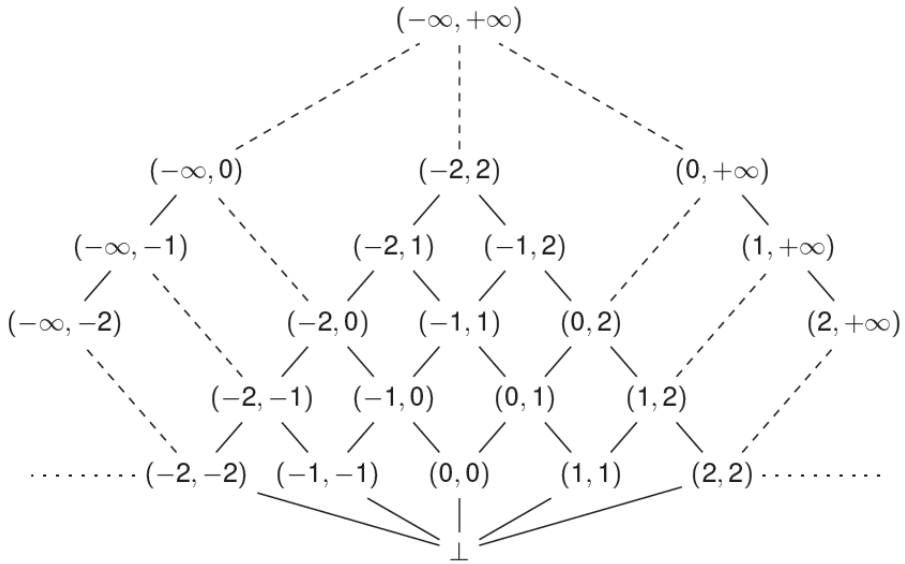
$$\alpha(x) \sqsubseteq^\sharp y^\sharp \quad \text{iff} \quad x \sqsubseteq \gamma(y^\sharp)$$

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.* $\sqsupseteq x$.

Galois Connection: Interval Abstraction

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



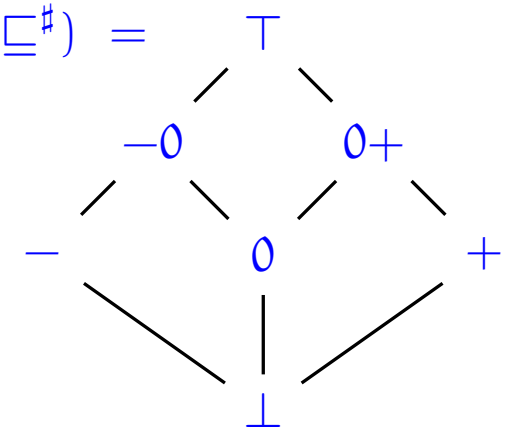
$$S \subset \mathbb{Z} \quad \xrightarrow{\alpha} \quad (\inf(S), \sup(S)) \qquad \textit{e.g.} \qquad \{-1, 2\} \xrightarrow{\alpha} (-1, 2)$$

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

Galois Connection: Sign Abstraction

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

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



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

Galois Connection: Characterisation

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 $f : L \to L$, let consider a monotonic function $g^\sharp : L^\sharp \to L^\sharp$ 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 $f : L \to L$ and any monotonic sound approximation of f, $g^\sharp : L^\sharp \to L^\sharp$ then

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

Definition

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

Example: Abstract Transformer for Intervals

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

The concrete transformer for $\mathrm{op} \equiv (z \geq 0)$ 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 $\mathrm{op} \equiv (z \geq 0)$ 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 $(\mathcal{P}(\mathbb{Z}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathrm{Int}, \subseteq)$

The concrete transformer for $\mathrm{op} \equiv (z \geq 0)$ 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 $g^{\sharp}_{z\geqslant 0} \sqsubseteq^{\sharp} h^{\sharp}_{z\geqslant 0} \sqsubseteq f^{\sharp}_{z\geqslant 0}$.

What happens with `lfp` (recall, used in $\overrightarrow{\mathrm{MFP}}$) 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 $f : L \to L$, the best abstraction of f is the monotonic function $f^\sharp : L^\sharp \to L^\sharp$ defined by:

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

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 $e * e = 0$.

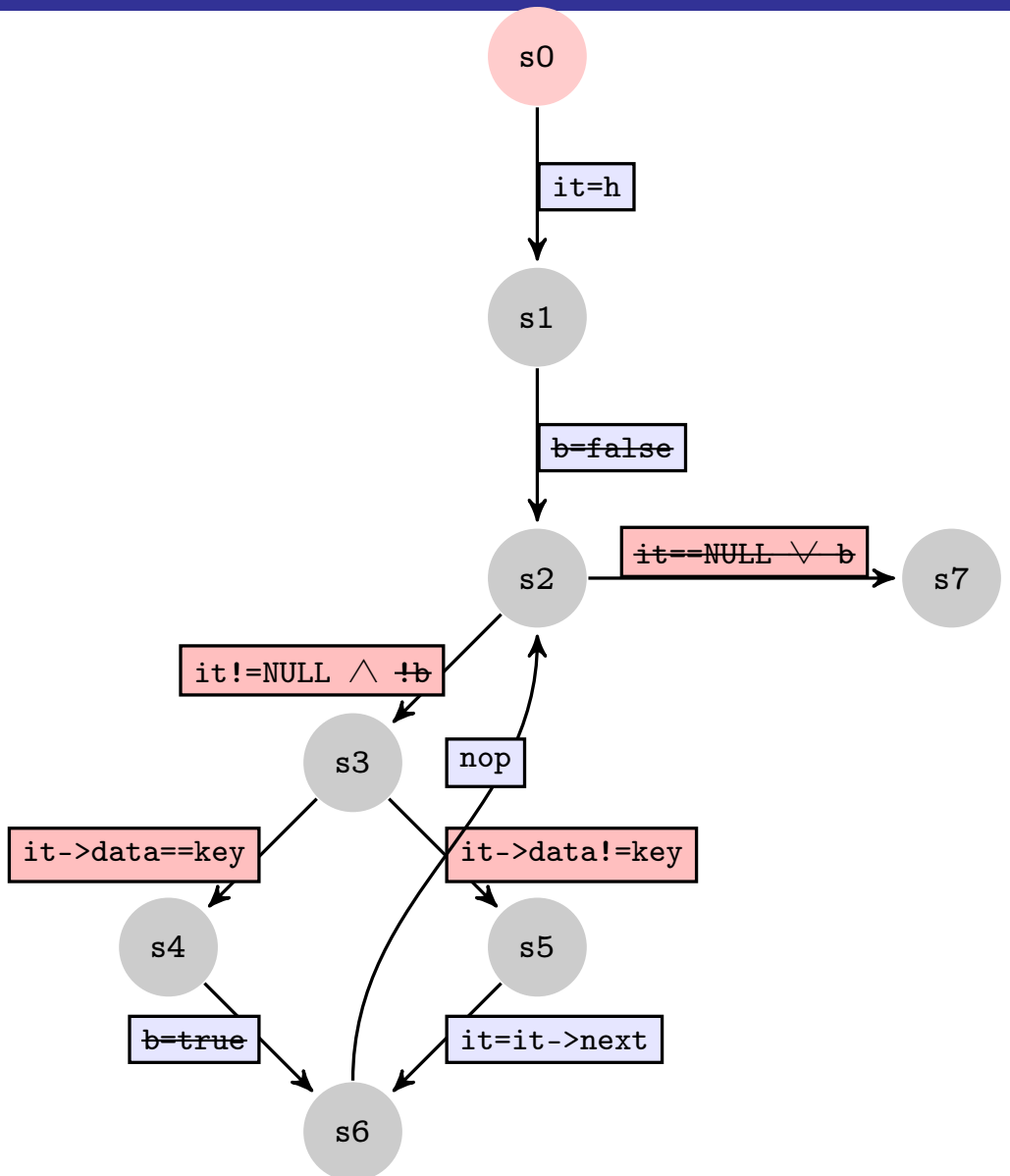
Recipe to Design an Analysis

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 $\sqsubseteq^\sharp, \sqcup^\sharp, \ldots$

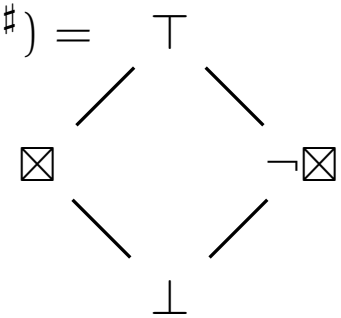
2. Design a sound abstract transformer $g^\sharp$ for each $\mathrm{op} \in \mathrm{OP}$ in ICFG.
   $\longrightarrow$ based on the natural semantics, try to be precise and efficient

3. Compute $\mathtt{lfp}(g^\sharp)$ 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 $g^\sharp$

Example: Analysis of Null Aliasing [CC'77]



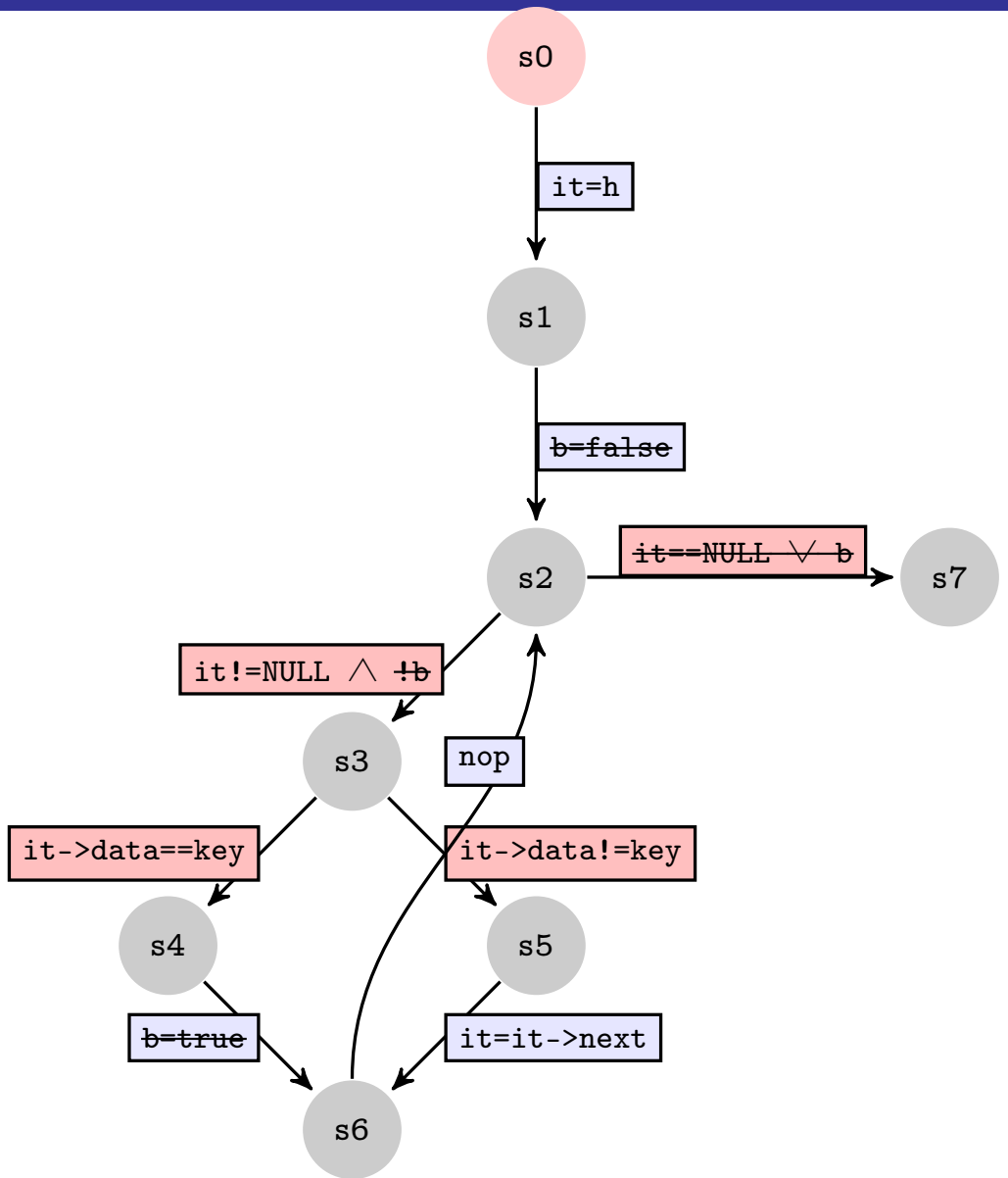
$(L^\sharp, \sqsubseteq^\sharp) =$



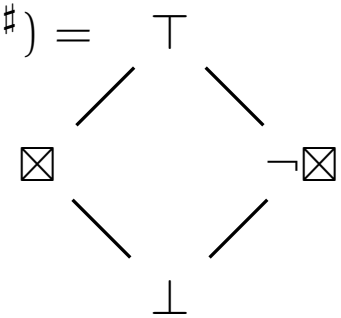
Initially:

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	$\bot$	$\bot$	

Example: Analysis of Null Aliasing [CC'77]



$(\mathrm{L}^\sharp, \sqsubseteq^\sharp) =$

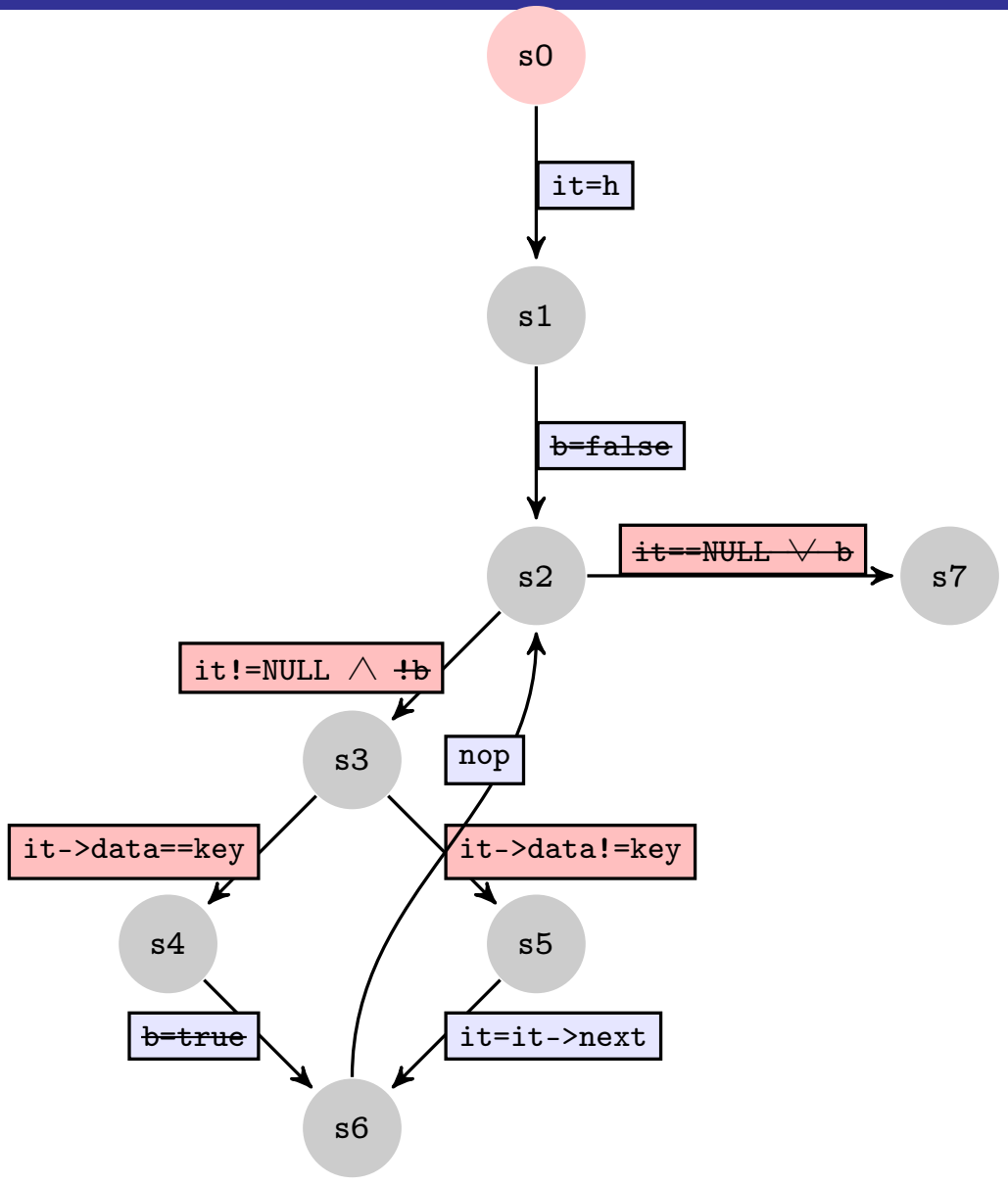


s0 extracted:

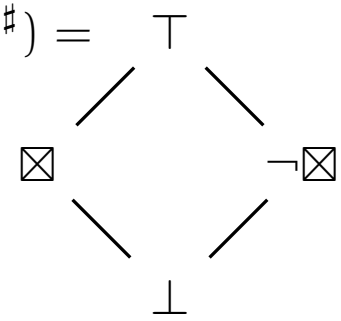
CP	it	h	W
s0	$\bot$	$\top$	
s1	$\top$	$\top$	
s2	$\top$	$\top$	✓
s3	$\bot$	$\bot$	
s4	$\bot$	$\bot$	
s5	$\bot$	$\bot$	
s6	$\bot$	$\bot$	
s7	$\bot$	$\bot$	

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

Example: Analysis of Null Aliasing [CC'77]



$(\mathrm{L}^\sharp, \sqsubseteq^\sharp) =$

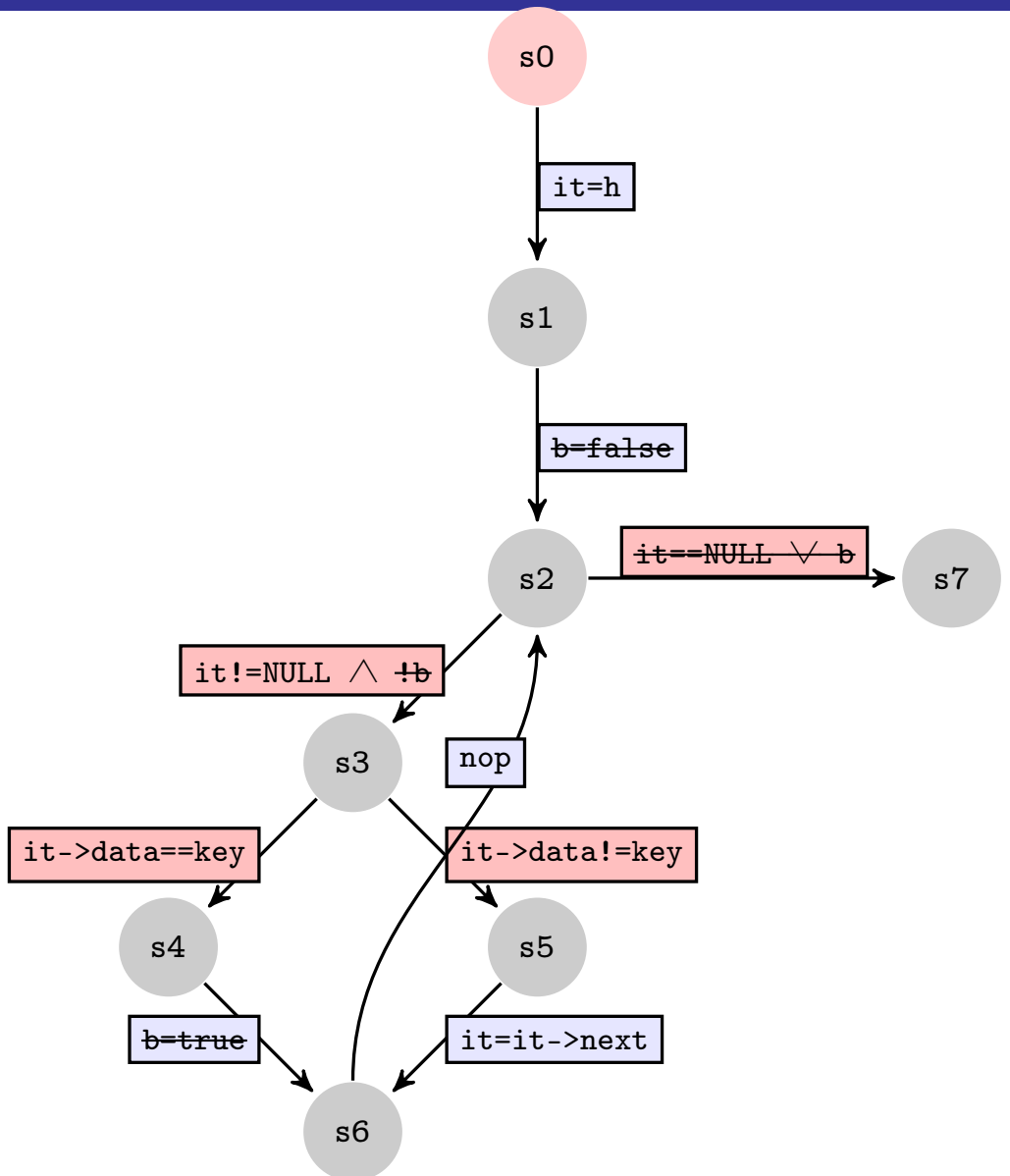


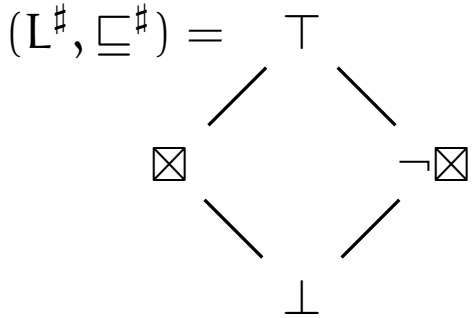
s2 extracted:

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	$\top$	$\top$	✓

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

Example: Analysis of Null Aliasing [CC'77]

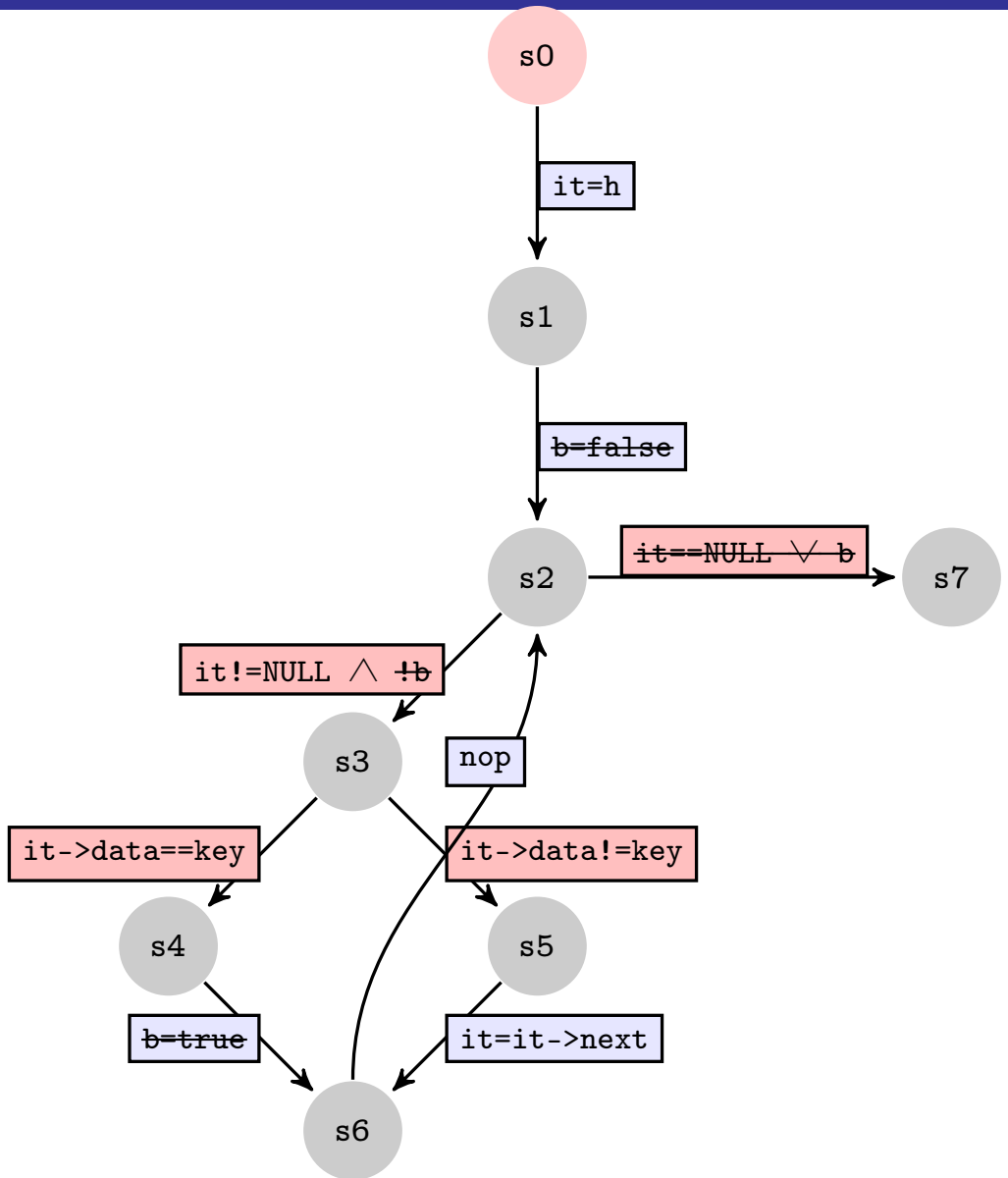




s3 extracted:

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

Example: Analysis of Null Aliasing [CC'77]

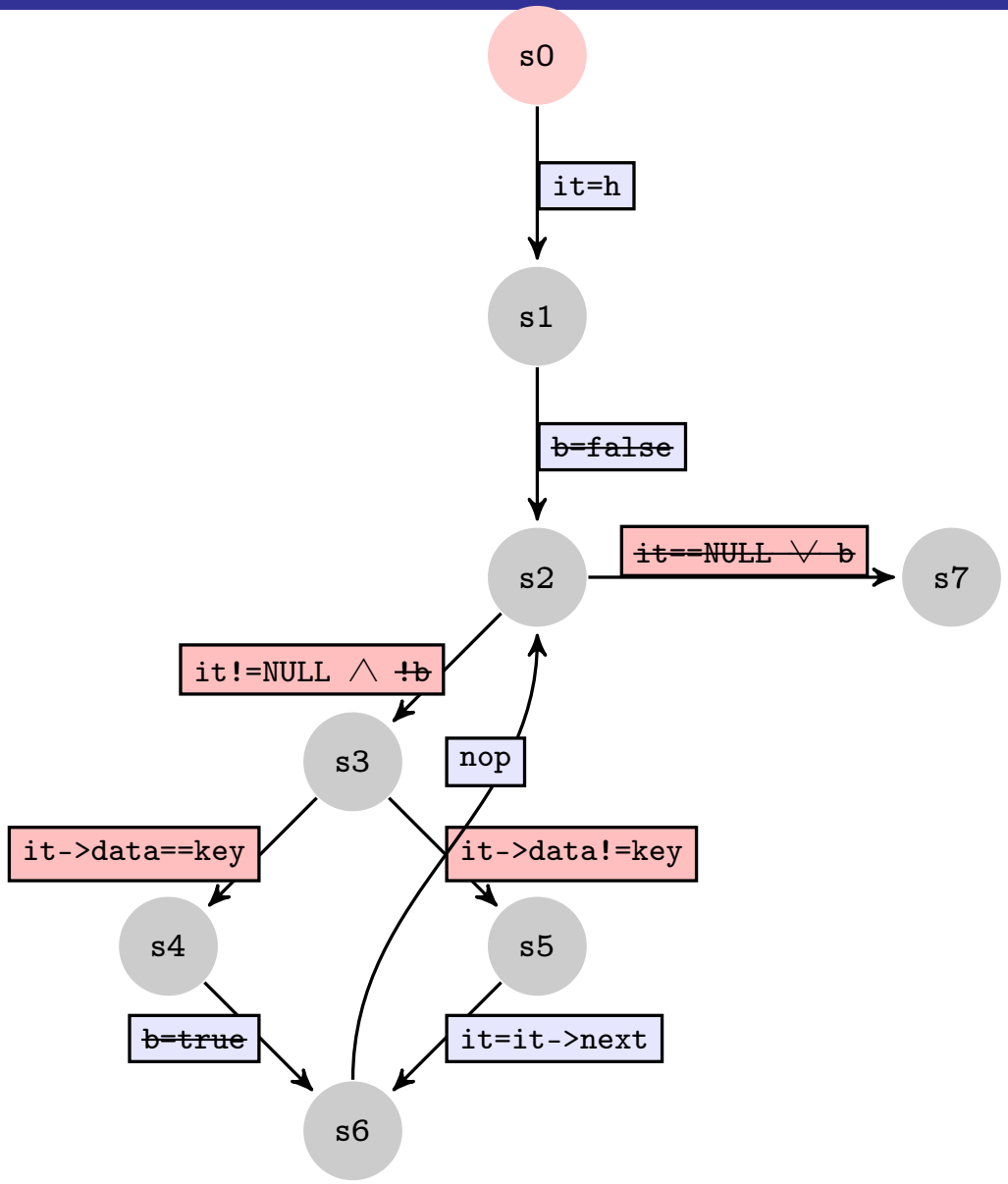


$$(\mathrm{L}^\sharp, \sqsubseteq^\sharp) = \quad \top \quad \boxtimes \quad \neg\boxtimes \quad \bot$$

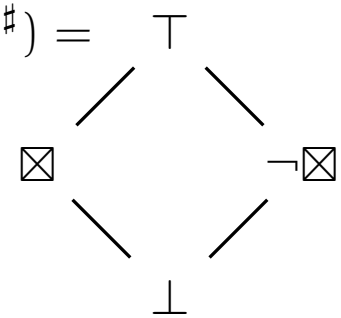
s4 extracted:

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	$\neg\boxtimes$	$\top$	✓
s7	$\top$	$\top$	✓

Example: Analysis of Null Aliasing [CC'77]



$(\mathrm{L}^\sharp, \sqsubseteq^\sharp) =$

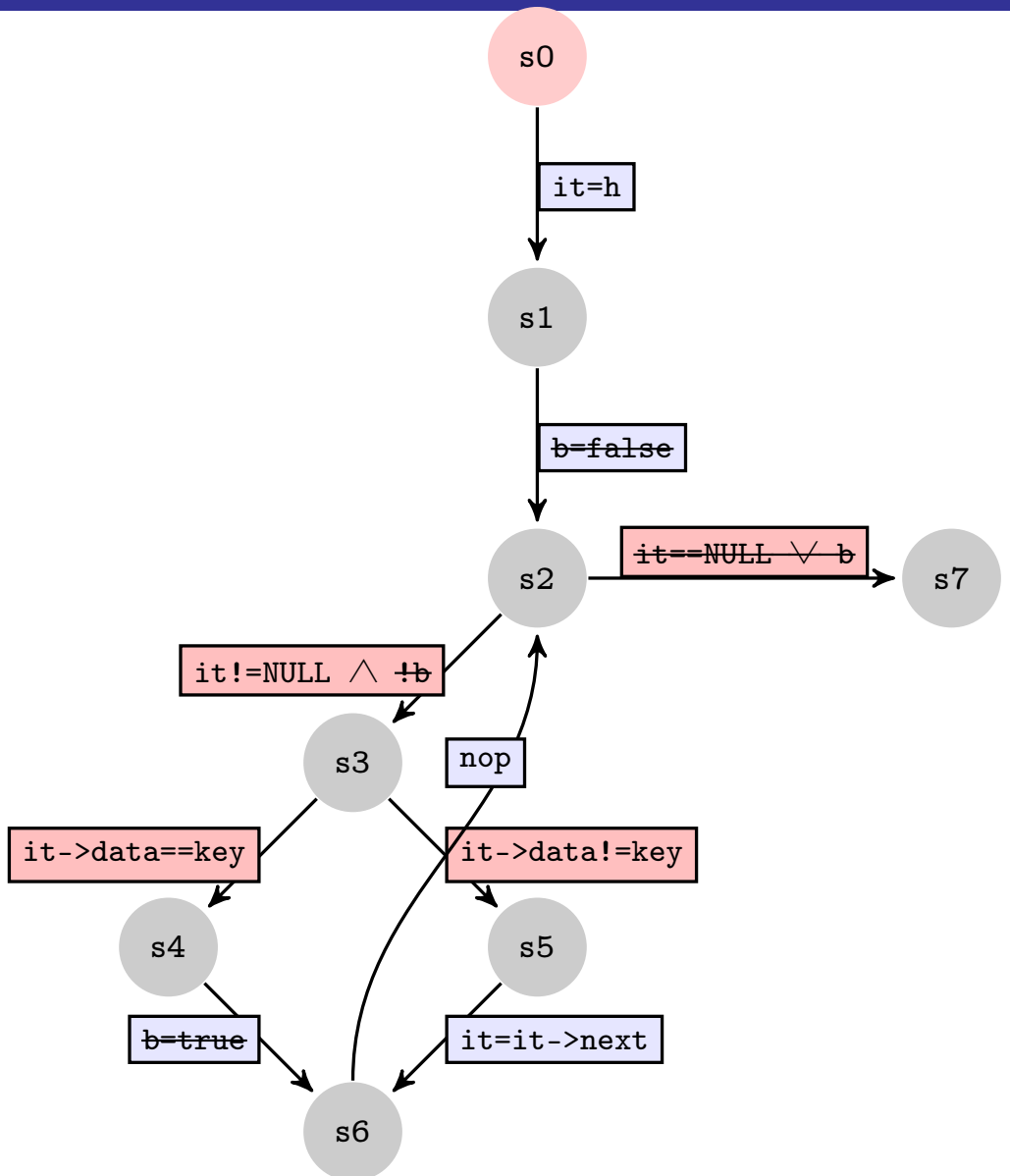


s5 extracted:

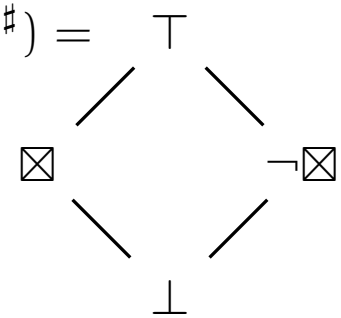
CP	it	h	W
s0	$\bot$	$\top$	
s1	$\top$	$\top$	
s2	$\top$	$\top$	
s3	$\neg\boxtimes$	$\top$	
s4	$\neg\boxtimes$	$\top$	
s5	$\neg\boxtimes$	$\top$	
s6	$\top$	$\top$	✓
s7	$\top$	$\top$	✓

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

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	$\neg\boxtimes$	$\top$	
s4	$\neg\boxtimes$	$\top$	
s5	$\neg\boxtimes$	$\top$	
s6	$\top$	$\top$	
s7	$\top$	$\top$	✓

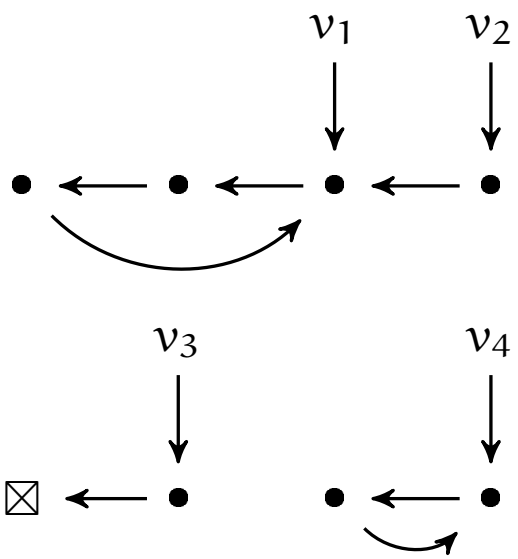
Example: Analysis of Heap Separation [CC'77]

Abstraction Idea

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

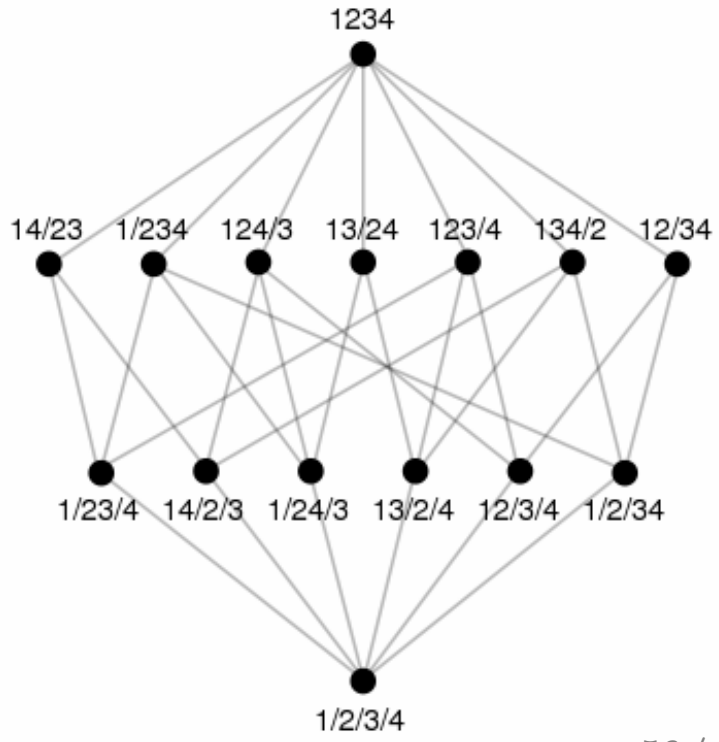
- v_1, v_2 belong to the same partition if $v_1 \xrightarrow{next^*} \cap\, v_2 \xrightarrow{next^*}$ may be non-empty,
- otherwise v_1, v_2 are in different partitions.

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

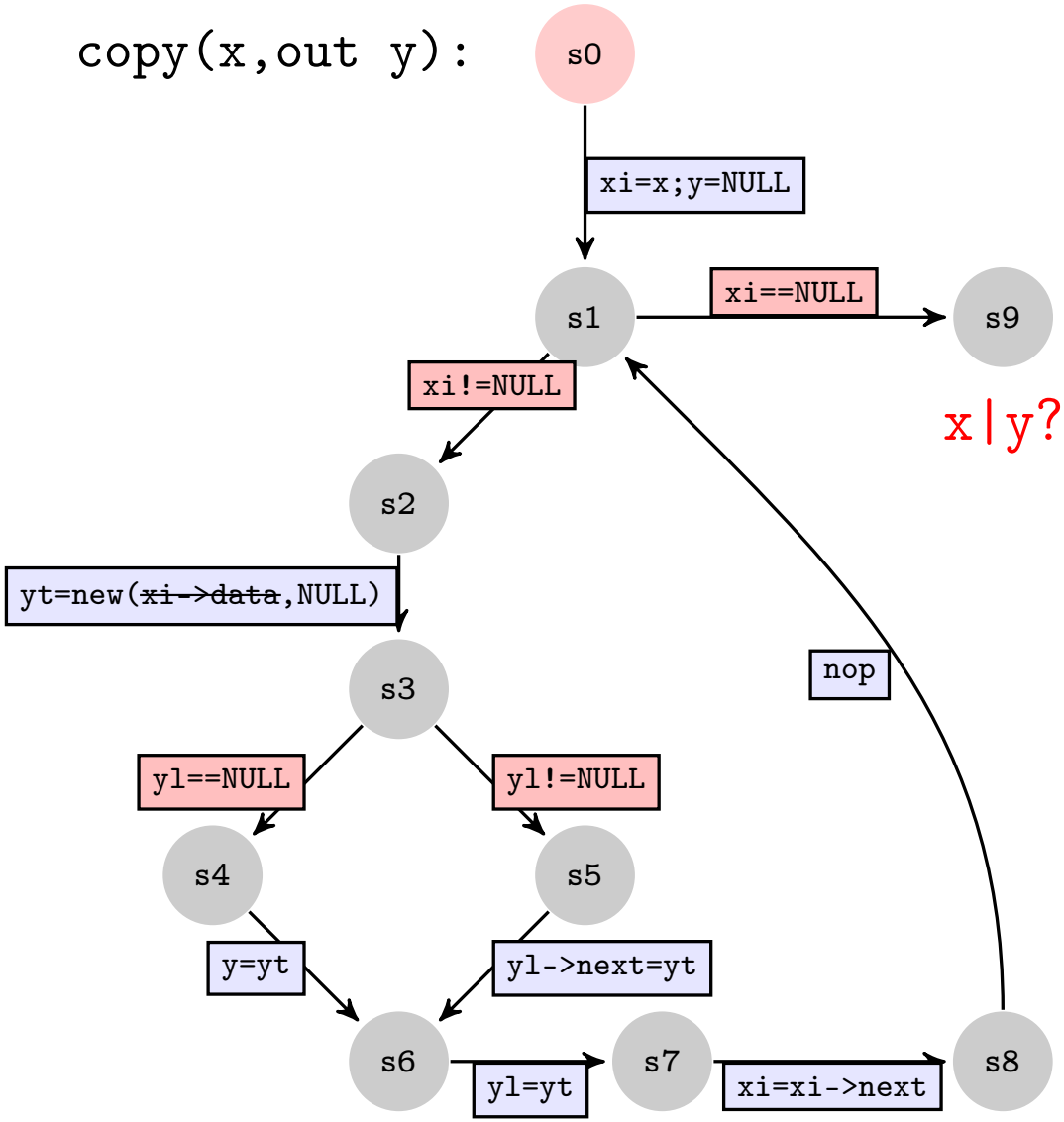


12/34 or 12/3/4

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



Example: Analysis of Heap Separation [CC'77]



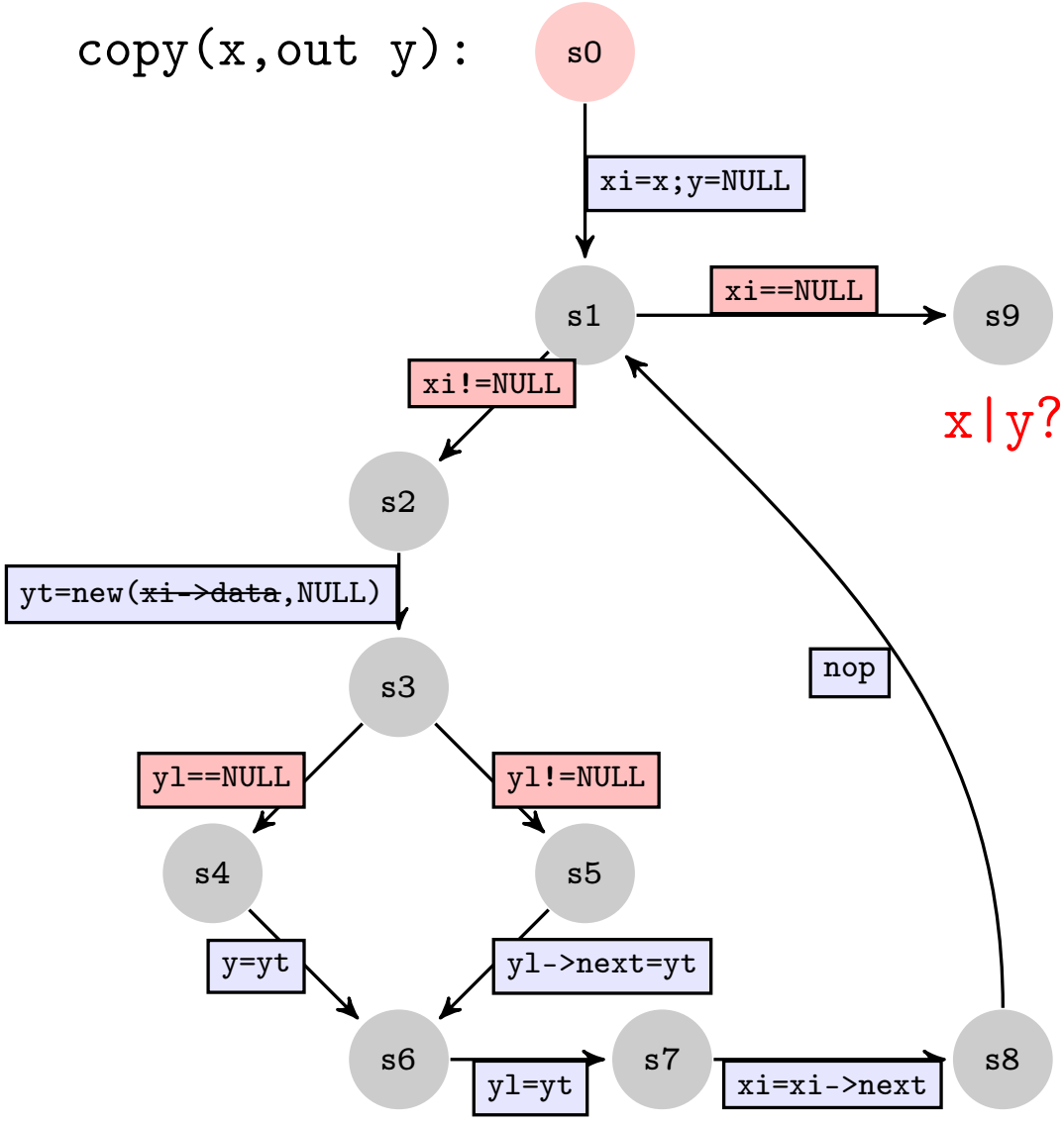
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

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

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]



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

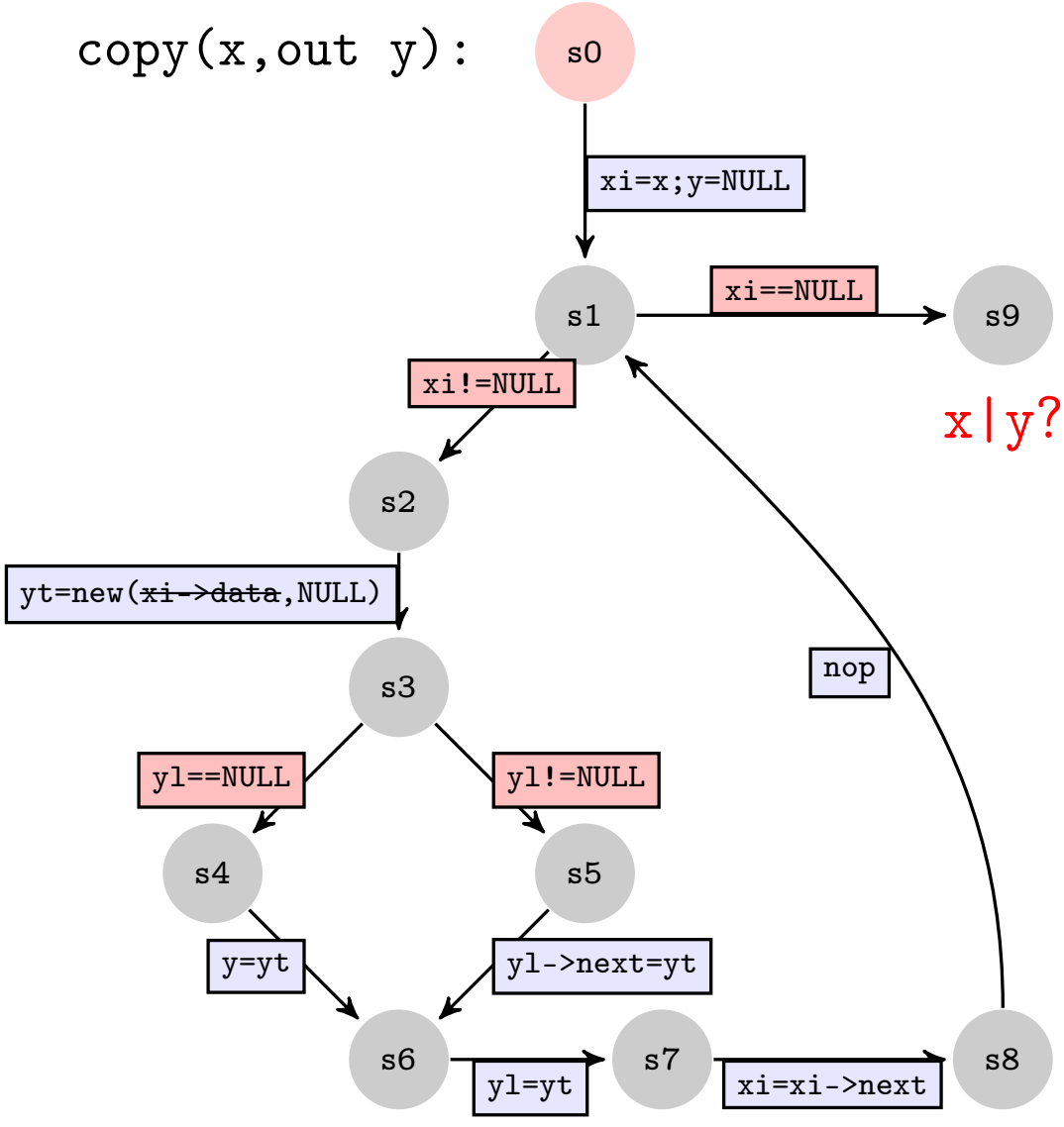
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

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	$\bot$	

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

Example: Analysis of Heap Separation [CC'77]



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

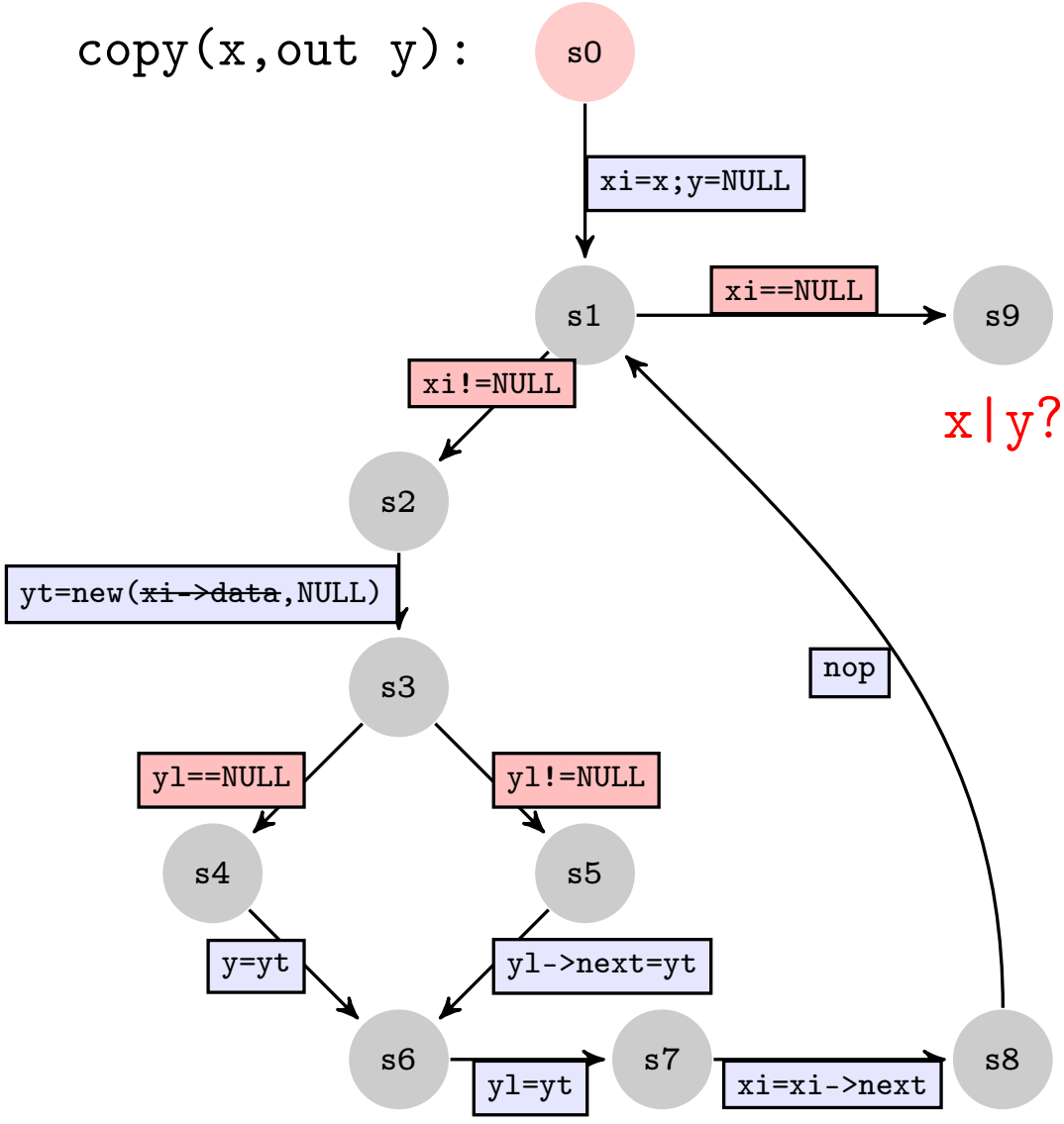
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s1 extracted:

CP	$\nu^\sharp$	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	✓

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

Example: Analysis of Heap Separation [CC'77]



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

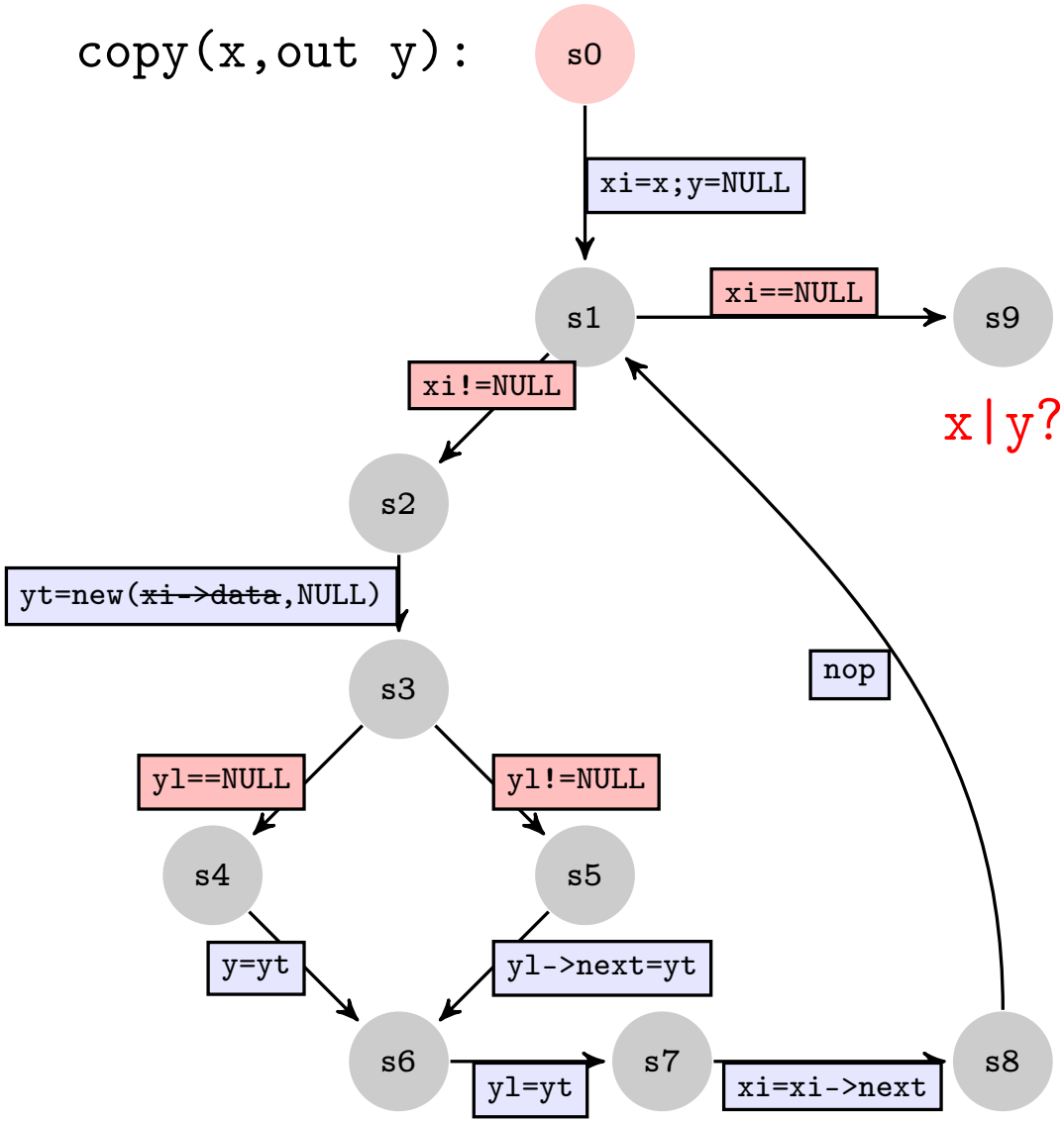
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

s2 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	$\bot$	
s5	$\bot$	
s6	$\bot$	
s7	$\bot$	
s8	$\bot$	
s9	x xi|y|yl yt	✓

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

Example: Analysis of Heap Separation [CC'77]



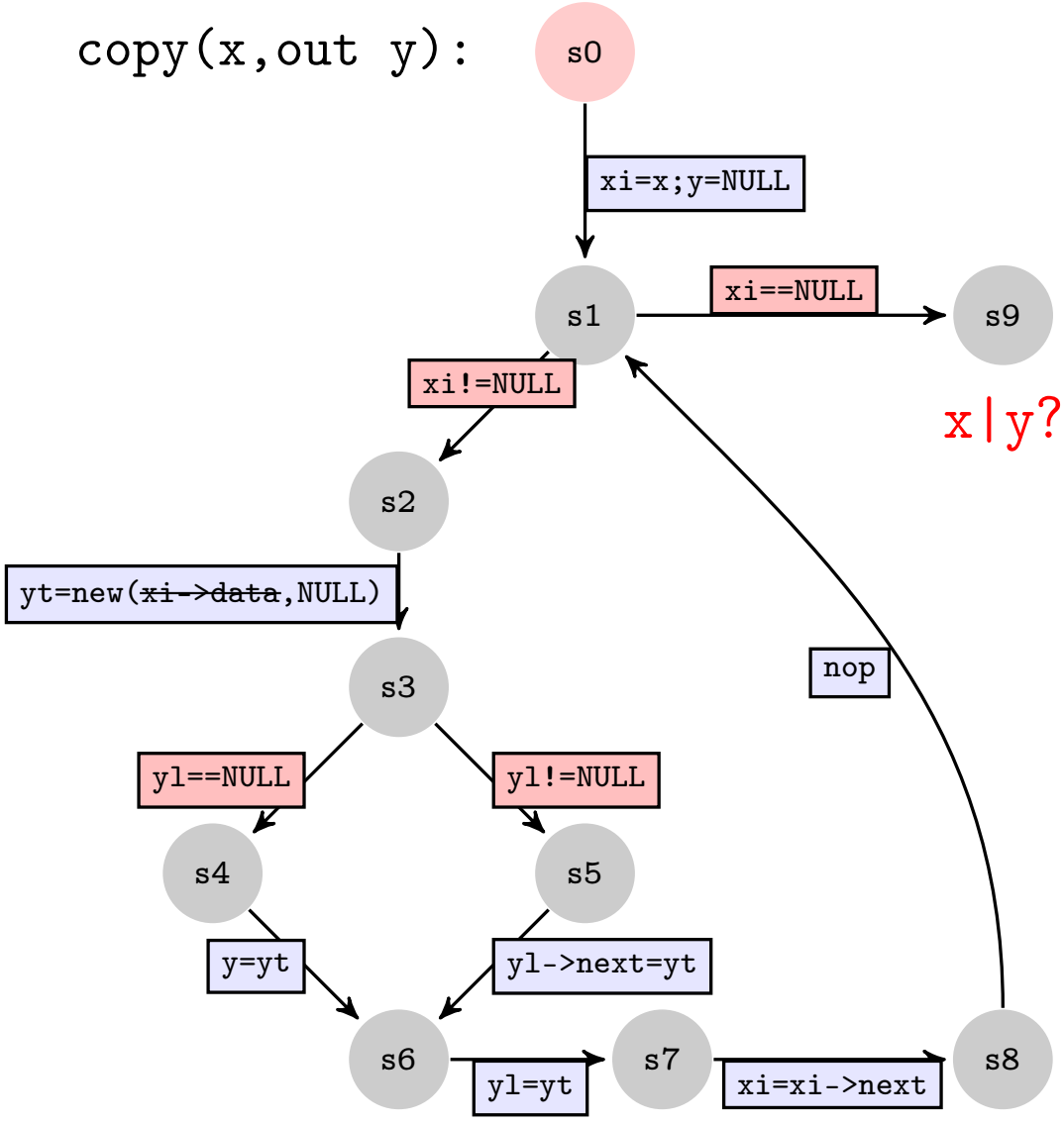
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

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

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	✓

Example: Analysis of Heap Separation [CC'77]



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

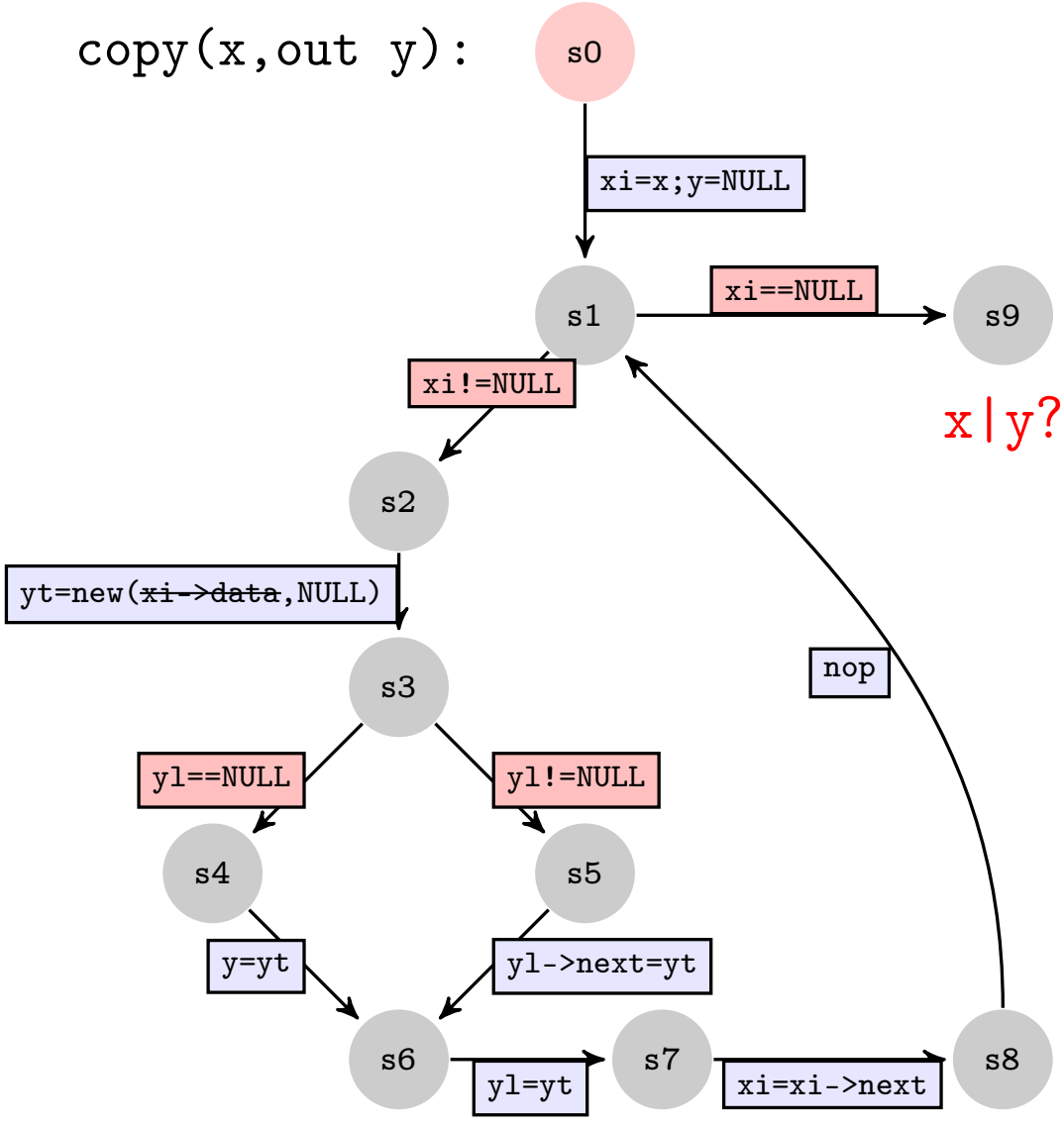
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

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	$\bot$	
s8	$\bot$	
s9	x xi|y|yl yt	✓

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

Example: Analysis of Heap Separation [CC'77]



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

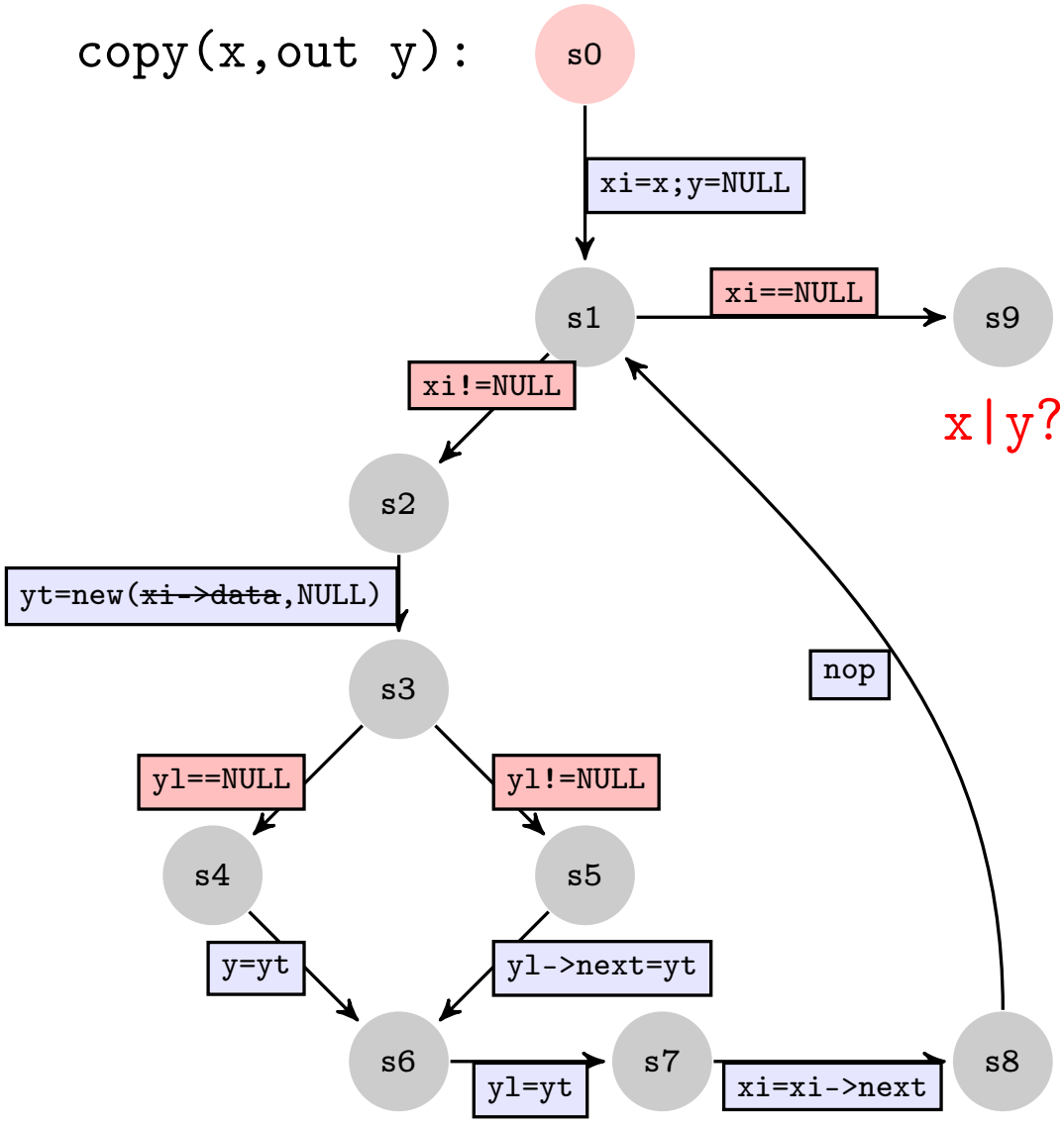
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

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	$\bot$	
s8	$\bot$	
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]



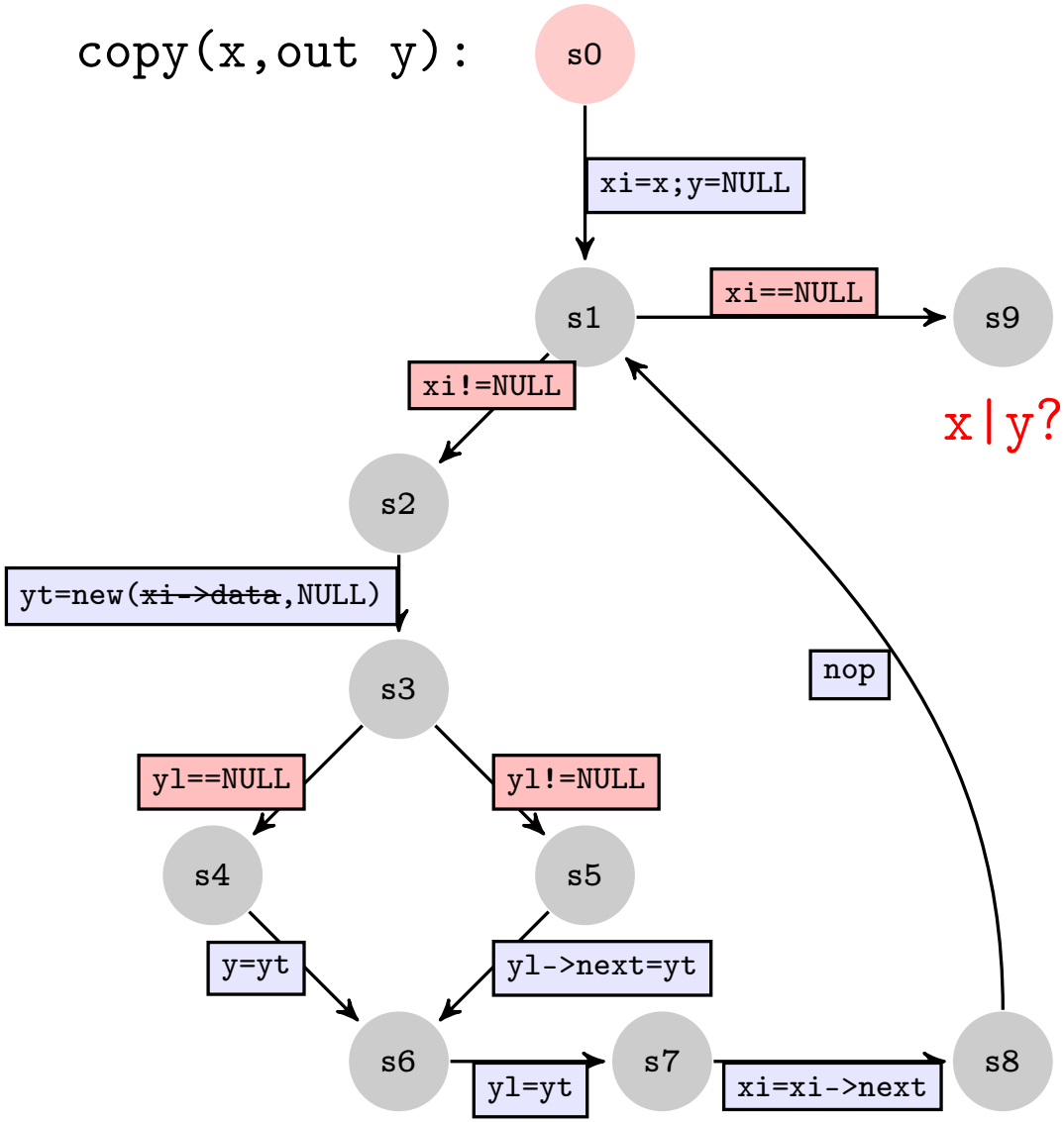
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

$\nu^\sharp \sqcup^\sharp u^\sharp$ 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	$\bot$	
s9	x xi|y|yl yt	✓

Example: Analysis of Heap Separation [CC'77]



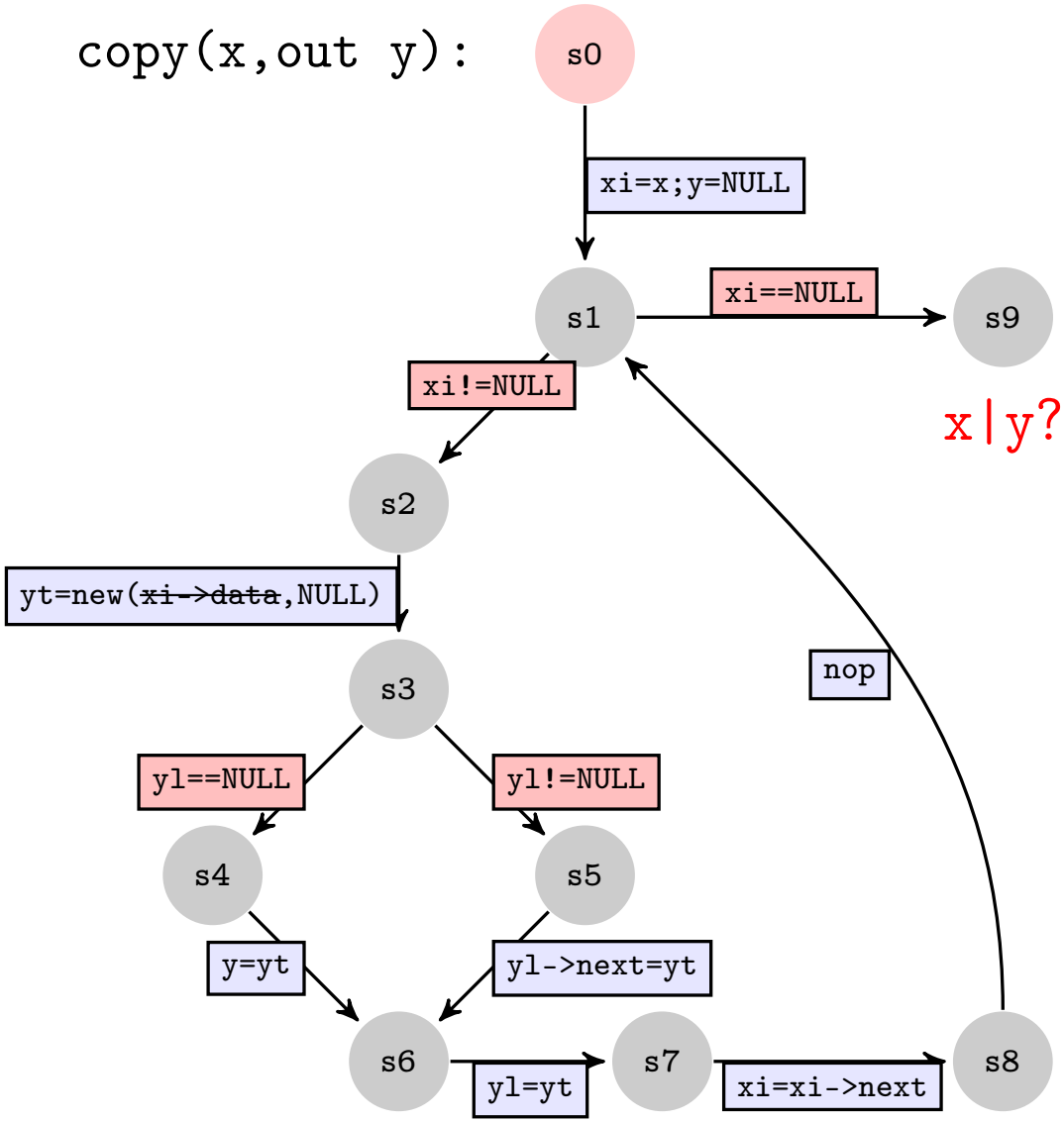
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

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

s7 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	x xi|y yt yl	✓
s9	x xi|y|yl yt	✓

Example: Analysis of Heap Separation [CC'77]



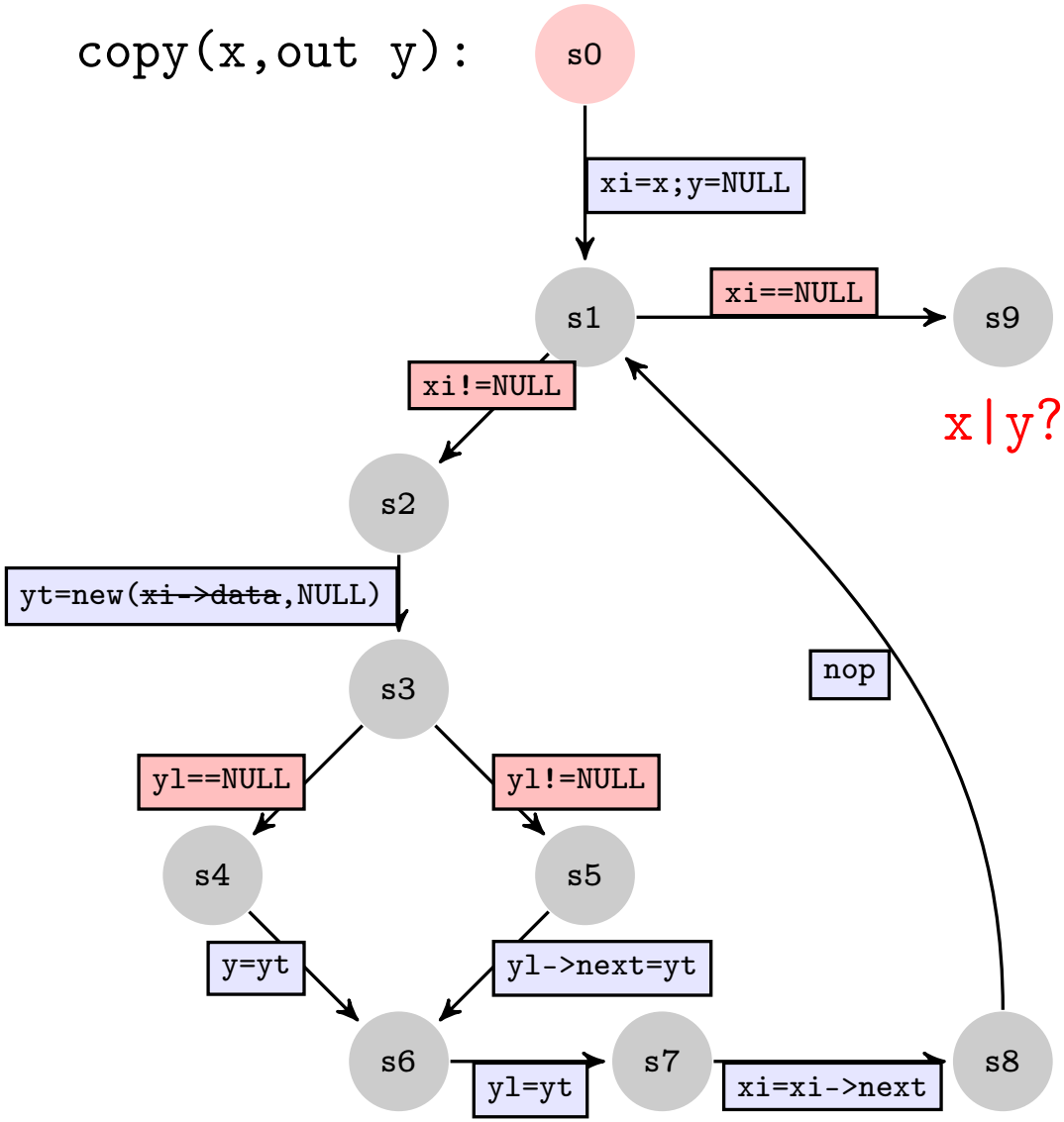
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

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

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	✓

Example: Analysis of Heap Separation [CC'77]



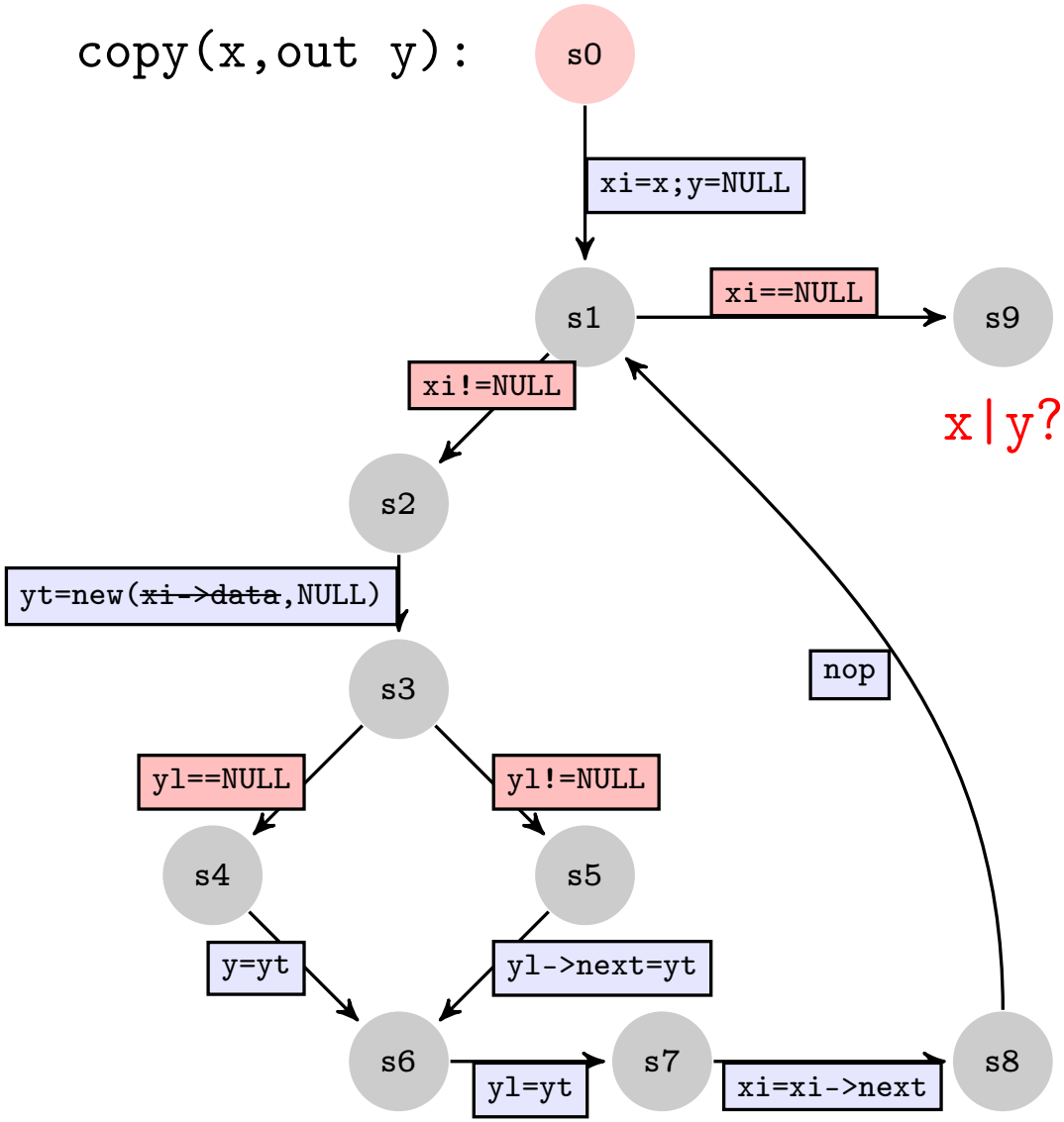
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

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

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	✓

Example: Analysis of Heap Separation [CC'77]



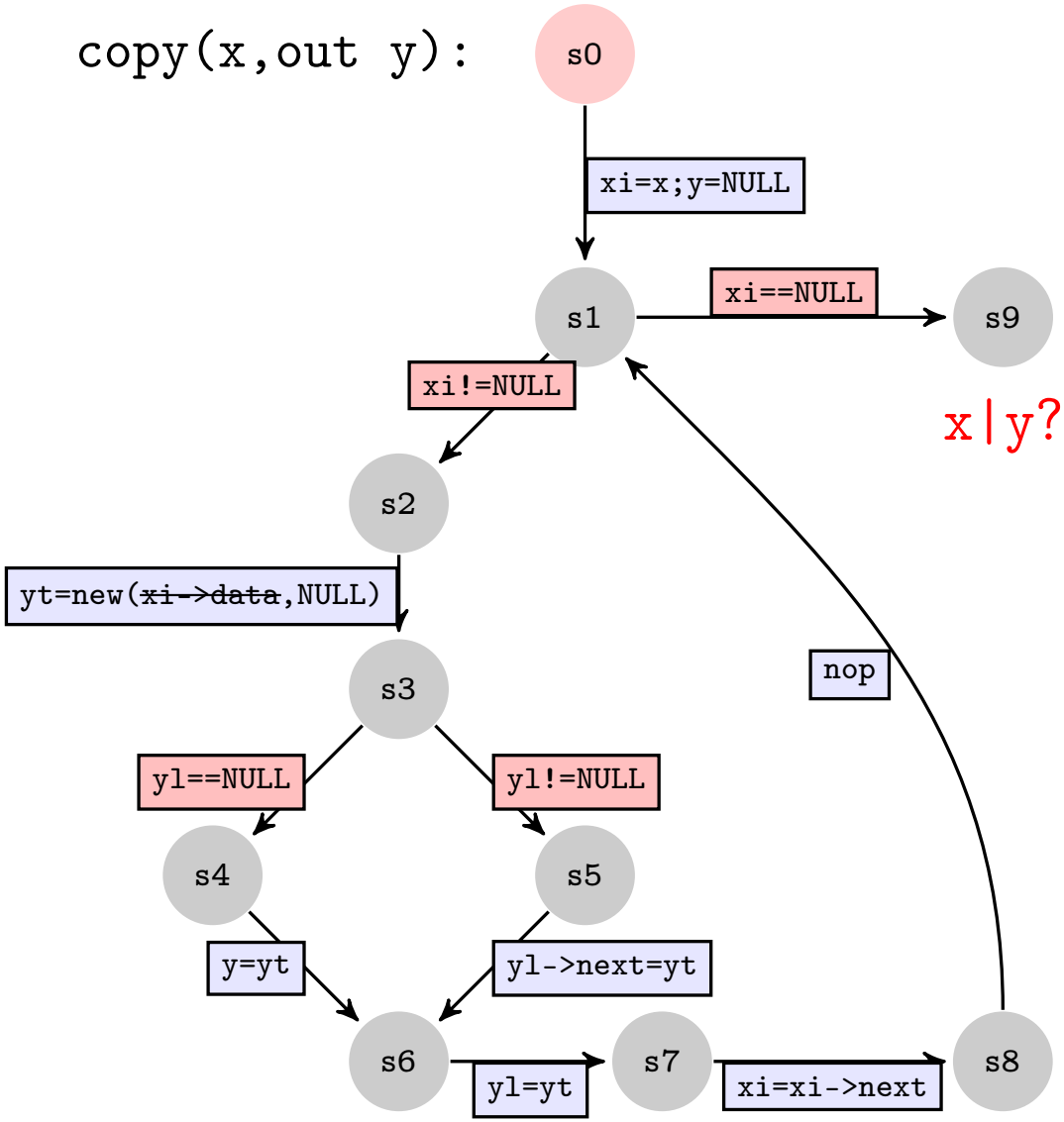
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

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

... 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	✓

Example: Analysis of Heap Separation [CC'77]



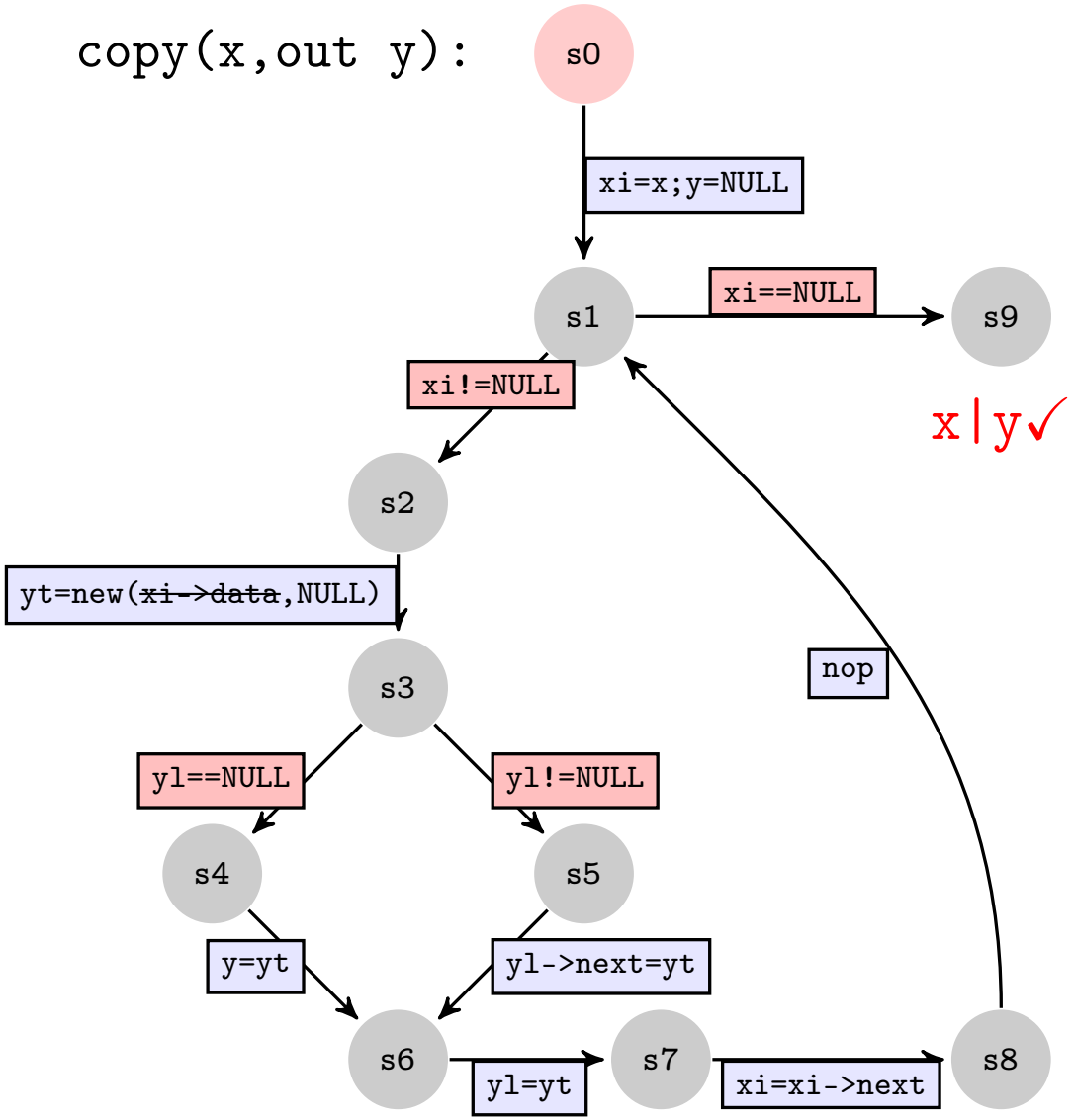
$\nu^\sharp \sqsubseteq^\sharp u^\sharp$ iff $\forall p \in \nu^\sharp\ \exists q \in u^\sharp.\ p \subseteq q$

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

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	

Example: Analysis of Heap Separation [CC'77]



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

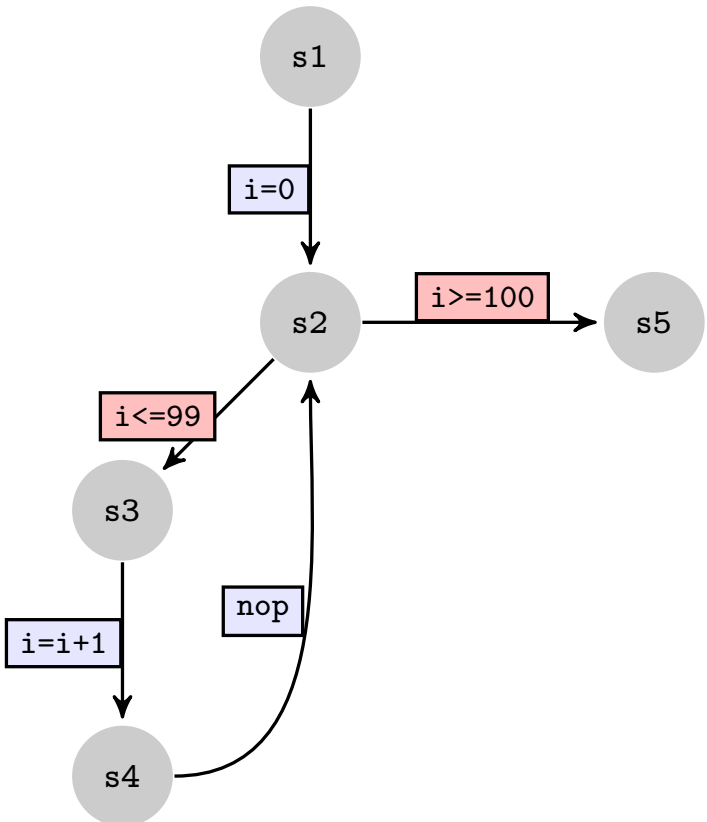
$\nu^\sharp \sqcup^\sharp u^\sharp$ based on union-find

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	

Example: Numeric Analysis with Intervals

Recall:

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



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

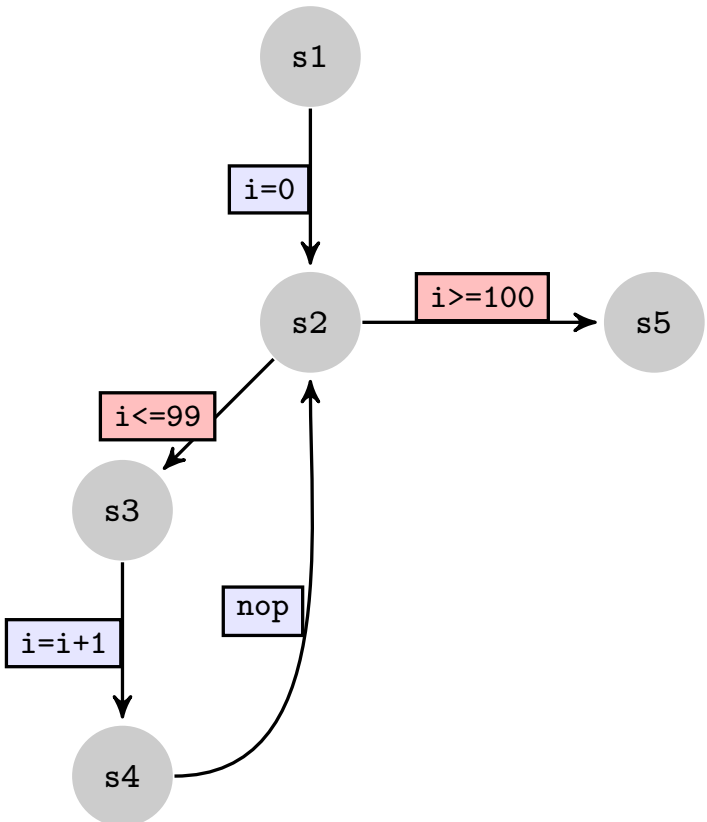
Initially:

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.!



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

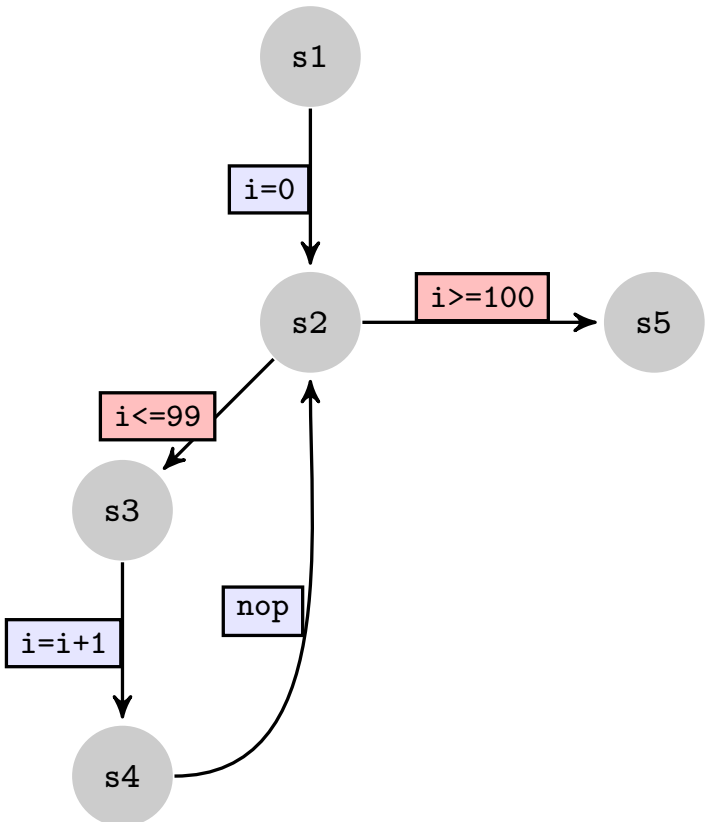
s1 extracted:

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

Example: Numeric Analysis with Intervals

Recall:

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



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

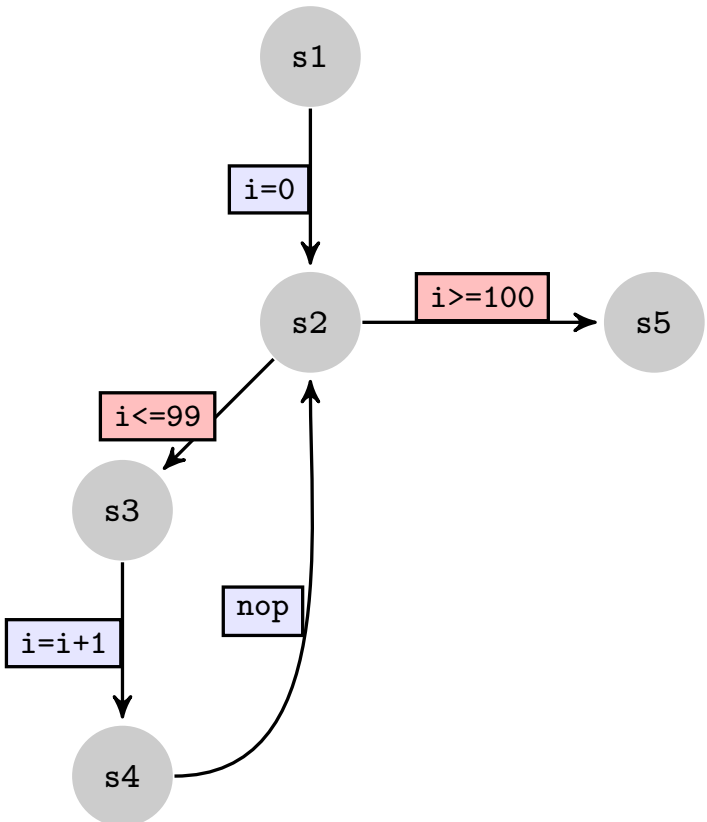
s2 extracted:

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

Example: Numeric Analysis with Intervals

Recall:

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



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

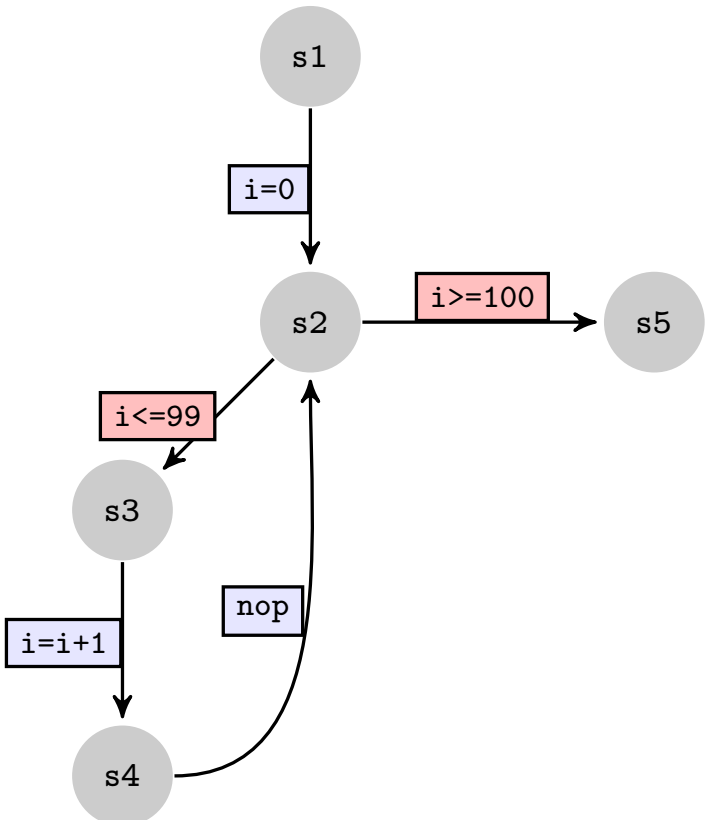
s3 extracted:

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:

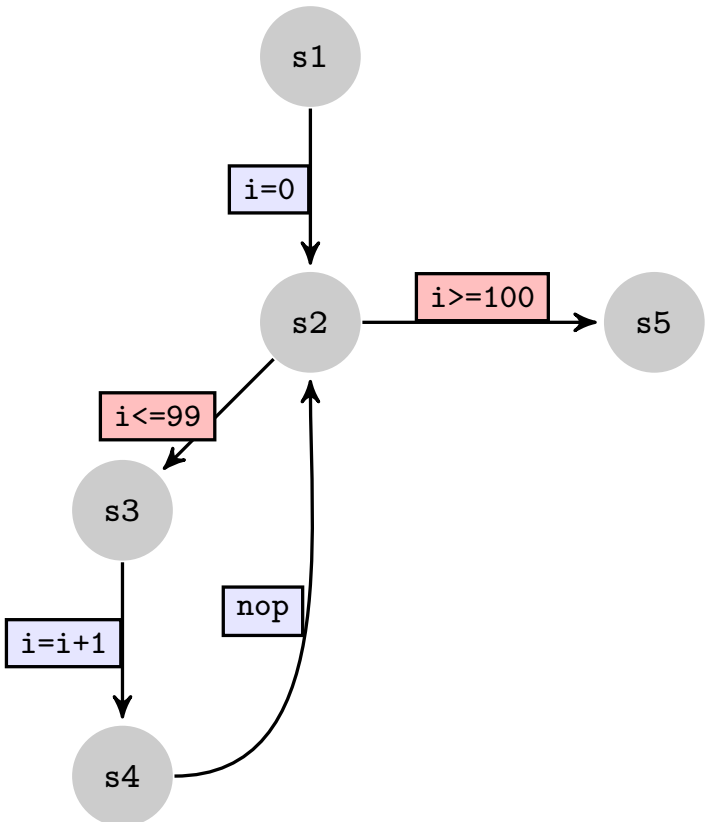
CP	i	W
s1	$\top$	
s2	(0,1)	✓
s3	(0,0)	
s4	(1,1)	
s5	$\bot$	

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

Example: Numeric Analysis with Intervals

Recall:

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



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

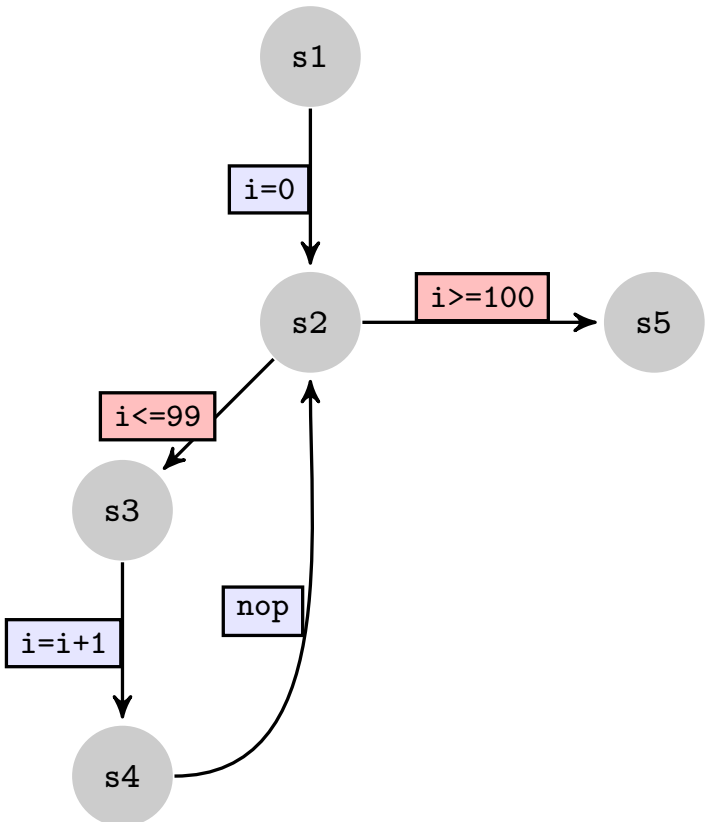
s2 extracted:

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

Example: Numeric Analysis with Intervals

Recall:

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



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

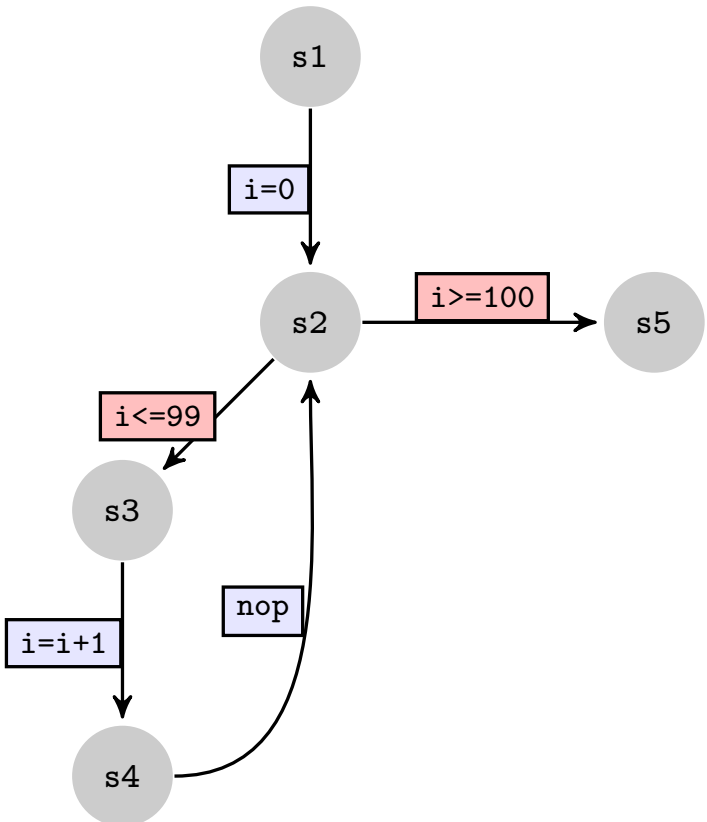
s3 extracted:

CP	i	W
s1	$\top$	
s2	(0,1)	
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.!



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

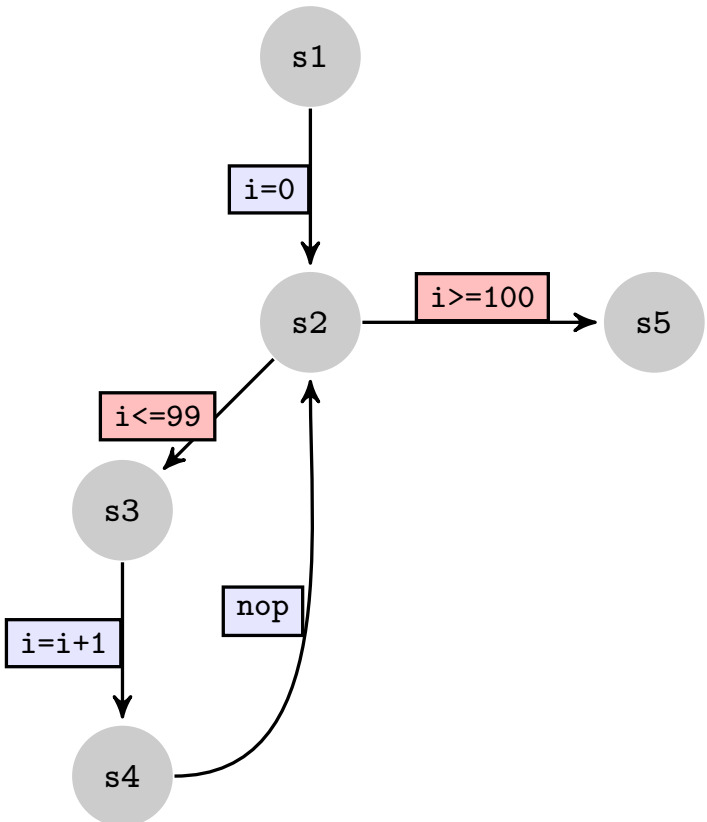
s4 extracted:

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.!



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

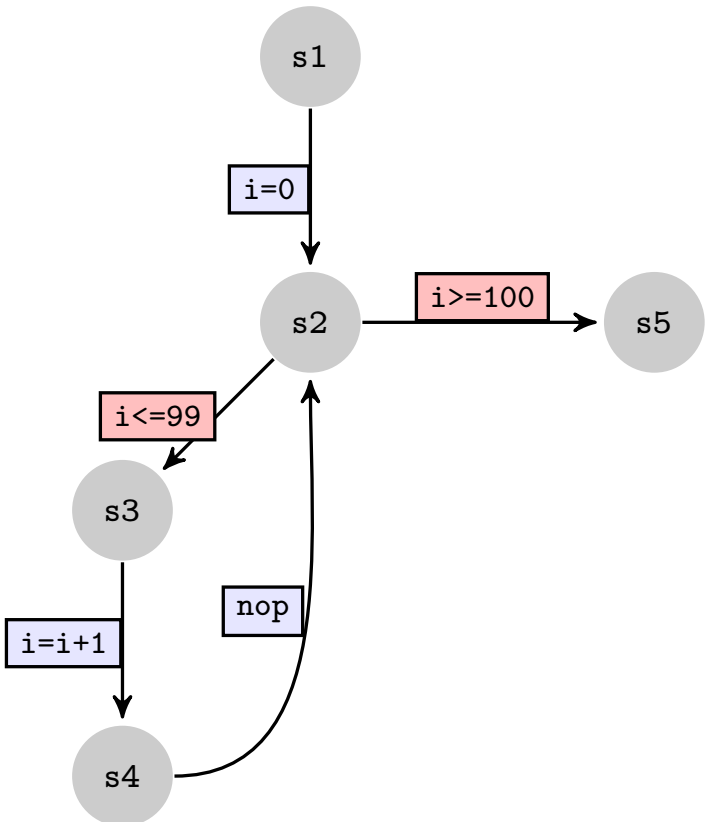
... 98 it. later

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



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

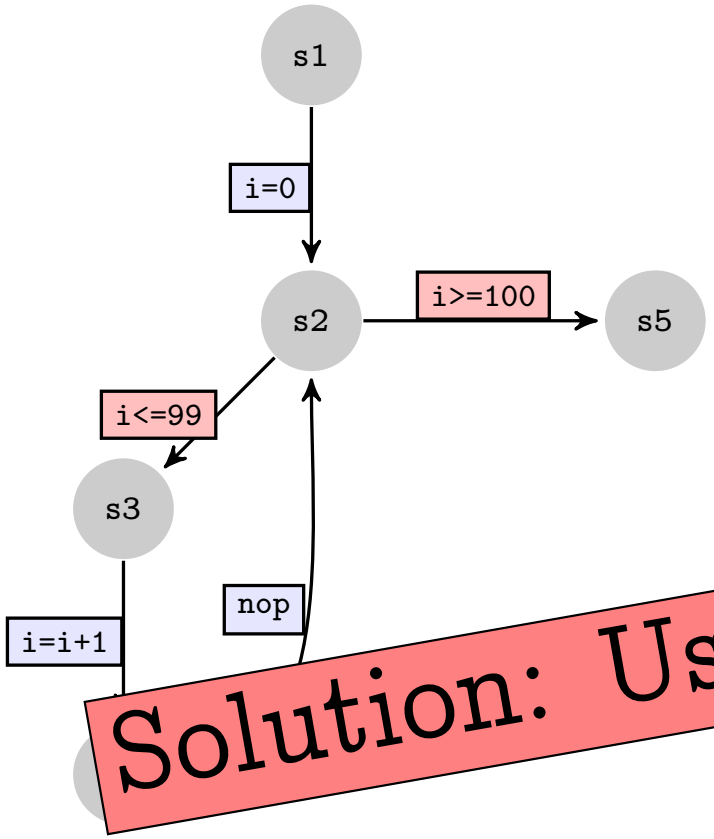
s1 extracted:

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

Example: Numeric Analysis with Intervals

Recall:

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



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

CP	i	W
s1	$\top$	
s4	(1,100)	
s5	(100,100)	

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

Definition

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

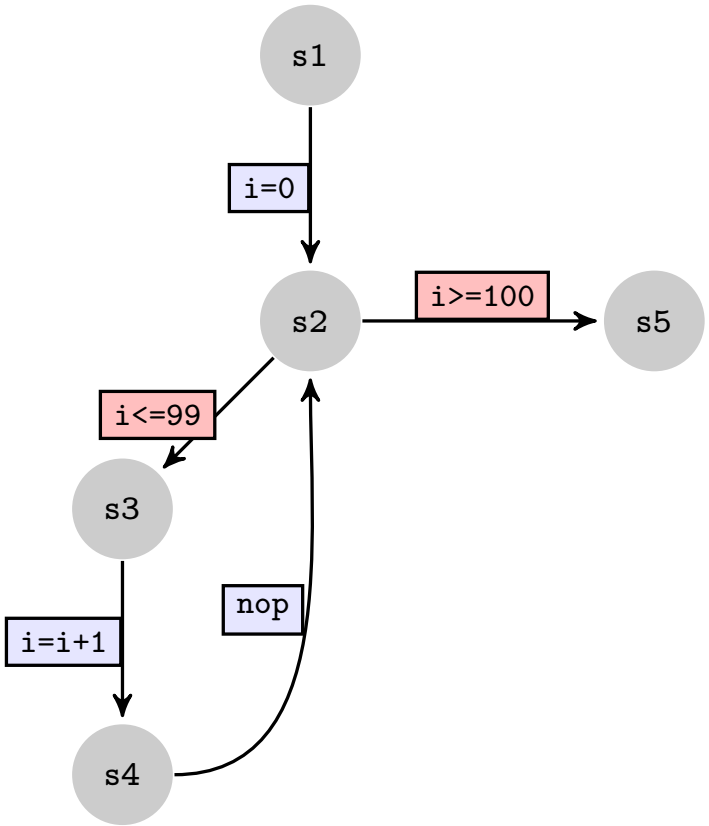
- $\forall x, y \in L.\ x \sqcup y \sqsubseteq x \nabla y$
- for all sequences $(l_n)_{n\in\mathbb{N}}$, the (ascending) chain $(w_n)_{n\in\mathbb{N}}$
  $w_0 = l_0, \quad w_{i+1} = w_i \nabla l_{i+1} \quad \text{for } i > 0$
  stabilises eventually.

Widening for $(\mathrm{Int}, \subseteq)$: $(\ell_0, u_0)\nabla(\ell_1, u_1) = (\ell_2, u_2)$ 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: $\bot \subseteq (0,0) \subseteq (0,0)\nabla(0,1) = (0, +\infty)$

Example: Numeric Analysis with Intervals



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

With $\sqcup$, in 100 it.:

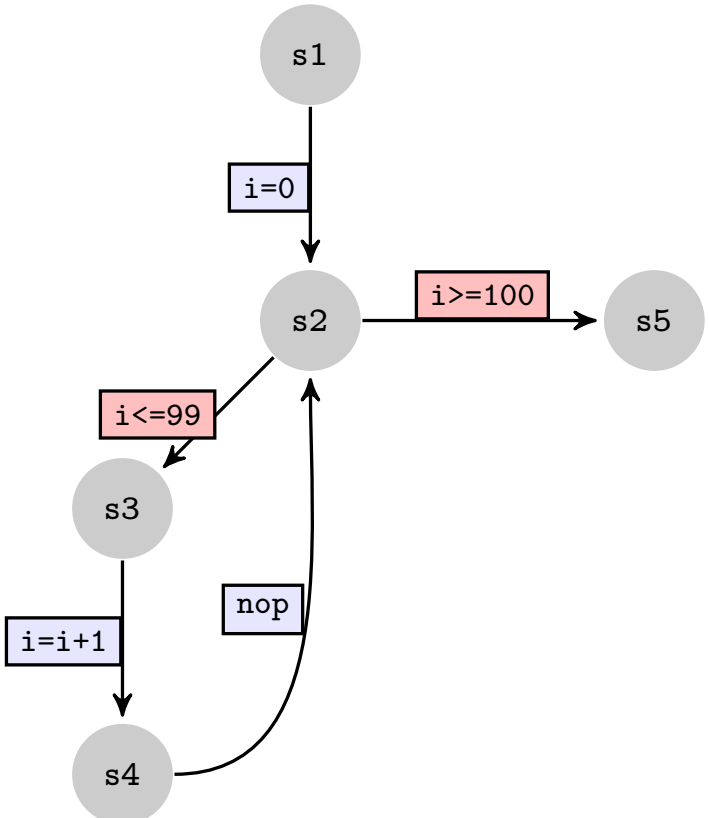
CP	i
s1	$\top$
s2	(0,100)
s3	(0,99)
s4	(1,100)
s5	(100,100)

Initialy:

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

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

Example: Numeric Analysis with Intervals



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

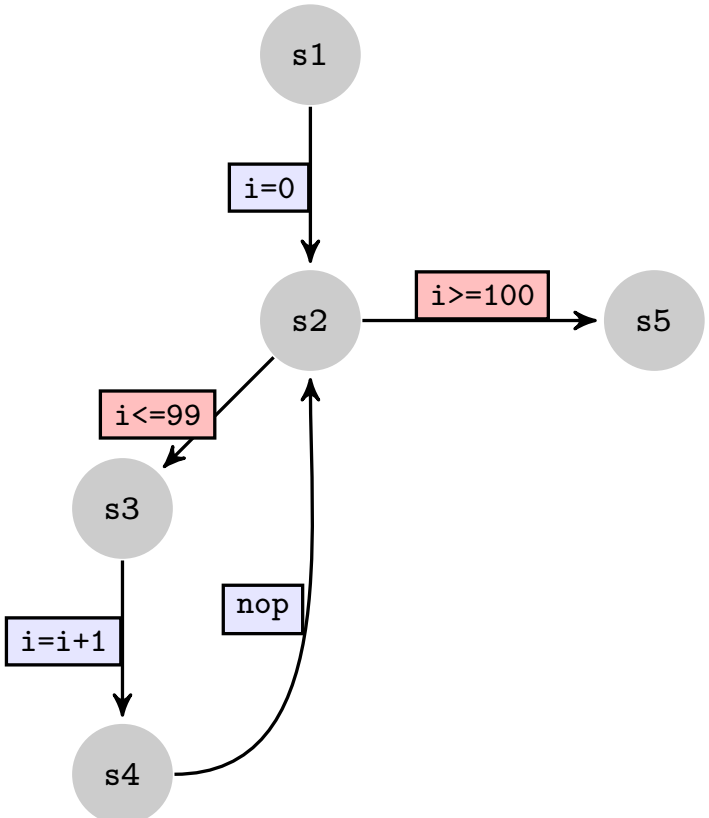
With $\sqcup$, in 100 it.:

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	$\bot$	

Example: Numeric Analysis with Intervals



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

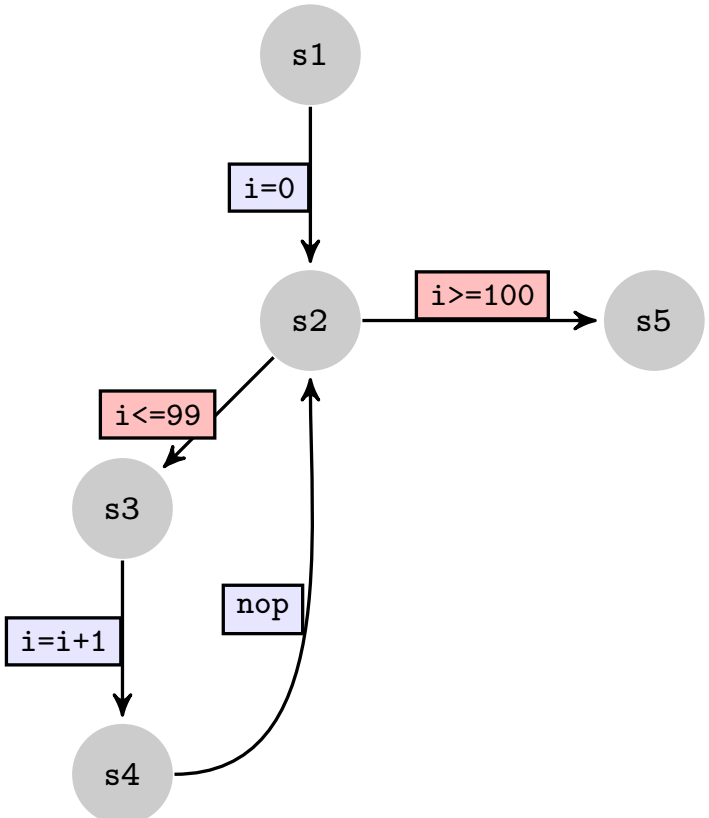
With $\sqcup$, in 100 it.:

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



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

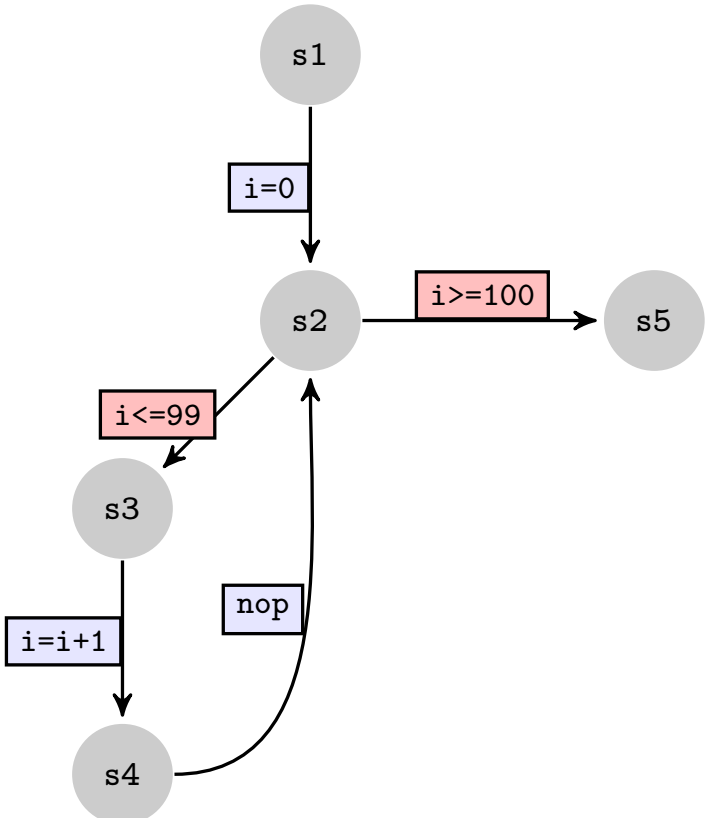
With $\sqcup$, in 100 it.:

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	$\bot$	

Example: Numeric Analysis with Intervals



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

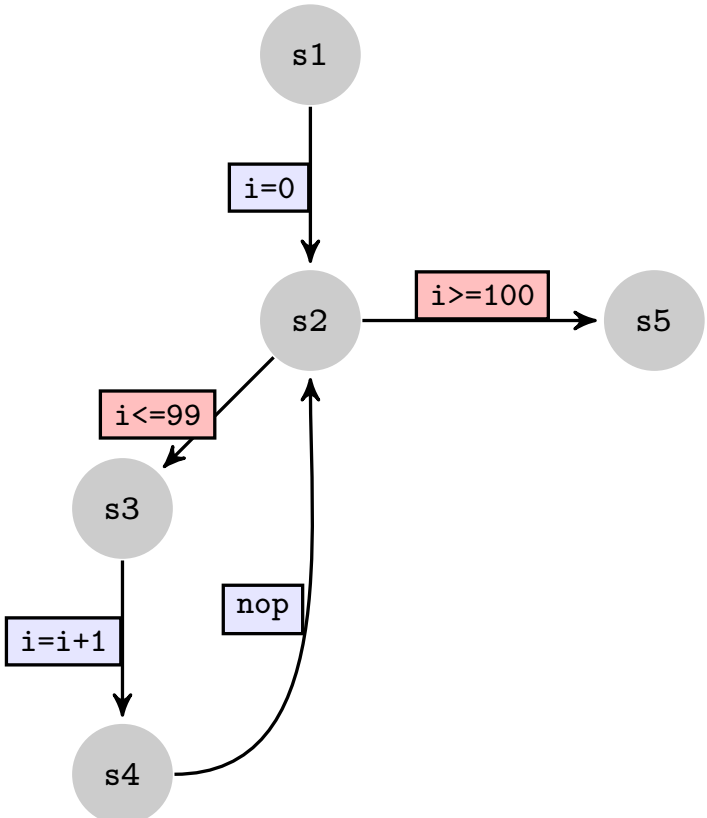
With $\sqcup$, in 100 it.:

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	(0,0)	
s4	(1,1)	
s5	$\bot$	

Example: Numeric Analysis with Intervals



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

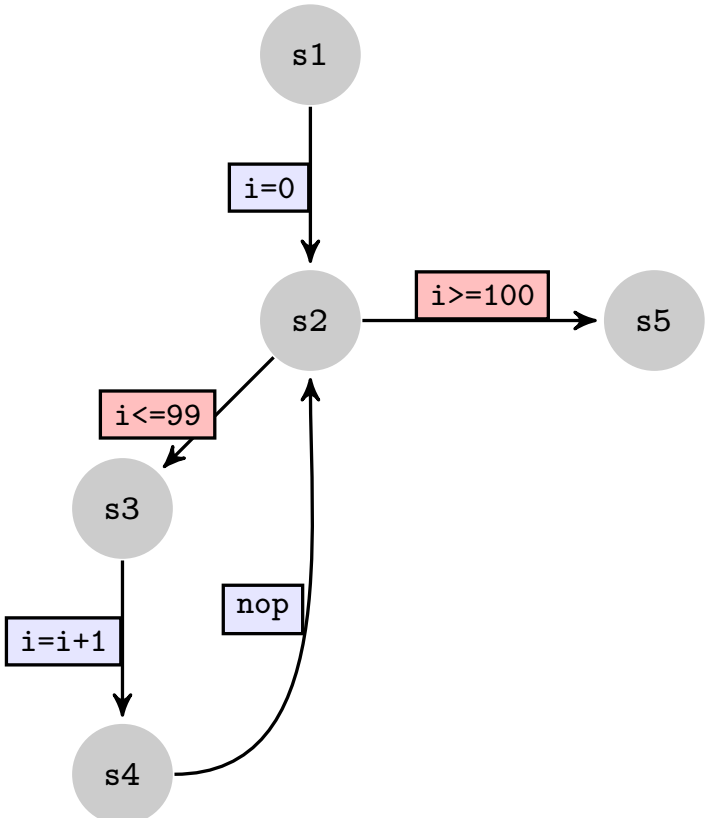
With $\sqcup$, in 100 it.:

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	$(100,+\infty)$	✓

Example: Numeric Analysis with Intervals



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

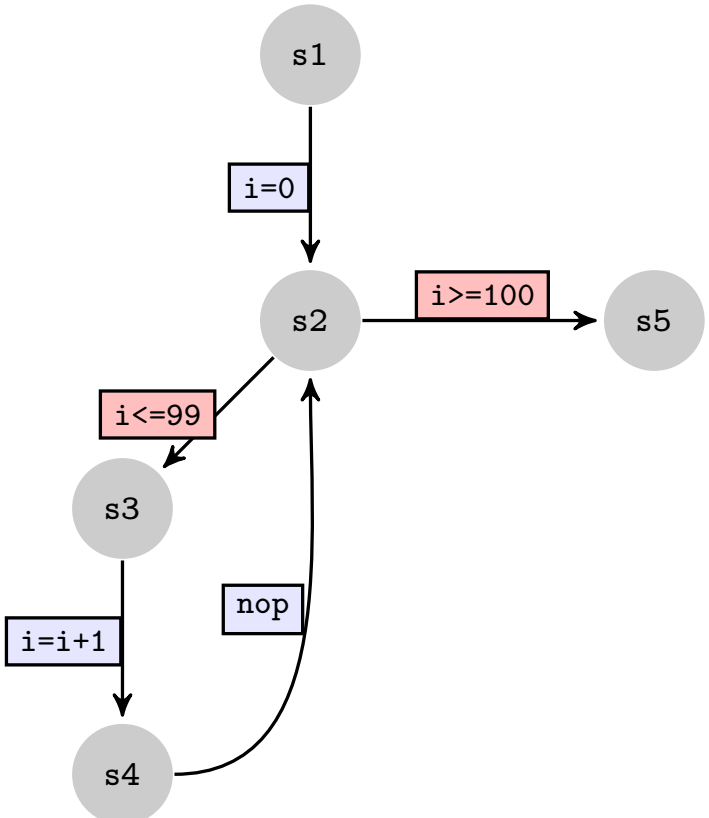
With $\sqcup$, in 100 it.:

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	$(100,+\infty)$	

Example: Numeric Analysis with Intervals



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

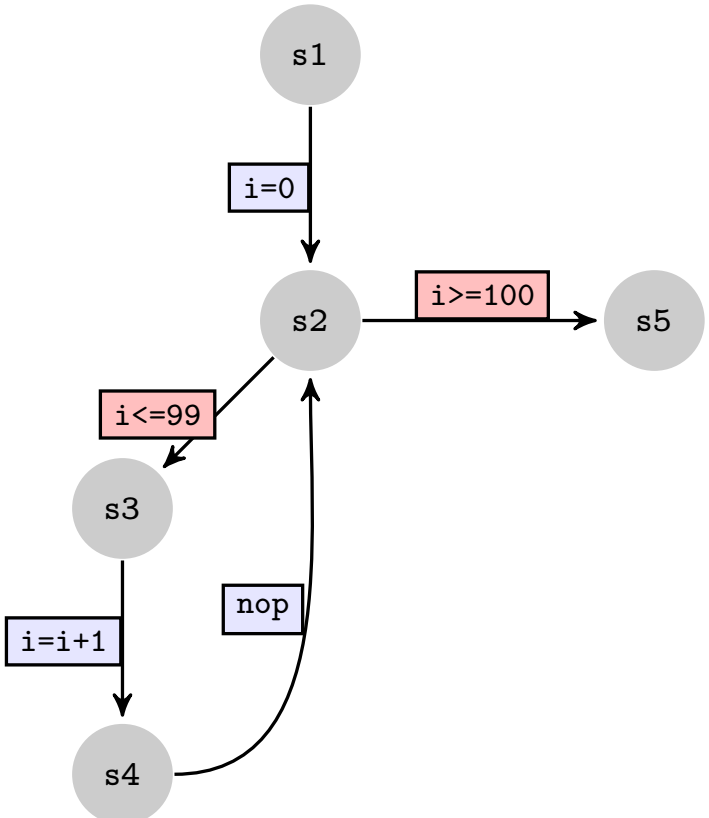
With $\sqcup$, in 100 it.:

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	$(100,+\infty)$	

Example: Numeric Analysis with Intervals



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

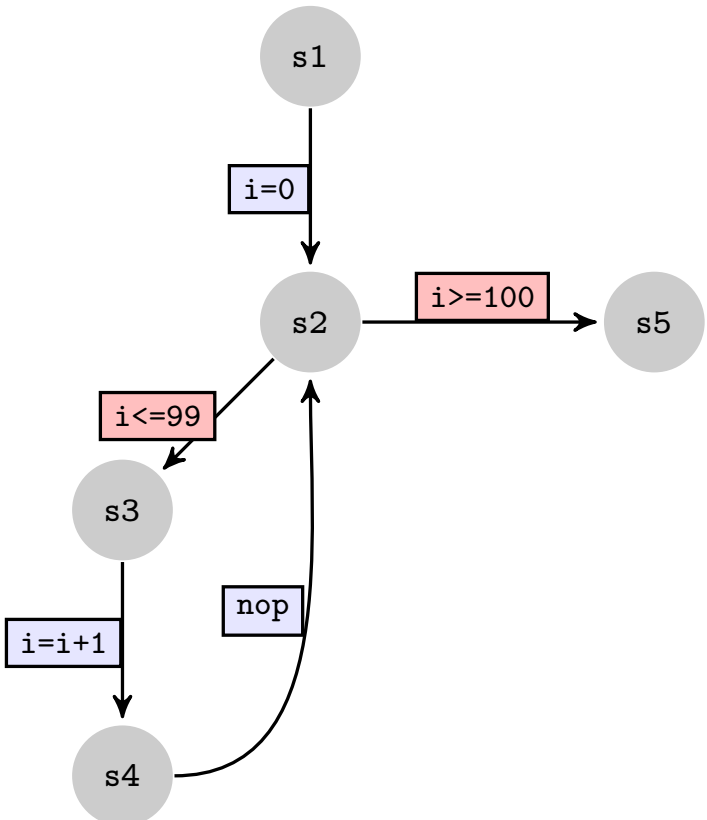
With $\sqcup$, in 100 it.:

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

With $\triangledown$, 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



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

With $\sqcup$, in 100 it.:

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

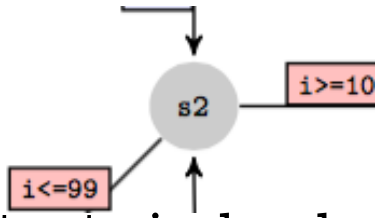
With $\triangledown$, in 2 it. 

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

Conclusion: Use widening carefully if precision is needed!

Other Solutions for Fast and Precise Termination

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



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

$\longrightarrow$[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

$\longrightarrow$[Gonnord & Halbwachs,06]

...

Numerical Abstract Lattices

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

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

Octagons: $\bigwedge_{x,y \in V} \ \pm x \pm y \leqslant c$ [Miné'01]

Polyhedra: $\bigwedge \sum_{x_i \in V} c_i x_i \leqslant c$ [Cousot& Halbwachs,78]

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

In general, 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
$\longrightarrow$ `apron.cri.ensmp.fr`, [Jeannet& Miné,09]

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

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

Frama-C: platform for verifying and analysing C programs
$\longrightarrow$ `frama-c.com`

PPL: numerical domains (C++)
$\longrightarrow$ `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

- G. Sutre, VTSA'08
- D. Monniaux, VTSA'12
- M. Müller-Olm, VTSA'10

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 ⟹ l̂f[i]=l̂f[i+1]+l̂f[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

Comparison with Numerical Analyses

```
/* @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

Intra-procedural Analysis Model

Recall the formal model of IMPR:

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

Intra-procedural Analysis Model

Consider IMPR without recursive procedures, then:

Control points	$\mathbf{CP} \ni \ell, \ell'$
Stack	$\mathtt{Stacks} \triangleq \mathbf{DV} \rightarrowtail \mathbb{D} \cup \mathbf{RV} \rightarrowtail \mathbb{L} \ni \mathtt{S}$
Heap	$\mathtt{Heaps} \triangleq [(\mathbb{L} \times \mathbf{FS}) \rightarrowtail (\mathbb{D} \cup \mathbb{L})] \ni \mathtt{H}$
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

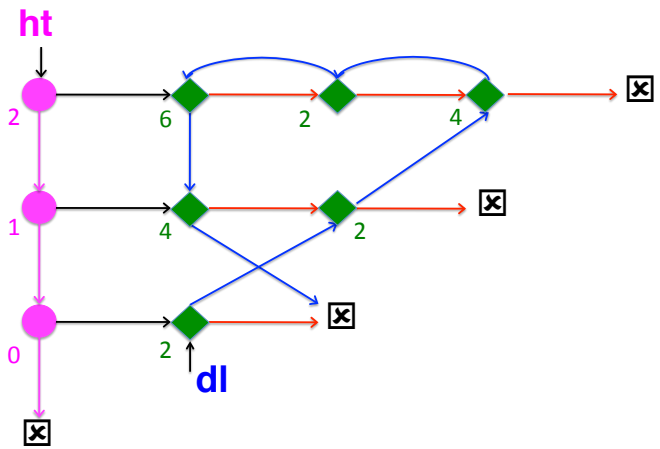
A heap graph is a tuple $\langle \mathcal{N}, \mathcal{E}, \mathcal{L} \rangle$ where $\mathcal{N} = \mathbb{L}$ is a set of (typed) graph *nodes*, $\mathcal{E} : \mathcal{N} \times \mathbf{RF} \rightarrowtail \mathcal{N}$ is a set of (reference field) *labeled edges*, and $\mathcal{L} : \big(\mathcal{N}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

Intra-procedural Analysis Model

Consider IMPR without recursive procedures, then:

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

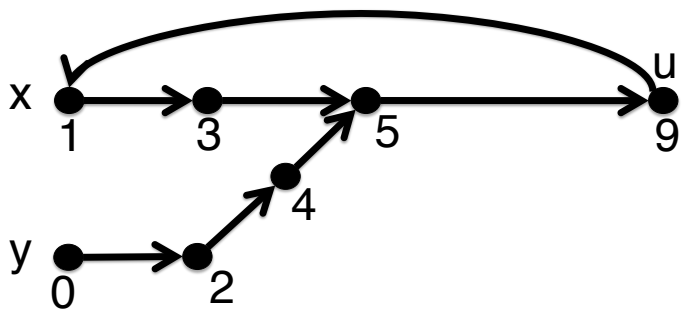
Mem is represented by deterministic labeled graphs, *e.g.*:



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

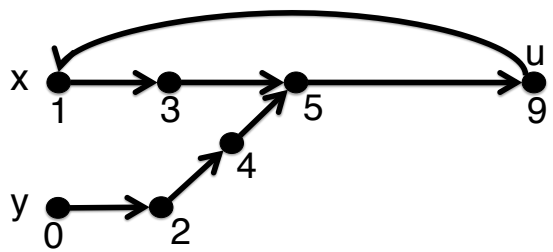
A heap graph without data labels on nodes is called pure heap graph.

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.

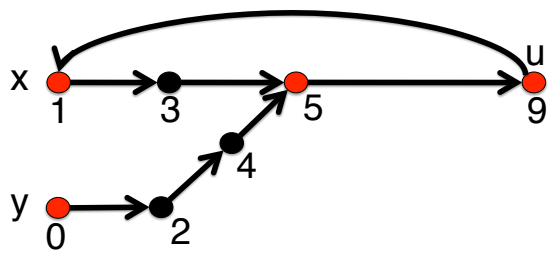


Heap Graphs for Lists of Integers (2/2)

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.

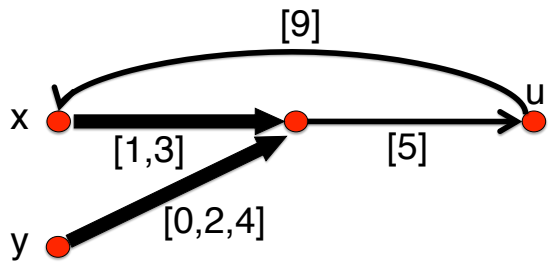


Heap Graphs for Lists of Integers (2/2)

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.

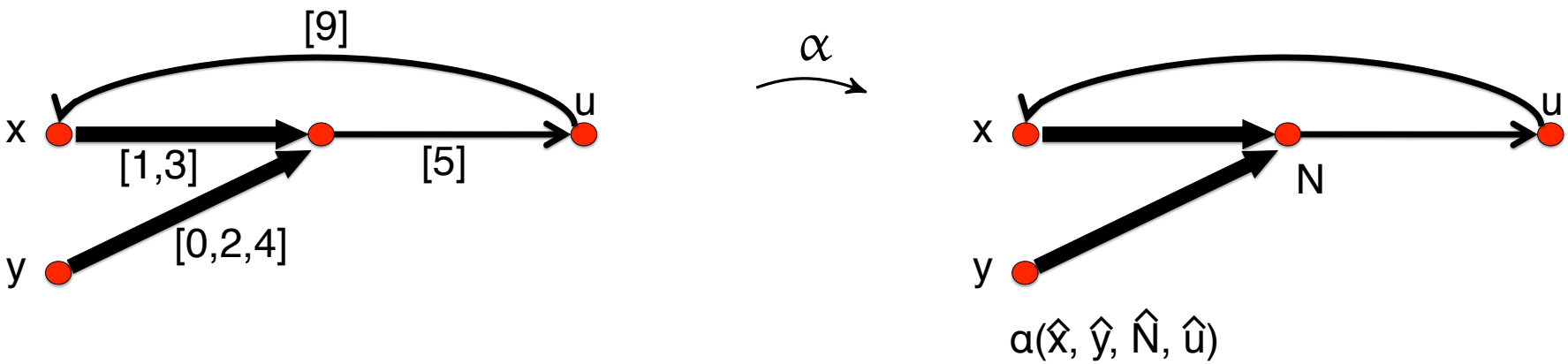


Property

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

Abstract Heap Graphs: Idea

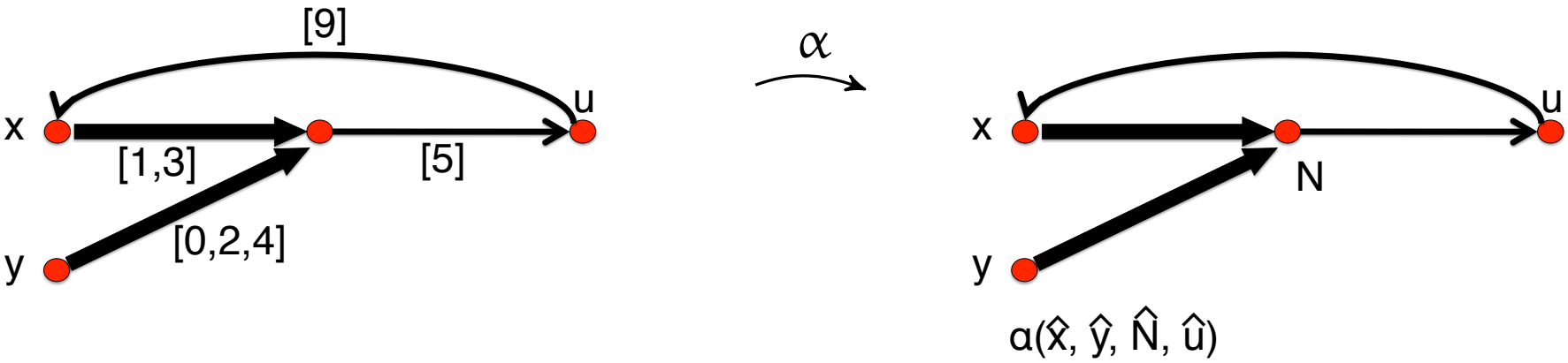
The only source of infinity are the integer words $\longrightarrow$ abstract them!



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

Abstract Heap Graphs: Idea

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



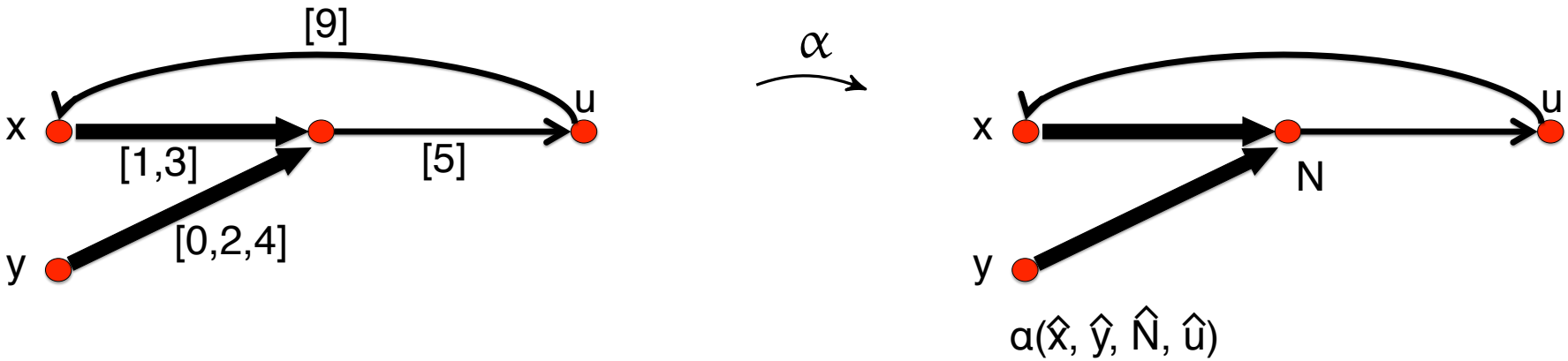
where α on integer words may be, *e.g.*:

Universal constraints abstraction

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

Abstract Heap Graphs: Idea

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



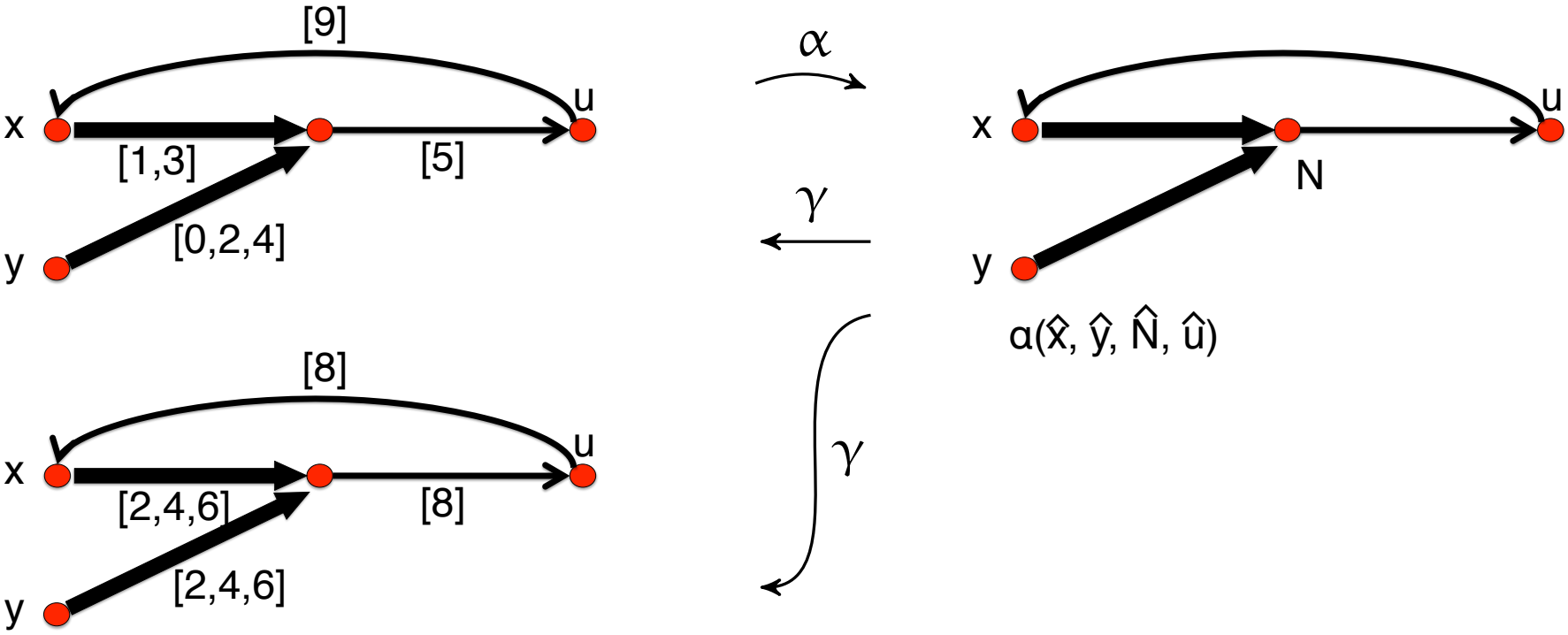
where α on integer words may be, *e.g.*:

Sum constraints abstraction

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

Abstract Heap Graphs: Idea

γ is defined using models of integer words constraints.



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

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

Definition

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

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

where

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

$$(G_1, W_1) \sqsubseteq^{\mathbb{H}} (G_2, W_2) \text{ iff } G_1 \approx_{\text{iso}} G_2 \text{ and } W_1 \sqsubseteq^{\mathbb{W}} W_2$$

$$(G_1, W_1) \sqcup^{\mathbb{H}} (G_2, W_2) = \begin{cases} (G_1, W_1 \sqcup^{\mathbb{W}} W_2) & \text{if } G_1 \approx_{\text{iso}} G_2 \\ \top^{\mathbb{H}} & \text{otherwise} \end{cases}$$

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

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

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.

Logical View: 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 $p \xrightarrow[x]{nxt}\!\!\!\!\rightarrow q$ 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]

Separation Logic: Fragment of Symbolic Heaps

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

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

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

Empty heap: $(S, H) \models \mathit{emp}$ 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
$(S, H_1) \models \Sigma_1$, $(S, H_2) \models \Sigma_2$

Separation Logic: Fragment of Symbolic Heaps

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

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

Examples:

$$
\begin{array}{lcl}
\mathtt{ls}(E, F) & \triangleq & E = F \wedge \mathit{emp} \\
\mathtt{ls}(E, F) & \triangleq & \exists X.\ E \neq F * E \mapsto \{(\mathit{next}, X)\} * \mathtt{ls}(X, F)
\end{array}
$$

$$
\begin{array}{lcl}
\mathtt{ls}^+(E, F) & \triangleq & E \neq F \wedge E \mapsto \{(\mathit{next}, F)\} \\
\mathtt{ls}^+(E, F) & \triangleq & \exists X.\ E \neq F * E \mapsto \{(\mathit{next}, X)\} * \mathtt{ls}^+(X, F)
\end{array}
$$

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

Separation Logic Fragment

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

1. Compositional reasoning due to separation conjunction $*$

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

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]

$$store(a, i, v) \wedge a[j] = b[j] \implies perm(a, b)$$

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

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

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

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

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

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

where:

- $\varphi^i_{SL} \in SL(ls^+)$ 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:

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

where:

- $\varphi^i_{SL} \in SL(ls^+)$ 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

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

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"

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

- "n contains a Fibonacci sequence"

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

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

- "lists n and m have the same content"

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

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

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

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

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

is parameterised by

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

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

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

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

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

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

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

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!

Widening of Abstract Heaps

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

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

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

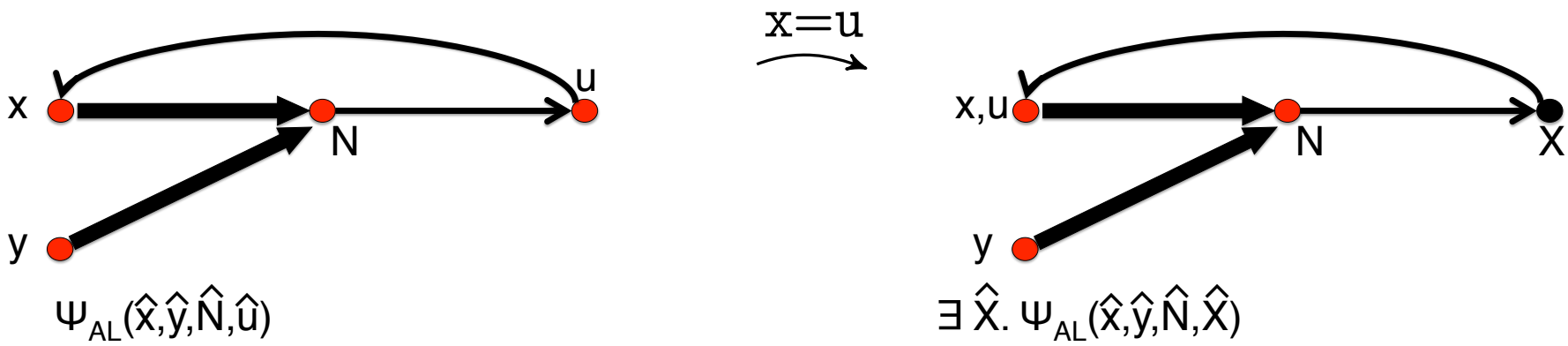
where

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

and

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

Abstract Transformers: Issues



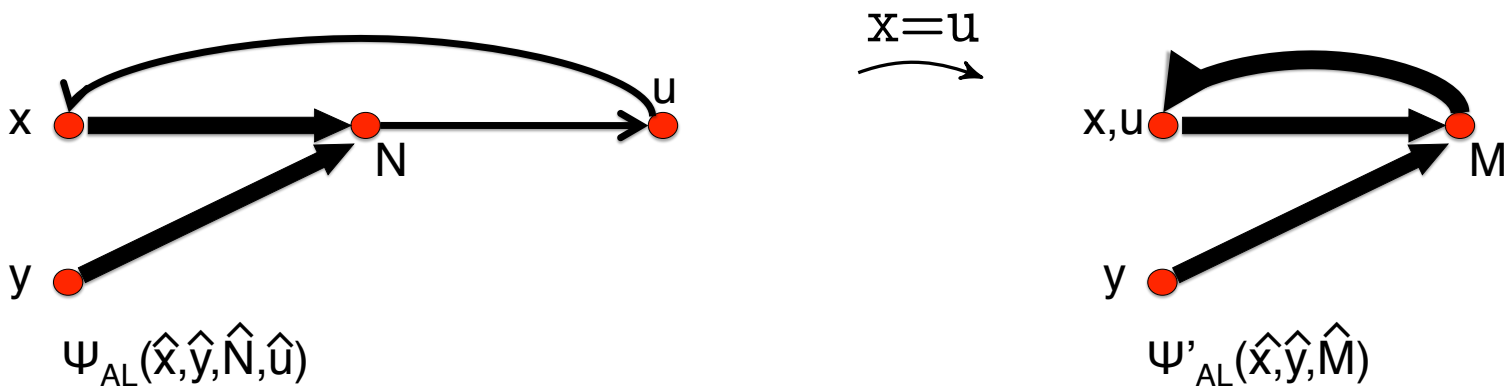
Issue 1:

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



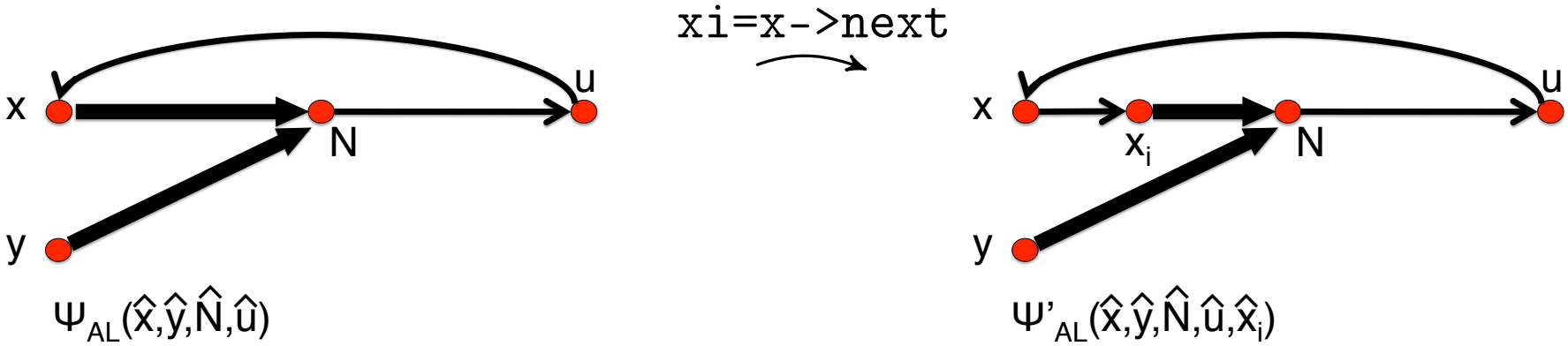
Issue 1:

Define a procedure for existential quantifier elimination in AL.

Required by Issue 1:

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

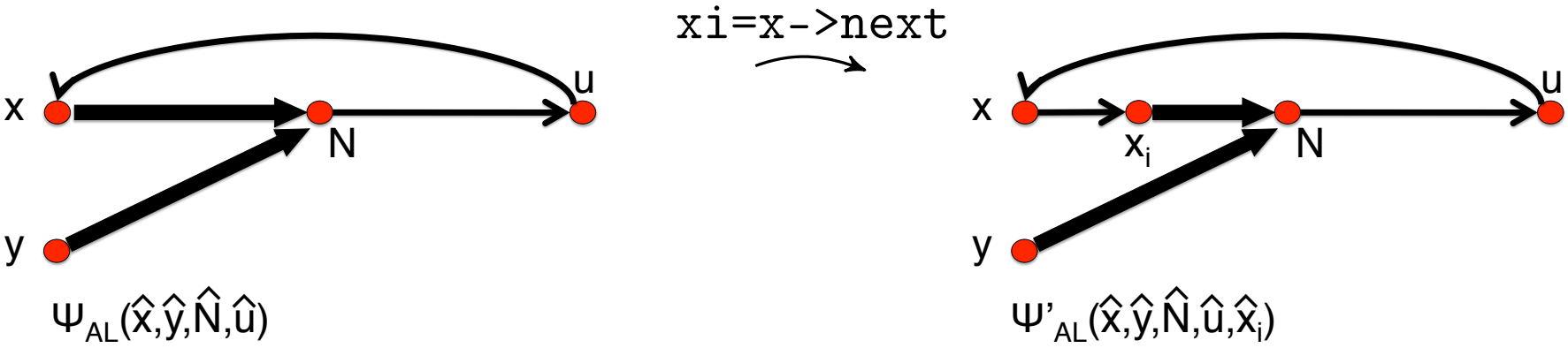
Abstract Transformers: Issues



Issue 2:

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

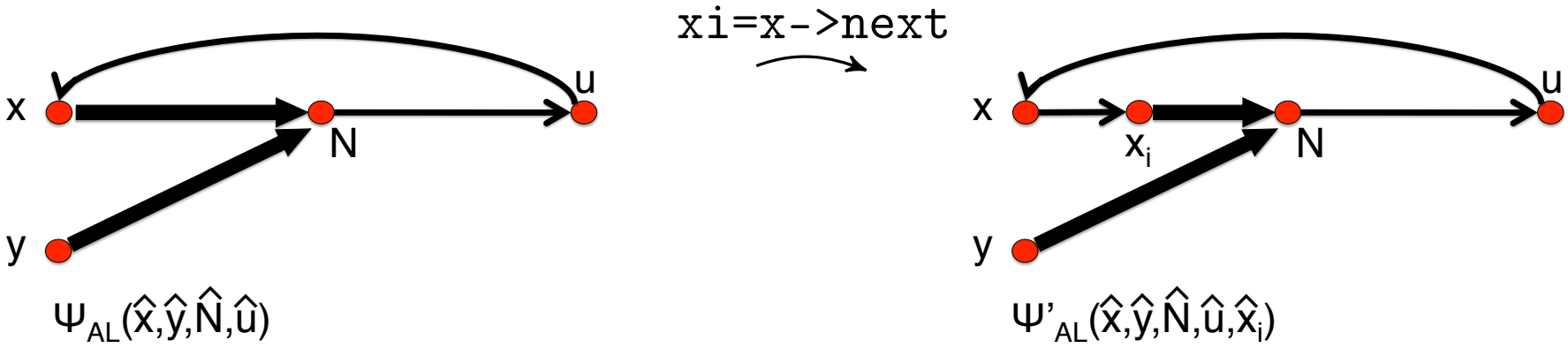
Abstract Transformers: unfold



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

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

Abstract Transformers: unfold

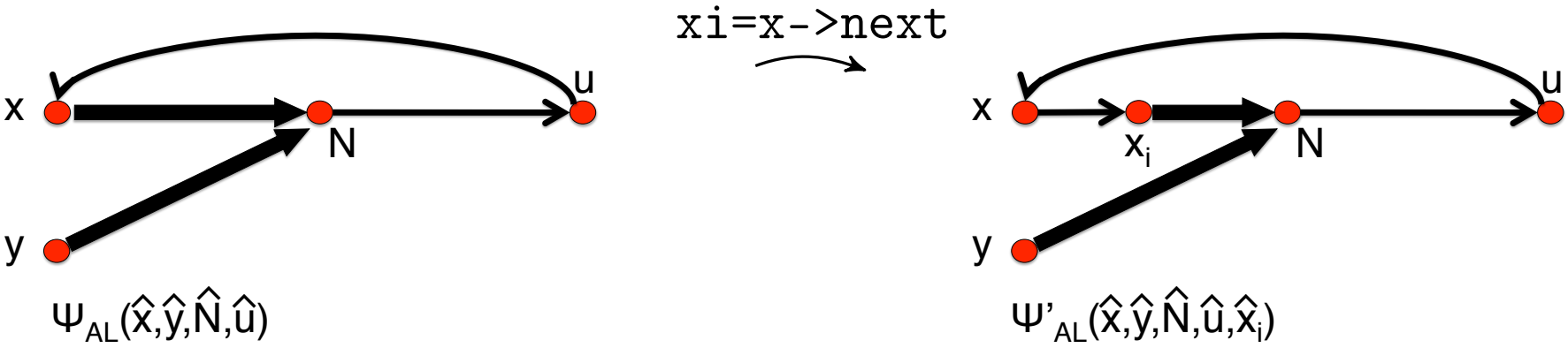


Easy case: only $g_{all}(i,\hat{n}) ::= 0 < i < |\hat{n}|$

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

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

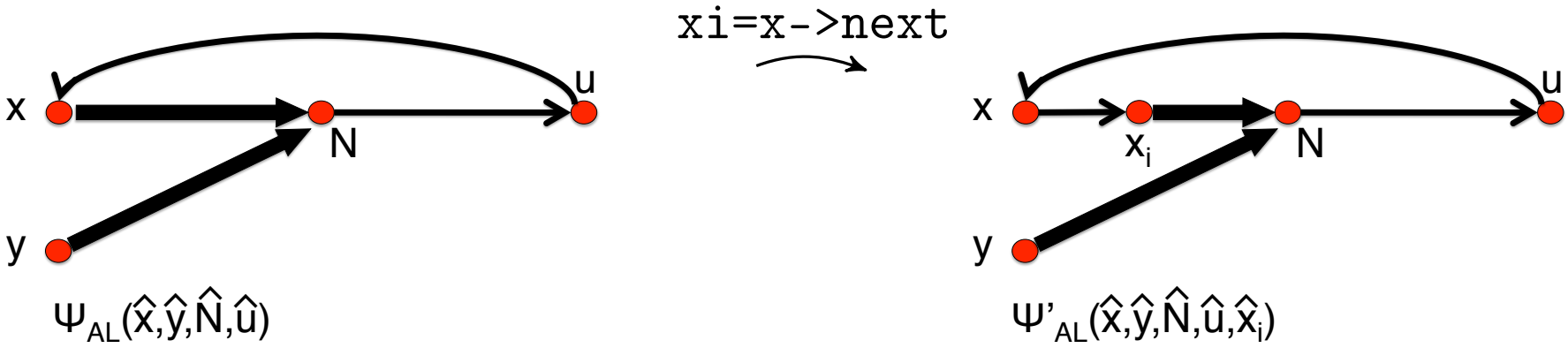
Abstract Transformers: unfold



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

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

Abstract Transformers: unfold

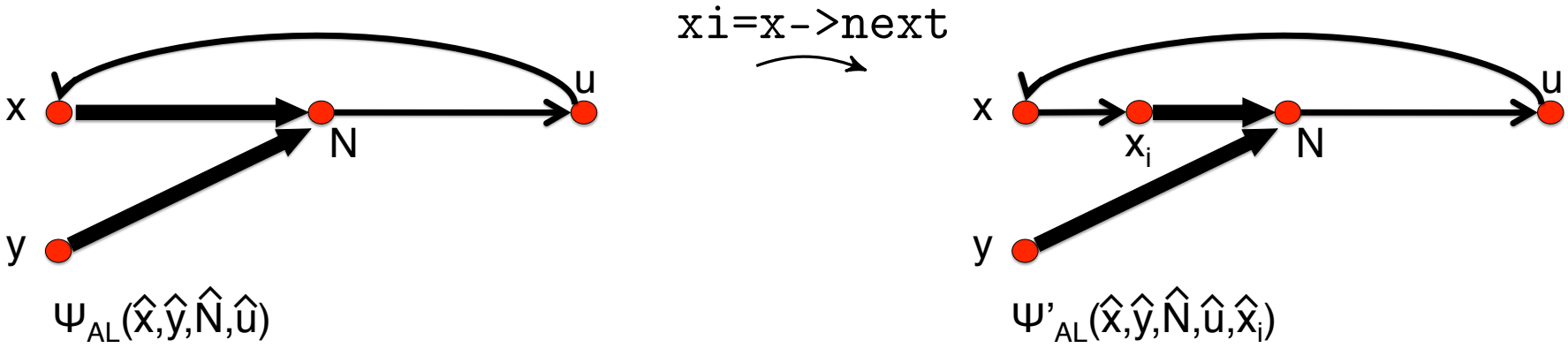


Less easy case: only g_{all}, $g_{\leqslant}(i_1, i_2, \hat{n}) ::= 0 < i_1 \leqslant i_2 < |\hat{n}|$

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

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

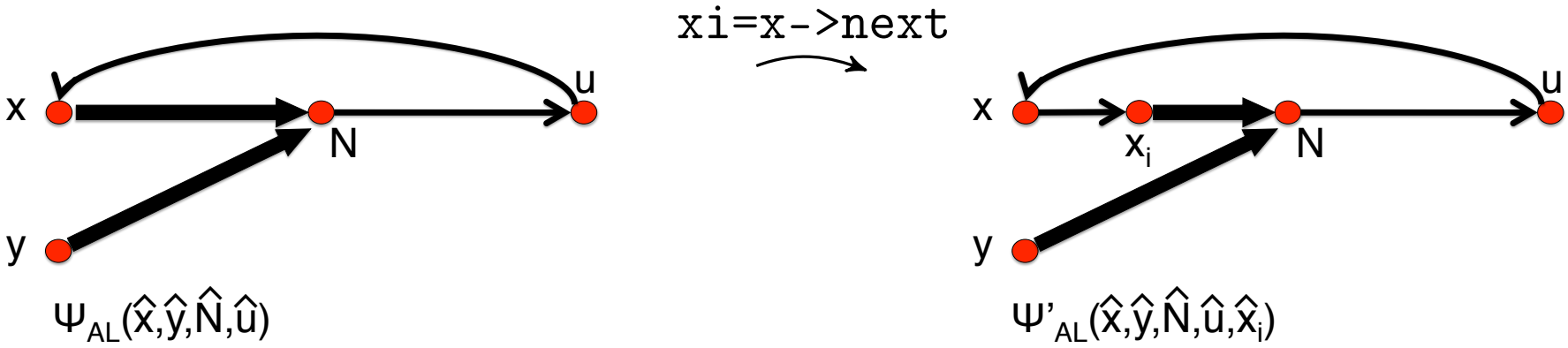
Abstract Transformers: unfold



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

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

Abstract Transformers: unfold



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

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

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

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

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

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

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

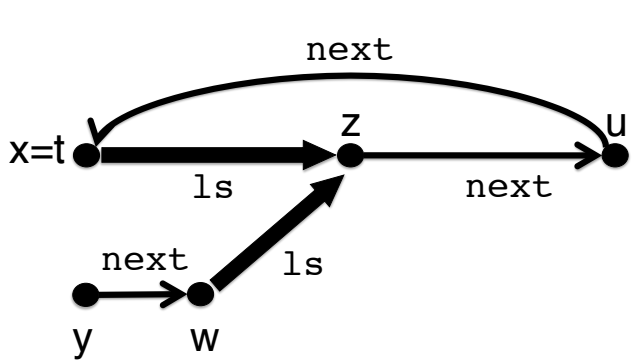
Satisfiability and entailment in SLAD are undecidable.

Theorem

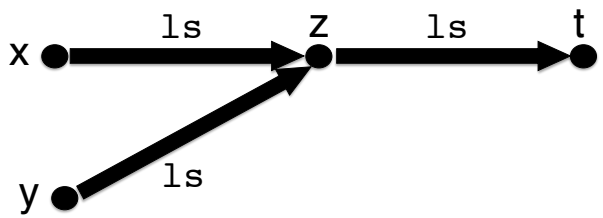
$\exists$ a sound procedure for checking satisfiability and entailment, which is complete when Array Logic has only $\leqslant$-constraints in $g(\overrightarrow{i})$.

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.



$\stackrel{?}{\Longrightarrow}$



$\widehat{x}[0] \geq 2 \wedge \mathrm{sorted}_{<}(\widehat{x}) \wedge$

$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge$

$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathrm{succ}(\widehat{w})$

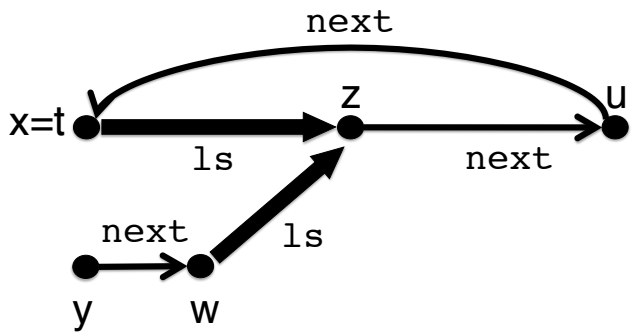
$\widehat{x}[0] \geq 1 \wedge \mathrm{sorted}_{\leqslant}(\widehat{x}) \wedge$

$\mathrm{succ}(\widehat{z}) \ \wedge$

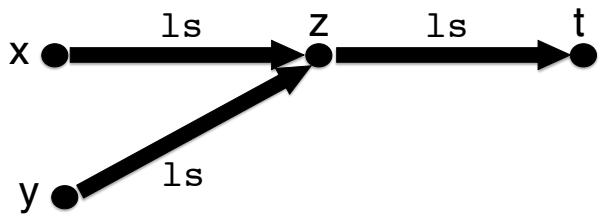
$\mathrm{all}_{\geq 1}(\widehat{y})$

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.



$\stackrel{?}{\Longrightarrow}$



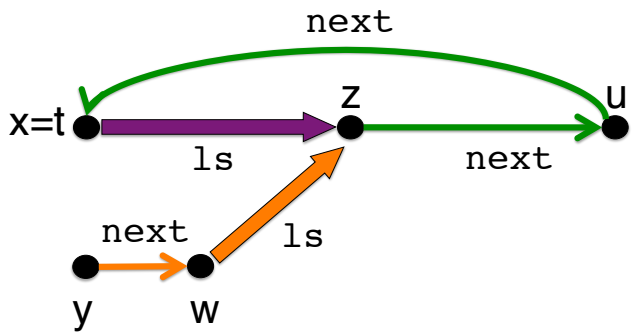
$$\widehat{x}[0] \geq 2 \wedge \mathrm{sorted}_{<}(\widehat{x}) \wedge \stackrel{?}{\Longrightarrow} \widehat{x}[0] \geq 1 \wedge \mathrm{sorted}_{\leqslant}(\widehat{x}) \wedge$$

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

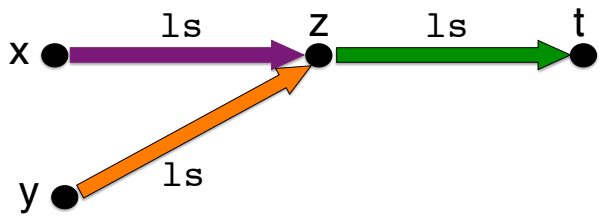
$$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathrm{succ}(\widehat{w}) \stackrel{?}{\Longrightarrow} \mathrm{all}_{\geq 1}(\widehat{y})$$

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.



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



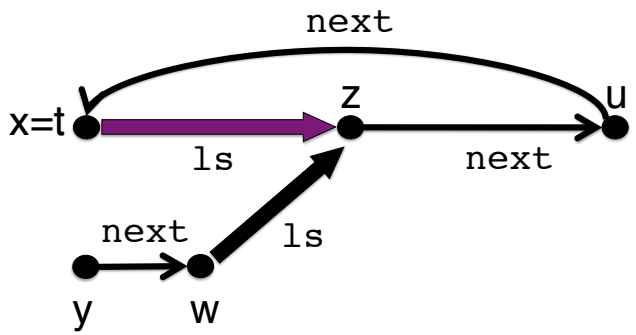
$$\widehat{x}[0] \geq 2 \wedge \mathrm{sorted}_{<}(\widehat{x}) \wedge \overset{?}{\Longrightarrow} \widehat{x}[0] \geq 1 \wedge \mathrm{sorted}_{\leqslant}(\widehat{x}) \wedge$$

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

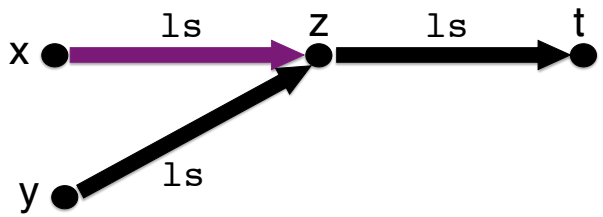
$$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathrm{succ}(\widehat{w}) \overset{?}{\Longrightarrow} \mathrm{all}_{\geq 1}(\widehat{y})$$

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.



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



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

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

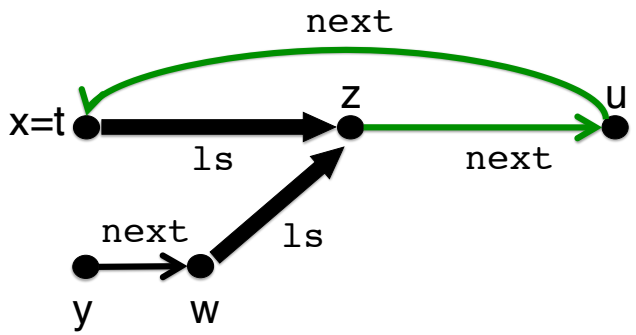
$$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathrm{succ}(\widehat{w}) \quad \overset{?}{\Longrightarrow} \quad \mathrm{all}_{\geq 1}(\widehat{y})$$

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

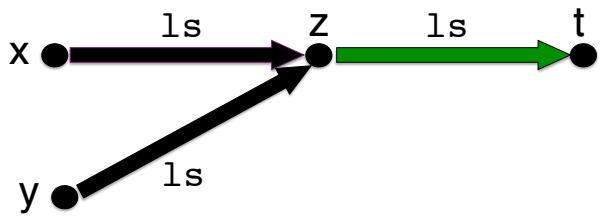
$$\mathrm{sorted}_{\leqslant}(\widehat{x}) \equiv \forall i_1, i_2.\ 0 \leqslant i_1 < i_2 < \mathrm{len}(\widehat{x}) \Longrightarrow \widehat{x}[i_1] \leqslant \widehat{x}[i_2]$$

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.



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



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

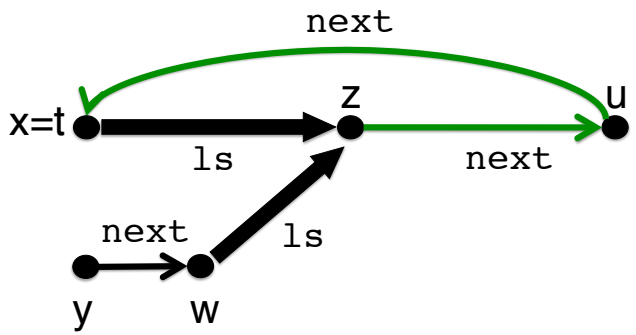
$$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge \quad \overset{?}{\Longrightarrow} \quad \mathrm{succ}(\widehat{z}) \ \wedge$$

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

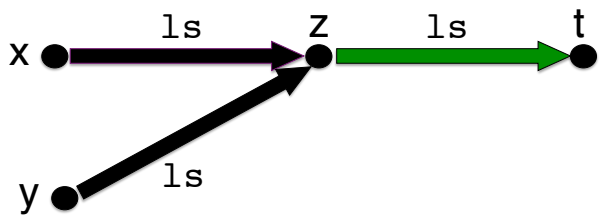
$$\mathrm{succ}(\widehat{z}) \quad \equiv \quad \forall i_1, i_2.\ 0 \leqslant i_1, i_2 < \mathrm{len}(\widehat{z}) \wedge i_1 + 1 = i_2 \Longrightarrow \widehat{z}[i_1] + 1 = \widehat{z}[i_2]$$

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.



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



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

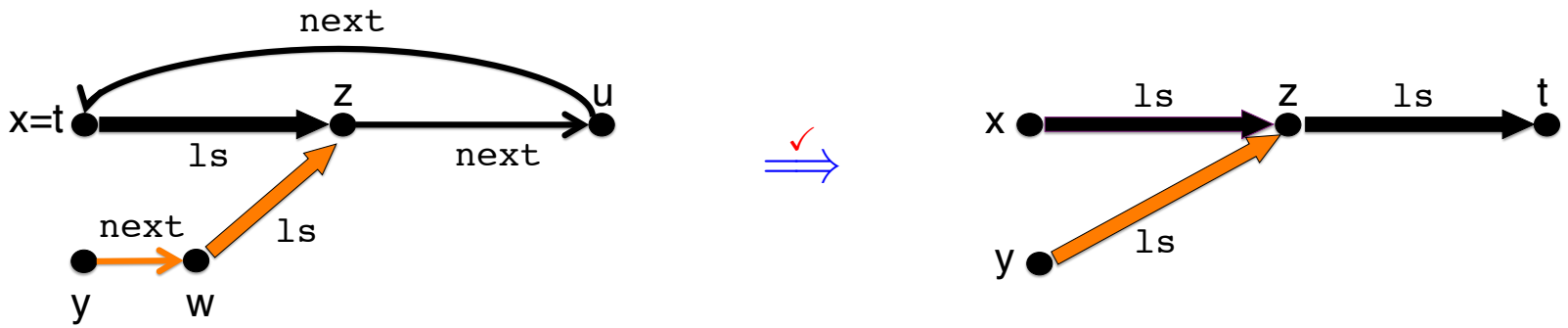
$$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge \mathrm{succ}(\widehat{z \cdot u}) \wedge \quad \overset{\checkmark}{\Longrightarrow} \quad \mathrm{succ}(\widehat{z}) \ \wedge$$

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

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

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.



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

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

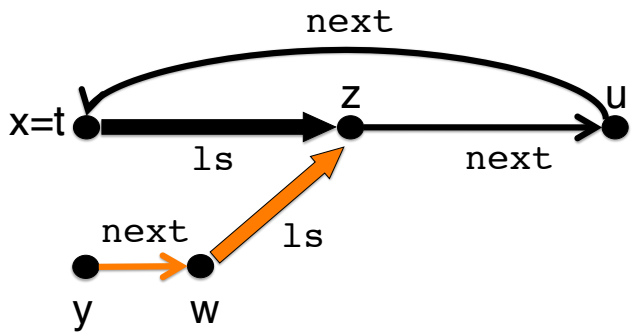
$$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathrm{succ}(\widehat{w}) \overset{?}{\Longrightarrow} \mathrm{all}_{\geq 1}(\widehat{y})$$

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

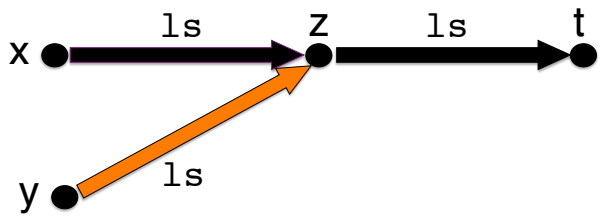
$$\mathrm{all}_{\geq 1}(\widehat{y}) \equiv \forall i.\ 0 \leqslant i < \mathrm{len}(\widehat{y}) \Longrightarrow \widehat{y}[i] \geq 1$$

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.



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



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

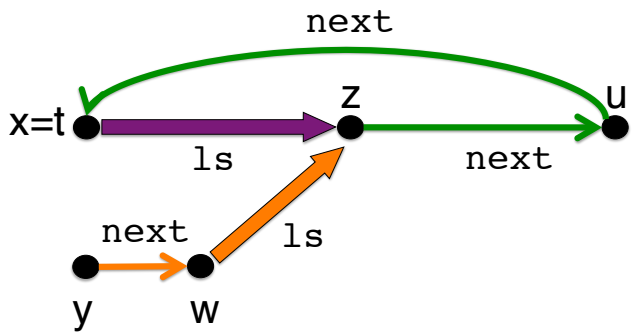
$$\widehat{z}[0] = 0 \wedge \widehat{u}[0] = 1 \wedge \mathrm{succ}(\widehat{z \cdot u}) \wedge \overset{\checkmark}{\Longrightarrow} \mathrm{succ}(\widehat{z}) \wedge$$

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

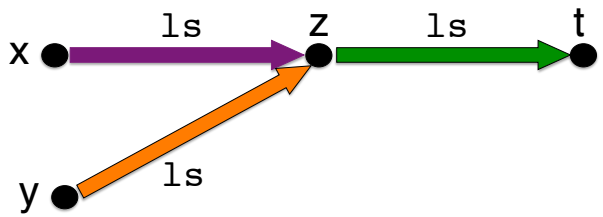
$$\mathrm{succ}(\widehat{w}) \equiv \forall i_1, i_2.\ 0 \leqslant i_1, i_2 < \mathrm{len}(\widehat{w}) \wedge i_1 + 1 = i_2 \Longrightarrow \widehat{w}[i_1] + 1 = \widehat{w}[i_2]$$
$$\mathrm{all}_{\geq 1}(\widehat{y}) \equiv \forall i.\ 0 \leqslant i < \mathrm{len}(\widehat{y}) \Longrightarrow \widehat{y}[i]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

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.



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



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

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

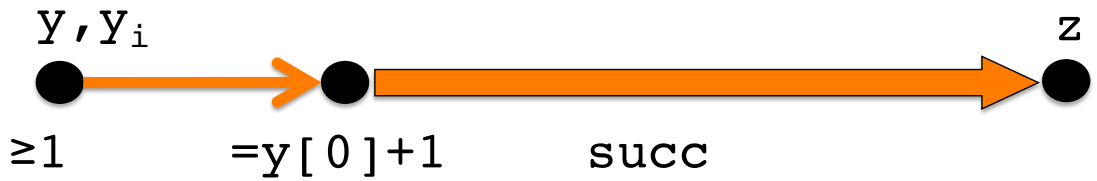
$$\widehat{y}[0] + 1 = \widehat{w}[0] \geq 2 \wedge \mathrm{succ}(\widehat{w}) \overset{\checkmark}{\Longrightarrow} \mathrm{all}_{\geq 1}(\widehat{y})$$
$$\wedge\ \mathrm{all}_{\geq 1}(\widehat{y \cdot w})$$

Saturating SLAD Formulas

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.

Saturating SLAD Formulas

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.

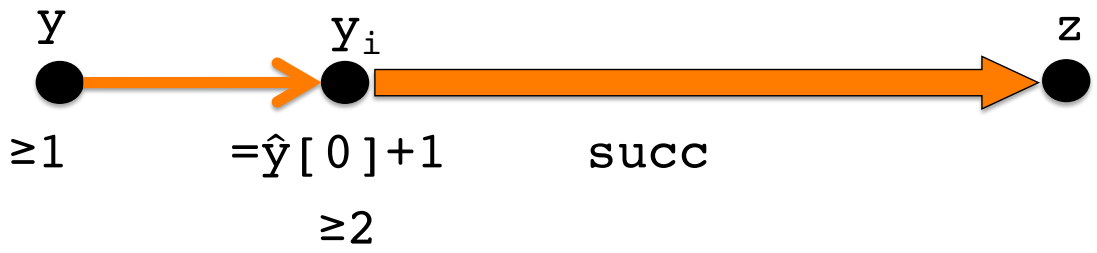



```
/* @assume: y->w * ls(w,z),
            ŷ[0]+1=ŵ[0]≥2, succ(ŵ) */
y_i=y;
/* @inv: ls(y,y_i) * ls(y_i,z), …
         ∀i 0≤i<len(y.w) => ?    */
while(y_i!= z)
  y_i=y_i->next;
```

in $\mathbf{G}$, used in $\text{all}_{\geq 1}$

Saturating SLAD Formulas

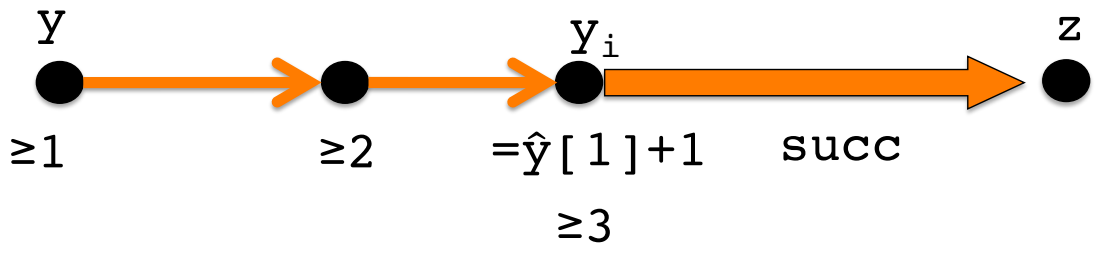
Main idea: Do analysis presented before on each list segment using as $\mathcal{A}_{\mathbb{H}}$ the SLAD formulas! The guards in universal formulas, size and data constraints fix the abstract domain of integer words.




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

Saturating SLAD Formulas

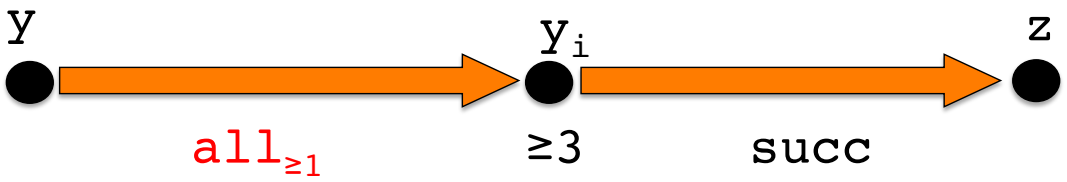
Main idea: Do analysis presented before on each list segment using as $\mathcal{A}_{\mathbb{H}}$ the SLAD formulas! The guards in universal formulas, size and data constraints fix the abstract domain of integer words.




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

Saturating SLAD Formulas

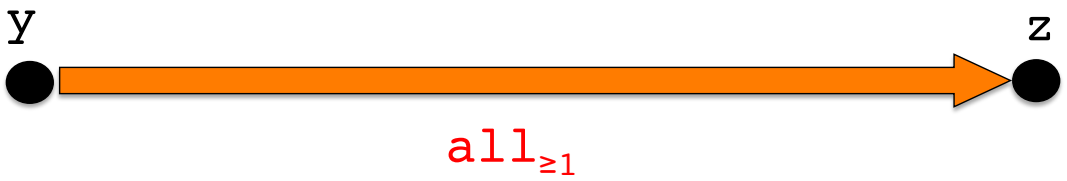
Main idea: Do analysis presented before on each list segment using as $\mathcal{A}_{\mathbb{H}}$ the SLAD formulas! The guards in universal formulas, size and data constraints fix the abstract domain of integer words.




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

Saturating SLAD Formulas

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) */
y_i=y;
/* @inv: ls(y,y_i) * ls(y_i,z), …
         ∀i 0≤i<len(ŷ)=> ŷ[i]≥1 */
while(y_i!= z)
  y_i=y_i->next;
```

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.