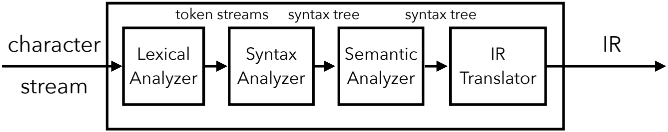
COSE312: Compilers

Lecture 13 — Semantic Analysis (1)

Hakjoo Oh
2025 Spring

Semantic Analysis

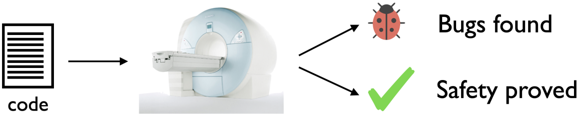


Semantic analysis aims to statically detect runtime errors, e.g.,

```
int a[10] = {...};
int x = rand();
int y = 1;
if (x > 0) {
  if (x < 15) {
    if (x < 10) a[x] = "hello" + y;
    a[x] = 1;
  }
} else {
  y = y / x;
}
```

Underlying Technology: Software Analysis

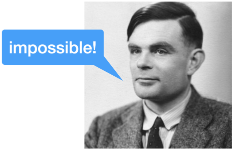
- Technology for catching bugs or proving correctness of software



- Widely used in software industry

A Hard Limit

- The Halting problem is not computable



Alan Turing (1936)

230 A. M. TURING [Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

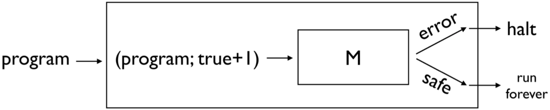
The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.

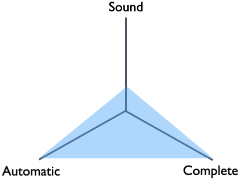
- If exact analysis is possible, we can solve the Halting problem



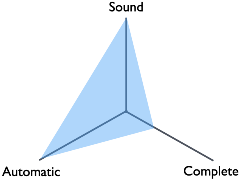
- Rice's theorem (1951): any non-trivial semantic property of a program is undecidable

Tradeoff

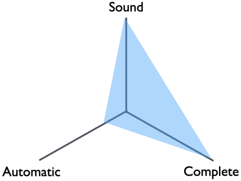
- Three desirable properties
  - Soundness: all program behaviors are captured
  - Completeness: only program behaviors are captured
  - Automation: without human intervention
- Achieving all of them is generally infeasible



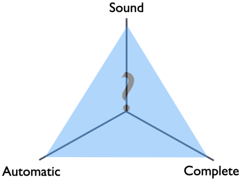
dynamic analysis



static analysis



verification



Principles of Static Analysis

$$\mathbf{30 \times 12 + 11 \times 9 = ?}$$

- Dynamic analysis (testing): 459
- Static analysis: a variety of answers
  - "integer", "odd integer", "positive integer", "$\mathbf{400 \leq n \leq 500}$", etc
- Static analysis process:
  1. Choose abstract value (domain), e.g., $\hat{V} = \{\top, e, o, \bot\}$
  2. Define abstract semantics in terms of abstract values:

$\hat{\times}$	$\top$	e	o	$\bot$
$\top$				
e				
o				
$\bot$				

$\hat{+}$	$\top$	e	o	$\bot$
$\top$				
e				
o				
$\bot$				

  3. "Execute" the program:

$$e \mathbin{\hat{\times}} e \mathbin{\hat{+}} o \mathbin{\hat{\times}} o = o$$

Principles of Static Analysis

- By contrast to testing, static analysis can prove the absence of bugs:

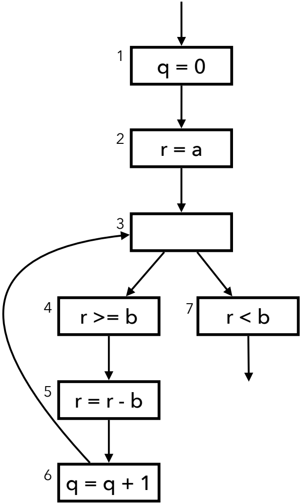
```
void f (int x)  {
    y = x * 12 + 9 * 11;
    assert (y % 2 == 1);
}
```

- Instead, static analysis may produce false alarms:

```
void f (int x)  {
    y = x + x;
    assert (y % 2 == 0);
}
```

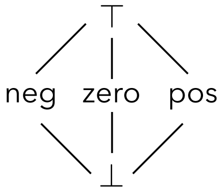
Example Program

```
// a >= 0, b >= 0
q = 0;
r = a;
while (r >= b) {
  r = r - b;
  q = q + 1;
}
assert(q >= 0);
assert(r >= 0);
```



A Simple Sign Analysis

- Abstract domain:

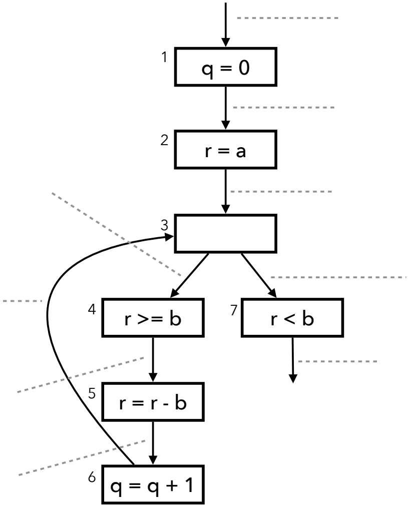


- Abstract semantics:

$+/-$	top	neg	zero	pos	bot
top					
neg					
zero					
pos					
bot					

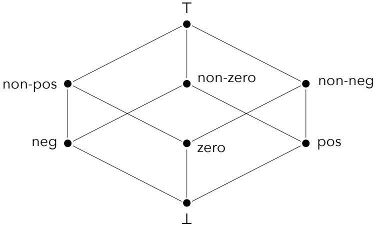
$\times$	top	neg	zero	pos	bot
top					
neg					
zero					
pos					
bot					

Fixed Point Computation



An Extended Sign Analysis

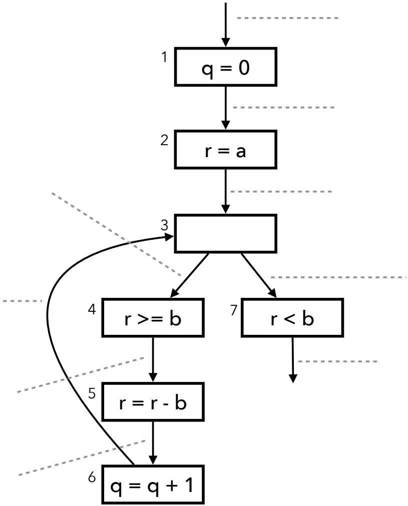
- Abstract domain:



- Abstract semantics:

+	top	neg	zero	pos	non-pos	non-zero	non-neg	bot
top								
neg								
zero								
pos								
non-pos								
non-zero								
non-neg								
bot								

Fixed Point Computation



An Abstract Semantics of **While**

The **While** language:

- Syntax:

$$\begin{array}{rcl} a & \rightarrow & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\ b & \rightarrow & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\ c & \rightarrow & x := a \mid \texttt{skip} \mid c_1; c_2 \mid \texttt{if}\ b\ c_1\ c_2 \mid \texttt{while}\ b\ c \end{array}$$

- (Concrete) Semantics

$$\mathcal{A}[\![\ a\ ]\!] : \mathbf{State} \rightarrow \mathbb{Z}$$

$$\mathcal{B}[\![\ b\ ]\!] : \mathbf{State} \rightarrow \mathbf{T}$$

$$\mathcal{C}[\![\ c\ ]\!] : \mathbf{State} \hookrightarrow \mathbf{State}$$

Abstract Values: Integers

- Concrete integers ($\mathbb{Z}$) are abstracted by the complete lattice $(\widehat{\mathbb{Z}}, \sqsubseteq_{\widehat{\mathbb{Z}}})$:

$$\widehat{\mathbb{Z}} = \{\top_{\widehat{\mathbb{Z}}}, \bot_{\widehat{\mathbb{Z}}}, \textbf{Pos}, \textbf{Neg}, \textbf{Zero}\}$$

$$\hat{a} \sqsubseteq_{\widehat{\mathbb{Z}}} \hat{b} \iff \hat{a} = \hat{b} \lor \hat{a} = \bot_{\widehat{\mathbb{Z}}} \lor \hat{b} = \top_{\widehat{\mathbb{Z}}}$$

- An abstract integer denotes a set of integers.
  - Abstraction function: $\alpha_{\widehat{\mathbb{Z}}} : \mathcal{P}(\mathbb{Z}) \to \widehat{\mathbb{Z}}$
  - Concretization function: $\gamma_{\widehat{\mathbb{Z}}} : \widehat{\mathbb{Z}} \to \mathcal{P}(\mathbb{Z})$

$$\begin{aligned}
\alpha_{\widehat{\mathbb{Z}}}(\emptyset) &= \bot_{\widehat{\mathbb{Z}}} \\
\alpha_{\widehat{\mathbb{Z}}}(S) &= \textbf{Pos} \quad (\forall n \in S.\ n > 0) \\
\alpha_{\widehat{\mathbb{Z}}}(S) &= \textbf{Neg} \quad (\forall n \in S.\ n < 0) \\
\alpha_{\widehat{\mathbb{Z}}}(S) &= \textbf{Zero} \quad (S = \{0\}) \\
\alpha_{\widehat{\mathbb{Z}}}(S) &= \top_{\widehat{\mathbb{Z}}} \quad (\text{otherwise})
\end{aligned}
\qquad
\begin{aligned}
\gamma_{\widehat{\mathbb{Z}}}(\bot_{\widehat{\mathbb{Z}}}) &= \emptyset \\
\gamma_{\widehat{\mathbb{Z}}}(\top_{\widehat{\mathbb{Z}}}) &= \mathbb{Z} \\
\gamma_{\widehat{\mathbb{Z}}}(\mathbf{Pos}) &= \{n \in \mathbb{Z} \mid n > 0\} \\
\gamma_{\widehat{\mathbb{Z}}}(\mathbf{Neg}) &= \{n \in \mathbb{Z} \mid n < 0\} \\
\gamma_{\widehat{\mathbb{Z}}}(\mathbf{Zero}) &= \{0\}
\end{aligned}$$

- Join (least upper bound) and meet (greatest lower bound):

$$\begin{aligned}
\hat{a} \sqcup_{\widehat{\mathbb{Z}}} \hat{b} &= \hat{a}\ (\hat{b} \sqsubseteq_{\widehat{\mathbb{Z}}} \hat{a}) \\
\hat{a} \sqcup_{\widehat{\mathbb{Z}}} \hat{b} &= \hat{b}\ (\hat{a} \sqsubseteq_{\widehat{\mathbb{Z}}} \hat{b}) \\
\hat{a} \sqcup_{\widehat{\mathbb{Z}}} \hat{b} &= \top_{\widehat{\mathbb{Z}}}
\end{aligned}
\qquad
\begin{aligned}
\hat{a} \sqcap_{\widehat{\mathbb{Z}}} \hat{b} &= \hat{b}\ (\hat{b} \sqsubseteq_{\widehat{\mathbb{Z}}} \hat{a}) \\
\hat{a} \sqcap_{\widehat{\mathbb{Z}}} \hat{b} &= \hat{a}\ (\hat{a} \sqsubseteq_{\widehat{\mathbb{Z}}} \hat{b}) \\
\hat{a} \sqcap_{\widehat{\mathbb{Z}}} \hat{b} &= \bot_{\widehat{\mathbb{Z}}}
\end{aligned}$$

Abstract Values: Booleans

- The truth values $\mathbf{T} = \{true, false\}$ are abstracted by $(\widehat{\mathbf{T}}, \sqsubseteq_{\widehat{\mathbf{T}}})$:

$$\widehat{\mathbf{T}} = \{\top_{\widehat{\mathbf{T}}}, \bot_{\widehat{\mathbf{T}}}, \widehat{true}, \widehat{false}\}$$

$$\hat{b}_1 \sqsubseteq_{\widehat{\mathbf{T}}} \hat{b}_2 \iff \hat{b}_1 = \hat{b}_2 \lor \hat{b}_1 = \bot_{\widehat{\mathbf{T}}} \lor \hat{b}_2 = \top_{\widehat{\mathbf{T}}}$$

- An abstract boolean denotes a set of concrete booleans:

$$\begin{array}{rcl} \alpha_{\widehat{\mathbf{T}}} & : & \mathcal{P}(\mathbf{T}) \to \widehat{\mathbf{T}} \\ \alpha_{\widehat{\mathbf{T}}}(\emptyset) & = & \bot_{\widehat{\mathbf{T}}} \\ \alpha_{\widehat{\mathbf{T}}}(\{true\}) & = & \widehat{true} \\ \alpha_{\widehat{\mathbf{T}}}(\{false\}) & = & \widehat{false} \\ \alpha_{\widehat{\mathbf{T}}}(\mathbf{T}) & = & \top_{\widehat{\mathbf{T}}} \end{array} \qquad \begin{array}{rcl} \gamma_{\widehat{\mathbf{T}}} & : & \widehat{\mathbf{T}} \to \mathcal{P}(\mathbf{T}) \\ \gamma_{\widehat{\mathbf{T}}}(\bot_{\widehat{\mathbf{T}}}) & = & \emptyset \\ \gamma_{\widehat{\mathbf{T}}}(\widehat{true}) & = & \{true\} \\ \gamma_{\widehat{\mathbf{T}}}(\widehat{false}) & = & \{false\} \\ \gamma_{\widehat{\mathbf{T}}}(\top_{\widehat{\mathbf{T}}}) & = & \mathbf{T} \end{array}$$

- Join and meet:

$$\begin{array}{l} \hat{a} \sqcup_{\widehat{\mathbf{T}}} \hat{b} = \hat{a} \ (\hat{b} \sqsubseteq_{\widehat{\mathbf{T}}} \hat{a}) \\ \hat{a} \sqcup_{\widehat{\mathbf{T}}} \hat{b} = \hat{b} \ (\hat{a} \sqsubseteq_{\widehat{\mathbf{T}}} \hat{b}) \\ \hat{a} \sqcup_{\widehat{\mathbf{T}}} \hat{b} = \top_{\widehat{\mathbf{T}}} \end{array} \qquad \begin{array}{l} \hat{a} \sqcap_{\widehat{\mathbf{T}}} \hat{b} = \hat{b} \ (\hat{b} \sqsubseteq_{\widehat{\mathbf{T}}} \hat{a}) \\ \hat{a} \sqcap_{\widehat{\mathbf{T}}} \hat{b} = \hat{a} \ (\hat{a} \sqsubseteq_{\widehat{\mathbf{T}}} \hat{b}) \\ \hat{a} \sqcap_{\widehat{\mathbf{T}}} \hat{b} = \bot_{\widehat{\mathbf{T}}} \end{array}$$

Abstract States

- Concrete states ($\mathbf{State}$) are abstracted by $(\widehat{\mathbf{State}}, \sqsubseteq_{\widehat{\mathbf{State}}})$:

$$\widehat{\mathbf{State}} = \mathbf{Var} \to \widehat{\mathbb{Z}}$$

$$\hat{s}_1 \sqsubseteq_{\widehat{\mathbf{State}}} \hat{s}_2 \iff \forall x \in \mathbf{Var}.\ \hat{s}_1(x) \sqsubseteq_{\widehat{\mathbb{Z}}} \hat{s}_2(x).$$

- An abstract state denotes a set of concrete states:

$$\begin{array}{rcl} \alpha_{\widehat{\mathbf{State}}} & : & \mathcal{P}(\mathbf{State}) \to \widehat{\mathbf{State}} \\ \alpha_{\widehat{\mathbf{State}}}(S) & = & \lambda x.\ \bigsqcup_{s \in S} \alpha_{\widehat{\mathbb{Z}}}(\{s(x)\}) \\ \gamma_{\widehat{\mathbf{State}}} & = & \widehat{\mathbf{State}} \to \mathcal{P}(\mathbf{State}) \\ \gamma_{\widehat{\mathbf{State}}}(\hat{s}) & = & \{s \in \mathbf{State} \mid \forall x \in \mathbf{Var}.\ s(x) \in \gamma_{\widehat{\mathbb{Z}}}(\hat{s}(x))\} \end{array}$$

- Join and meet:

$$\hat{s_1} \sqcup_{\widehat{\mathbf{State}}} \hat{s_2} = \lambda x.\ \hat{s}_1(x) \sqcup_{\widehat{\mathbb{Z}}} \hat{s}_2(x)$$
$$\hat{s_1} \sqcap_{\widehat{\mathbf{State}}} \hat{s_2} = \lambda x.\ \hat{s}_1(x) \sqcap_{\widehat{\mathbb{Z}}} \hat{s}_2(x)$$

Abstract Semantics

$$
\begin{array}{rcl}
\widehat{\mathcal{A}}[\![a]\!] & : & \widehat{\text{State}} \to \widehat{\mathbb{Z}} \\
\widehat{\mathcal{A}}[\![n]\!](\hat{s}) & = & \alpha_{\widehat{\mathbb{Z}}}(\{n\}) \\
\widehat{\mathcal{A}}[\![x]\!](\hat{s}) & = & \hat{s}(x) \\
\widehat{\mathcal{A}}[\![a_1 + a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) +_{\widehat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![a_1 \star a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) \star_{\widehat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{A}}[\![a_1 - a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) -_{\widehat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s})
\end{array}
$$

$$
\begin{array}{rcl}
\widehat{\mathcal{B}}[\![b]\!] & : & \widehat{\text{State}} \to \widehat{\mathbf{T}} \\
\widehat{\mathcal{B}}[\![\texttt{true}]\!](\hat{s}) & = & \widehat{\mathit{true}} \\
\widehat{\mathcal{B}}[\![\texttt{false}]\!](\hat{s}) & = & \widehat{\mathit{false}} \\
\widehat{\mathcal{B}}[\![a_1 = a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) =_{\widehat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![a_1 \le a_2]\!](\hat{s}) & = & \widehat{\mathcal{A}}[\![a_1]\!](\hat{s}) \le_{\widehat{\mathbb{Z}}} \widehat{\mathcal{A}}[\![a_2]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![\neg b]\!](\hat{s}) & = & \neg_{\widehat{\mathbf{T}}} \widehat{\mathcal{B}}[\![b]\!](\hat{s}) \\
\widehat{\mathcal{B}}[\![b_1 \land b_2]\!](\hat{s}) & = & \widehat{\mathcal{B}}[\![b_1]\!](\hat{s}) \land_{\widehat{\mathbf{T}}} \widehat{\mathcal{B}}[\![b_2]\!](\hat{s})
\end{array}
$$

Abstract Semantics

$$\begin{array}{rcl}
\widehat{\mathcal{C}}[\![c]\!] & : & \widehat{\mathbf{State}} \to \widehat{\mathbf{State}} \\
\widehat{\mathcal{C}}[\![x := a]\!] & = & \lambda \hat{s}.\hat{s}[x \mapsto \widehat{\mathcal{A}}[\![a]\!](\hat{s})] \\
\widehat{\mathcal{C}}[\![\texttt{skip}]\!] & = & \mathsf{id} \\
\widehat{\mathcal{C}}[\![c_1; c_2]\!] & = & \widehat{\mathcal{C}}[\![c_2]\!] \circ \widehat{\mathcal{C}}[\![c_1]\!] \\
\widehat{\mathcal{C}}[\![\texttt{if}\ b\ c_1\ c_2]\!] & = & \widehat{\mathsf{cond}}(\widehat{\mathcal{B}}[\![b]\!], \widehat{\mathcal{C}}[\![c_1]\!], \widehat{\mathcal{C}}[\![c_2]\!]) \\
\widehat{\mathcal{C}}[\![\texttt{while}\ b\ c]\!] & = & \mathit{fix}\widehat{F}
\end{array}$$

where

$$\widehat{\mathsf{cond}}(\hat{f}, \hat{g}, \hat{h}) = \lambda \hat{s}. \begin{cases} \bot_{\widehat{\mathbf{State}}} & \cdots \hat{f}(\hat{s}) = \bot_{\widehat{\mathbf{T}}} \\ \hat{g}(\hat{s}) & \cdots \hat{f}(\hat{s}) = \widehat{\mathit{true}} \\ \hat{h}(\hat{s}) & \cdots \hat{f}(\hat{s}) = \widehat{\mathit{false}} \\ \hat{g}(\hat{s}) \sqcup_{\widehat{\mathbf{State}}} \hat{h}(\hat{s}) & \cdots \hat{f}(\hat{s}) = \top_{\widehat{\mathbf{T}}} \end{cases}$$

$$\widehat{F}(\hat{g}) = \widehat{\mathsf{cond}}(\widehat{\mathcal{B}}[\![b]\!], \hat{g} \circ \widehat{\mathcal{C}}[\![c]\!], \mathsf{id})$$

Example: while $\neg(x = 0)$ skip

- $\widehat{F} : (\widehat{\text{State}} \to \widehat{\text{State}}) \to (\widehat{\text{State}} \to \widehat{\text{State}})$:

$$\widehat{F}(\hat{g}) = \widehat{\text{cond}}(\widehat{\mathcal{B}}[\![\neg(x = 0)]\!], \hat{g} \circ \widehat{\mathcal{C}}[\![\text{skip}]\!], \text{id}) = \widehat{\text{cond}}(\lambda\hat{s}.\hat{s}(x) \neq_{\widehat{\mathbb{Z}}} \text{Zero}, \hat{g}, \text{id})$$

$$= \lambda\hat{s}. \begin{cases} \bot & \text{if } (\hat{s}(x) \neq_{\widehat{\mathbb{Z}}} \text{Zero}) = \bot \\ \hat{g}(\hat{s}) & \text{if } (\hat{s}(x) \neq_{\widehat{\mathbb{Z}}} \text{Zero}) = \widehat{true} \\ \hat{s} & \text{if } (\hat{s}(x) \neq_{\widehat{\mathbb{Z}}} \text{Zero}) = \widehat{false} \\ \hat{g}(\hat{s}) \sqcup \hat{s} & \text{if } (\hat{s}(x) \neq_{\widehat{\mathbb{Z}}} \text{Zero}) = \top \end{cases}$$

$$= \lambda\hat{s}. \begin{cases} \bot & \text{if } \hat{s}(x) = \bot \\ \hat{g}(\hat{s}) & \text{if } \hat{s}(x) \in \{\text{Pos}, \text{Neg}\} \\ \hat{s} & \text{if } \hat{s}(x) = \text{Zero} \\ \hat{g}(\hat{s}) \sqcup \hat{s} & \text{if } \hat{s}(x) = \top \end{cases}$$

- $\widehat{\mathcal{C}}[\![\text{while } \neg(x = 0) \text{ skip}]\!] = \bigsqcup_{i \geq 0} \widehat{F}^i(\bot_{\widehat{\text{State}} \to \widehat{\text{State}}})$:

1. $\widehat{g_0} = \bot_{\widehat{\text{State}} \to \widehat{\text{State}}} = \lambda\hat{s}.\bot_{\widehat{\text{State}}}$
2. $\widehat{g_1} = \widehat{F}(\widehat{g_0}) = \lambda\hat{s}. \begin{cases} \bot & \text{if } \hat{s}(x) = \bot \\ \bot & \text{if } \hat{s}(x) \in \{\text{Pos}, \text{Neg}\} \\ \hat{s} & \text{if } \hat{s}(x) = \text{Zero} \\ \hat{s} & \text{if } \hat{s}(x) = \top \end{cases}$
3. $\widehat{g_2} = \widehat{F}(\widehat{g_1}) = \lambda\hat{s}. \begin{cases} \bot & \text{if } \hat{s}(x) = \bot \\ \widehat{g_1}(\hat{s}) = \bot & \text{if } \hat{s}(x) \in \{\text{Pos}, \text{Neg}\} \\ \hat{s} & \text{if } \hat{s}(x) = \text{Zero} \\ \widehat{g_1}(\hat{s}) \sqcup \hat{s} = \hat{s} \sqcup \hat{s} = \hat{s} & \text{if } \hat{s}(x) = \top \end{cases} = \widehat{g_1}$

Static Analysis of Control-Flow Graphs

- Programs in **While** can be represented by control-flow graph $G = (N, \hookrightarrow)$, where each node $n \in N$ contains a command (denoted $cmd(n)$) defined as follows:

$$c \to x := a \mid assume(b) \mid skip$$

- Abstract semantics (transfer function) $\hat{f}_n : \widehat{\mathbf{State}} \to \widehat{\mathbf{State}}$:

$$\hat{f}_n(\hat{s}) = \begin{cases} \hat{s} & \text{if } cmd(n) = skip \\ \hat{s}[x \mapsto \hat{\mathcal{A}}[\![a]\!](\hat{s})] & \text{if } x := a \\ \hat{s} & \text{if } assume(b), \widehat{true} \sqsubseteq \hat{\mathcal{B}}[\![b]\!](\hat{s}) \\ \bot & \text{if } assume(b), \widehat{false} \sqsupseteq \hat{\mathcal{B}}[\![b]\!](\hat{s}) \end{cases}$$

- The analysis is to compute the least fixed point of the function:

$$\hat{F} : (N \to \widehat{\mathbf{State}}) \to (N \to \widehat{\mathbf{State}})$$

$$\hat{F}(X) = \lambda n.\ \hat{f}_n\Big(\bigsqcup_{n' \hookrightarrow n} X(n')\Big)$$

Fixed Point Computation

- Tabulation algorithm:

$$\bigsqcup_{i\geq 0} \widehat{F}^i(\lambda n.\bot) = \begin{array}{l} X := X' := \lambda n.\bot \\ \texttt{repeat} \\ \quad X' := X \\ \quad X := X \sqcup \widehat{F}(X) \\ \texttt{until } X \sqsubseteq X' \\ \texttt{return } X' \end{array}$$

- Worklist algorithm:

$$\begin{array}{l} W := N \\ X := \lambda n.\bot \\ \texttt{repeat} \\ \quad n := \textsf{choose}(W) \\ \quad W := W \setminus \{n\} \\ \quad s := \hat{f}_n(\bigsqcup_{n' \hookrightarrow n} X(n')) \\ \quad \texttt{if } s \not\sqsubseteq X(n) \\ \quad\quad X(n) := X(n) \sqcup s \\ \quad\quad W := W \cup \{n' \in N \mid n \hookrightarrow n'\} \\ \texttt{until } W = \emptyset \end{array}$$

Summary

- Approaches to software analysis
- Principles of static analysis
- Simple and extended sign analysis
- Static analysis for **While**