Abstract Interpretation

Dr. Florian Martin
AbsInt

AbsInt.com
Copy Analysis

\[ i := 0; \]
\[ t3 := 0; \]
\[ \text{while } i \leq m \text{ do} \]
\[ \quad (j := 0; \quad t2 := t3; \]
\[ \quad \quad \text{while } j \leq n \text{ do} \]
\[ \quad \quad \quad (t1 := t2 + j; \]
\[ \quad \quad \quad \quad ta := \text{Base}(a) + t1; \]
\[ \quad \quad \quad \quad C(ta) := C(\text{Base}(b) + t1) + \]
\[ \quad \quad \quad \quad \quad C(\text{Base}(c) + t1); \]
\[ \quad \quad \quad \quad j := j + 1 \]
\[ \quad \quad ); \]
\[ t3 := t3 + (n+1); \]
\[ i := i + 1 ); \]
Copy Propagation

\[ i := 0; \]
\[ t3 := 0; \]
\[ \text{while } i \leq m \text{ do} \]
\[ \quad ( j := 0; \quad t2 := t3; \]
\[ \quad \quad \text{while } j \leq n \text{ do} \]
\[ \quad \quad \quad ( t1 := t3 + j; \]
\[ \quad \quad \quad \quad ta := \text{Base}(a) + t1; \]
\[ \quad \quad \quad \quad C(ta) := C(\text{Base}(b) + t1) + \]
\[ \quad \quad \quad \quad \quad C(\text{Base}(c) + t1); \]
\[ \quad \quad \quad \quad j := j + 1 \]
\[ \quad \quad \quad ) ; \]
\[ t3 := t3 + (n+1); \]
\[ i := i + 1 ) ; \]
Copy Propagation Analysis

• The *copy-propagation analysis* determines for each program point the set of variable pairs, for which there is a path from a copy assignment, say \( x := y \), and no further assignments to \( y \) and \( x \).
**TYPE**

- Set of pairs of variables which are copies

\[
\text{VarVarPair} = \text{Var} \times \text{Var}
\]

\[
\text{VarVarPairs} = \text{set}(\text{VarVarPair})
\]
Excursion: Solution Algorithm

• In PAG the worklist is initialized only with the start node

• The programmer must ensure that each node of interest is visited at least once

→ Add extra \( \perp \)
  
  if the \( \perp \) of the lattice has a meaningful value
TYPE

Is {} a meaningful value?
→ add additional lifting

VarVarPairsLifted = lift(VarVarPairs)
PROBLEM Copy Propagation

... path from the assignment to the point ...

direction : forward
carrier : VarVarPairsLifted

... there is a path ...

greatest fixpoint

init : top
combine : glb
Problem (cnt.)

Which variables are copies at the beginning?
None

`init_start : lift({})`
Transfer Function

\[
\begin{align*}
(a,c), (b,e), (f,a) \\
\{ (b,e), (a,b) \}
\end{align*}
\]
TRANSFER

ASSIGN(var, exp) =

let entry <= @ in lift(
    remove(entry, var)
    + gen(var, exp))
SUPPORT

gen::Var* Expression -> VarVarPairs

gen(var, exp) = 
  if expType(exp) = "VAR"
    then {(var, expVar(exp))}
  else {} 
endif
remove::VarVarPairs*Var->VarVarPairs

remove(pairs, var) = 

{ (a,b) | (a,b) in pairs;
  a!=var && b!=var
}

SUPPORT
Abstract Interpretation

• Systematic derivation of correct (program) analyses

• Derivation of new simpler analyses from old analyses
  • Obtain computable analyses
  • Reduce analysis complexity
  • Trade precision vs efficiency

• Provides for analysis correction proves
Correctness

• To deduce the correctness of a program analysis a connection to the formal semantics of the programming language has to be established.
(Denotational) Semantics

- The domain of a computation is a complete lattice
- The semantics of a program construct is given as a function over this domain
- The semantics of the whole program is obtained by the least fixed point of a set of recursive equations
Semantics: Example

- The semantic domain of a Pascal-like language is usually chosen as a mapping from program variables to values (value mapping).

- The semantics of an assignment
  \[ x := \exp \]
  is given described as
  1. the evaluation of \( \exp \) according to the current value mapping
  2. changing the value of \( x \) in the mapping
Semantics: Example

- The semantics of
  \[ \text{if } \text{exp} \text{ then } \text{stmt1 } \text{else } \text{stmt2 } \text{endif} \]
  is constructed from the semantics of \text{stmt1}, if \text{exp} evaluates to \text{true} or of the semantics of \text{stmt2} otherwise

- To describe the semantics of loops or recursions fixed points are needed
AI: Overview

- Use **abstract** values instead of concrete ones
- Each abstract value describes a set of concrete values
- The program is executed/interpreted with abstract values
  - Fixed Points
- Correctness prove
  - By providing a relationship between the abstract and the concrete values
Example: Signed Integers

• Concrete computation:
  Integer multiplication

• Abstract values: Sign={+,−,0,?}

<table>
<thead>
<tr>
<th>⊗</th>
<th>+</th>
<th>−</th>
<th>0</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>−</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>−</td>
<td>−</td>
<td>0</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>0</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>⊕</th>
<th>+</th>
<th>−</th>
<th>0</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>?</td>
<td>+</td>
<td>?</td>
</tr>
<tr>
<td>−</td>
<td>?</td>
<td>−</td>
<td>−</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>+</td>
<td>−</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Signed Integers

• Each abstract element represents a set of concrete elements
  • $+ : \{ x \mid x > 0 \}$
  • $- : \{ x \mid x < 0 \}$
  • $0 : \{ x \mid x = 0 \}$
  • $? : \{ x \mid x > 0 \lor x < 0 \lor x = 0 \}$
Order of $\mathcal{D}_{\text{abs}}$

- $(\mathcal{D}_{\text{abs}}, \sqsubseteq)$ is a complete lattice
- $x \sqsubseteq y$ means:
  - the set of values represented by $x$
  - is a subset of values represented by $y$
Example: Signed Integers

\[ \begin{array}{c}
? \\
\downarrow \\
\pm \\
\downarrow \\
+ \\
\downarrow \\
\perp \\
\end{array} \quad \begin{array}{c}
\downarrow \\
\pm \\
\downarrow \\
0 \\
\end{array} \quad \begin{array}{c}
\downarrow \\
\pm \\
\downarrow \\
+ \\
\end{array} \quad \begin{array}{c}
\downarrow \\
\pm \\
\downarrow \\
- \\
\end{array} \quad \begin{array}{c}
\downarrow \\
\pm \\
\downarrow \\
\perp \\
\end{array} \]
Other Example: Checksum

373 \times 8847 + 12345 \equiv 3312266

4 \times 9 + 6 = 42 \text{ Checksum } 6 \neq 5
Abstract-Concrete Relation

• Abstraction function
  \[ \alpha: \wp(D_{conc}) \rightarrow D_{abs} \]
  computes the (best) description

• Concretion function
  \[ \gamma: D_{abs} \rightarrow \wp(D_{conc}) \]
  computes the set of elements described
Def. Galois Connection

\((D_{\text{conc}}, \alpha, \gamma, D_{\text{abs}})\) is called a Galois Connection iff

1. \(\alpha, \gamma\) are monotone functions

2. \(\gamma \circ \alpha \supseteq \text{id}_{\text{conc}}\)

3. \(\alpha \circ \gamma \subseteq \text{id}_{\text{abs}}\)
Galois Connection

2. \( \forall X \subseteq D_{\text{conc}} : X \subseteq \gamma(\alpha(X)) \)
Galois Connection

3. \( \forall x \in D_{\text{abs}} : \alpha(\gamma(x)) \sqsubseteq x \)
Lifting of Functions

A function \( f: L \rightarrow L \) can be lifted to a function \( f^\#: \wp(L) \rightarrow \wp(L) \) by defining

\[
f^\#(X) := \{ f(x) \mid x \in X \} \]

Local Consistency

A concrete function $f: D_{\text{conc}} \rightarrow D_{\text{conc}}$ and an abstract function $\overline{f}: D_{\text{abs}} \rightarrow D_{\text{abs}}$ are (locally) consistent, if

$$f`(X) \subseteq \gamma(\overline{f}(\alpha(X)))$$

abstract  \hspace{2cm} \overline{f} \hspace{2cm} \overline{f}(\alpha(X)) \hspace{2cm} \gamma \hspace{2cm} \gamma \overline{f}(\alpha(X))$$

concrete  \hspace{2cm} X \hspace{2cm} f` \hspace{2cm} f`(X)
Abstract Interpretation

• An abstract interpretation consists of
  1. A Galois connection \((D_{conc}, \alpha, \gamma, D_{abs})\)
  2. Locally consistent functions \(f, \bar{f}\)

• For an abstract interpretation,
  1. \(\text{lfp}(f) \subseteq \gamma(\text{lfp}(\bar{f}))\)
  2. \(\text{gfp}(f) \subseteq \gamma(\text{gfp}(\bar{f}))\)
Program Analysis

• A program analysis is correct iff everything which can happen during the runtime is predicted by the program analysis

→ Each state which can occur during the execution must be contained by the abstraction
Program Analysis

• The (denotation) program semantics defines a given input the behavior of the program

• The program analysis is interested in the behavior of the program for all possible inputs

To be able to talk also about the history of the program execution we introduce the Collecting Semantics
Collecting Semantics

- collect the set of traces that can reach a given program point l

\[
\{ (x,?) : (y,?) : (z,?) \}
\]

\[
[y := x]^1
\]

\[
[z := 1]^2
\]

\[
[y > 1]^3 \quad [y := 0]^6
\]

\[
[z := z*y]^4
\]

\[
[y := y-1]^5
\]
Collecting Semantics

- collect the set of traces that can reach a given program point $l$

```
\begin{array}{l}
\{ (x,?) : (y,?) : (z,?) \} \\
[y := x]^1 \\
\{ (x,?) : (y,?) : (z,?) : (y,1) \} \\
[z := 1]^2 \\
[y > 1]^3 \\
[z := z*y]^4 \\
[y := y-1]^5 \\
[y := 0]^6
\end{array}
```
Collecting Semantics

- collect the set of traces that can reach a given program point

```
[y := x]¹
{ (x,?) : (y,?) : (z,?) }

[z := 1]²
{ (x,?) : (y,?) : (z,?) : (y,1) }

[y > 1]³

[z := z*y]⁴

[y := y-1]⁵

[y := 0]⁶
{ (x,?) : (y,?) : (z,?) : (y,1) : (z,2) }
```
Collecting Semantics

- collect the set of traces that can reach a given program point \( l \)
Collecting Semantics

- collect the set of traces that can reach a given program point $l$

\[
\begin{align*}
[y := x]^1 & \rightarrow \{ (x,?) : (y,?) : (z,?) \} \\
[z := 1]^2 & \rightarrow \{ (x,?) : (y,?) : (z,?) : (y,1) \} \\
[y > 1]^3 & \rightarrow \{ (x,?) : (y,?) : (z,?) : (y,1) : (z,2) \} \\
[z := z*y]^4 & \rightarrow \{ (x,?) : (y,?) : (z,?) : (z,4) : (y,5) \} \\
y := y-1]^5 & \rightarrow \\
[y := 0]^6 & \rightarrow \\
\end{align*}
\]
Collecting Semantics

• collect the set of traces that can reach a given program point \( l \)

\[
\begin{align*}
\{ (x,?) : (y,?) : (z,?) : \\
(y,1) : (z,2),
\} \\
\{ (x,?) : (y,?) : (z,?) : \\
(y,1) : (z,2) : \\
(z,4) : (y,5),
\} \\
\{ (x,?) : (y,?) : (z,?) : \\
(y,1) : (z,2) : \\
(z,4) : (y,5) : \\
(z,4) : (y,5),
\}
\]

\[
\{ (x,?) : (y,?) : (z,?) : \\
(y,1) : (z,2) : \\
(z,4) : (y,5),
\}
\]

\[
\{ (x,?) : (y,?) : (z,?) : \\
(y,1) : (z,2) : \\
(z,4) : (y,5) : \\
(z,4) : (y,5),
\}
\]

\[
\{ (x,?) : (y,?) : (z,?) : \\
(y,1) : (z,2) : \\
(z,4) : (y,5) : \\
(z,4) : (y,5),
\}
\]

...
Reaching Definitions

trace:  
(x,?) : (y,?) : (z,?) : (y,1) : (z,2)

SRD(trace):  
{(x,?) , (y,1) , (z,2)}

\[ X \in \mathcal{P}(\text{Trace}_*): \{(x,?) : (y,?) : (z,?) : (y,1) : (z,2), (x,?) : (y,?) : (z,?) : (y,1) : (z,2) : (z,4) : (y,5), ... \} \]

SRD(X):  
{(x,?) , (y,1) , (z,2) , (z,4) , (y,5) }

Take SRD as \( \alpha \)
Collecting Semantics

• As before:
  • Extract a set of equations defining the possible set of traces
  • Compute the least fixed point of the set of equations

• And furthermore:
  • Prove the correctness: the set of traces computed this way is a superset of the possible traces
The equation system - 1

\[ CS_{entry}(l) \]

\[ CS_{exit}(l) = \{ \text{trace : (...,} l \text{)} | \text{trace} \in CS_{entry}(l) \} \]

\[ y := x \]

\[ z := 1 \]

\textbf{while} \ [y > 1] \textbf{do} ( \ [z := z \cdot y] \textbf{; } \ [y := y - 1] ) ;

\[ y := 0 \]

\[ CS_{exit}(1) = \{ \text{trace : (y,} 1 \text{)} | \text{trace} \in CS_{entry}(1) \} \]

\[ CS_{exit}(2) = \{ \text{trace : (z,} 2 \text{)} | \text{trace} \in CS_{entry}(2) \} \]

\[ CS_{exit}(3) = CS_{entry}(3) \]

\[ CS_{exit}(4) = \{ \text{trace : (z,} 4 \text{)} | \text{trace} \in CS_{entry}(4) \} \]

\[ CS_{exit}(5) = \{ \text{trace : (y,} 5 \text{)} | \text{trace} \in CS_{entry}(5) \} \]

\[ CS_{exit}(6) = \{ \text{trace : (y,} 6 \text{)} | \text{trace} \in CS_{entry}(6) \} \]
The equation system - 2

- initially all variables are uninitialized
- join information from where control could come from

\[
\begin{align*}
[y := x]^{1}\; & \quad \text{CS}_{\text{entry}}(1) = \{ (x,?) : (y,?) : (z,?) \} \\
[z := 1]^{2}\; & \quad \text{CS}_{\text{entry}}(2) = \text{CS}_{\text{exit}}(1) \\
\text{while } [y>1]^{3}\; & \quad \text{CS}_{\text{entry}}(3) = \text{CS}_{\text{exit}}(2) \cup \text{CS}_{\text{exit}}(5) \\
\text{do } ( \quad [z := z*y]^{4}\; & \quad \text{CS}_{\text{entry}}(4) = \text{CS}_{\text{exit}}(3) \\
[y := y-1]^{5}\; & \quad \text{CS}_{\text{entry}}(5) = \text{CS}_{\text{exit}}(4) \\
\text{)};
\end{align*}
\]

\[\begin{align*}
[y := 0]^{6}\; & \quad \text{CS}_{\text{entry}}(6) = \text{CS}_{\text{exit}}(3)
\end{align*}\]
Summary

• 12 sets:
  • $\text{CS}_{\text{entry}}(1), \ldots, \text{CS}_{\text{exit}}(6)$

• 12 equations:
  • $\text{CS}_j = G_j (\text{CS}_{\text{entry}}(1), \ldots, \text{CS}_{\text{exit}}(6))$

• One function:
  • $G: (\wp(\text{Trace}_*))^{12} \rightarrow (\wp(\text{Trace}_*))^{12}$

• The least fixed point of $G$ exists
  • But it is uncomputable!
Galois connection: inducing

\[ G_{\text{exit}}(4) \Rightarrow \mathcal{P}(\text{Var}_* \times \text{Lab}_*) \]

\[ \mathcal{P}(\text{Trace}_*) \Rightarrow \mathcal{P}(\text{Var}_* \times \text{Lab}_*) \]

\[ F_{\text{exit}}(4) = \alpha \circ G_{\text{exit}}(4) \circ \gamma \]
Inducing an analysis

Calculate:

$$RD_{exit}(4) = F_{exit}(4) (\ldots, RD_{entry}(4), \ldots)$$
Inducing an analysis

Using:

\[ F_{\text{exit}}(4) = \alpha \circ G_{\text{exit}}(4) \circ \gamma \]

Calculate:

\[ R_{\text{exit}}(4) = F_{\text{exit}}(4) (..., R_{\text{entry}}(4), ...) \]
\[ = \alpha (G_{\text{exit}}(4) (..., \gamma (R_{\text{entry}}(4)), ...) ) \]
Inducing an analysis

Using:

\[ G_{exit}(4)(\ldots, CS_{entry}(4), \ldots) = \{ \text{trace} : (z,4) \mid \text{trace} \in CS_{entry}(4) \} \]

Calculate:

\[ RD_{exit}(4) = F_{exit}(4) (\ldots, RD_{entry}(4), \ldots) \]
\[ = \alpha (G_{exit}(4) (\ldots, \gamma (RD_{entry}(4)), \ldots)) \]
\[ = \alpha (\{ \text{trace} : (z,4) \mid \text{trace} \in \gamma (RD_{entry}(4)) \}) \]
Inducing an analysis

Using:
\[ \alpha(X) = \text{SRD}(X) \]

Calculate:

\[ \text{RD}_{\text{exit}}(4) = F_{\text{exit}}(4) (\ldots, \text{RD}_{\text{entry}}(4), \ldots) \]
\[ = \alpha \left( G_{\text{exit}}(4) (\ldots, \gamma (\text{RD}_{\text{entry}}(4)), \ldots) \right) \]
\[ = \alpha \left( \{ \text{trace} : (z,4) \mid \text{trace} \in \gamma (\text{RD}_{\text{entry}}(4)) \} \right) \]
\[ = \text{SRD} \left( \{ \text{trace} : (z,4) \mid \text{trace} \in \gamma (\text{RD}_{\text{entry}}(4)) \} \right) \]
Inducing an analysis

Calculate:

\[ RD_{exit}(4) = F_{exit}(4) (..., RD_{entry}(4), ...) \]
\[ = \alpha \left( G_{exit}(4) (..., \gamma (RD_{entry}(4)), ...) \right) \]
\[ = \alpha (\{ \text{trace} : (z,4) | \text{trace} \in \gamma (RD_{entry}(4)) \}) \]
\[ = \text{SRD} (\{ \text{trace} : (z,4) | \text{trace} \in \gamma (RD_{entry}(4)) \}) \]
\[ = \text{SRD} (\{ \text{trace} | \text{trace} \in \gamma (RD_{entry}(4)) \} \setminus \{ (z,l) | l \in \text{Lab} \} \cup \{ (z,4) \}) \]
Inducing an analysis

Using:

\[ \alpha(X) = \text{SRD}(X) \]

Calculate:

\[
\begin{align*}
\text{RD}_{\text{exit}}(4) & = \text{F}_{\text{exit}}(4) (\ldots, \text{RD}_{\text{entry}}(4), \ldots) \\
& = \alpha (G_{\text{exit}}(4) (\ldots, \gamma (\text{RD}_{\text{entry}}(4)), \ldots)) \\
& = \alpha (\{ \text{trace} : (z,4) \mid \text{trace} \in \gamma (\text{RD}_{\text{entry}}(4)) \}) \\
& = \text{SRD} (\{ \text{trace} : (z,4) \mid \text{trace} \in \gamma (\text{RD}_{\text{entry}}(4)) \}) \\
& = \text{SRD} (\{ \text{trace} \mid \text{trace} \in \gamma (\text{RD}_{\text{entry}}(4)) \}) \setminus \{ (z,l) \mid l \in \text{Lab} \} \cup \{ (z,4) \} \\
& = \alpha (\gamma (\text{RD}_{\text{entry}}(4))) \setminus \{ (z,l) \mid l \in \text{Lab} \} \cup \{ (z,4) \}
\end{align*}
\]
Inducing an analysis

Using:

\[ \alpha (\gamma (Y)) = Y \]

Calculate:

\[
\begin{align*}
RD_{exit}(4) &= F_{exit}(4) (...) \cup RD_{entry}(4), ...
\end{align*}
\]

\[
= \alpha (G_{exit}(4) (...) \cup \gamma (RD_{entry}(4)), ...
\]

\[
= \alpha (\{ trace : (z,4) | trace \in \gamma (RD_{entry}(4))\})
\]

\[
= SRD (\{ trace : (z,4) | trace \in \gamma (RD_{entry}(4))\})
\]

\[
= SRD (\{ trace | trace \in \gamma (RD_{entry}(4)) \} \setminus \{ (z,l) | l \in Lab \} \cup \{ (z,4) \})
\]

\[
= \alpha (\gamma (RD_{entry}(4))) \setminus \{ (z,l) | l \in Lab \} \cup \{ (z,4) \}
\]

\[
= RD_{entry}(4) \setminus \{ (z,l) | l \in Lab \} \cup \{ (z,4) \}
\]

just as before!
Systematic Design of G.C.

- From one or several Galois Connection new Galois connection can be designed
- There are several methods
- Advantage: for each of these methods the correctness of the newly developed analysis follows by design (i.e. the new Galois Connection forms indeed a Galois Connection)
- Stating with the actual semantics analyses can be developed
Example: Interval Analysis

- Tries to approximate the runtime value of a program variable by an interval, e.g. \( i \in [0, 100] \)
- Useful to eliminate checks of bounds
- Abstract values: environments mapping variables to intervals
Example: Interval Analysis

abstract domain:

- Bound: \( B = \mathbb{Z} \cup \{-\infty, +\infty\} \)
- Interval: \( I = B \times B \)
- Environment: \( \text{Env} = \text{Var} \rightarrow I \)

\( D = \text{Env} \cup \{\perp\} \)

\[(l_1,u_1) \subseteq (l_2,u_2) \iff l_1 \geq l_2 \land u_1 \leq u_2\]
Example: Interval Analysis

• Transfer function:
  • For assignments: change the variable in the environment
  • For branches: try to refine the intervals for the true and the false directions

• Combine function:
  • Pointwise union the intervals for every variable

• **Attention:** infinite ascending chains
  → Termination not guaranteed
Technique: Widening

• **Idea:** it is always safe to take a fixed point above the smallest one

• **Method:** define a (widening) operator which takes the old and the new values of an iteration sequence such that the result is always greater than the new value

• **Result:** the iteration sequence gets shorter and the result less precise
Technique: Narrowing

- **Idea:** improve the quality of the results obtained by a widening
- **Method:** define a (narrowing) operator which takes the old and the new values of an iteration sequence such that one is guaranteed not to jump over the smallest fixed point
- **Result:** the results get more precise but are still correct
Widening

computation of least fixed points

a technique for speeding up the computation of fixed points

\{ x | x \supseteq \text{lfp}(f') \}
Narrowing

\[ \{ x \mid x \supseteq \text{lfp}(f) \} \]

improving fixed point approximations

\[ \text{lfp}^\uparrow(f) \]
Widening

A widening $\forall$ is an Operator $\forall: D \times D \rightarrow D$ such that
1. $x \subseteq (x \forall y)$ and $y \subseteq (x \forall y)$
2. for all ascending chains $x_0 \subseteq x_1 \subseteq \ldots$ the chain
   
   $y_0 = x_0$
   
   $y_{i+1} = y_i \forall x_{i+1}$

   is finite.
Narrowing

A narrowing $\Delta$ is an Operator

$\Delta : D \times D \rightarrow D$ such that

1. $l \subseteq x$ and $l \subseteq y \Rightarrow l \subseteq (x \Delta y) \subseteq x$

2. forall ascending chains $x_0 \supseteq x_1 \supseteq \ldots$ the chain

   $y_0 = x_0$

   $y_{i+1} = y_i \Delta x_{i+1}$

is finite.
Program test;
var i, s, r1, r2 : integer;
a : array [0..131] of integer;
begin
  i := 0;
  s := 1;
  while i <= 100 do begin
    r1 := s%2;
    r2 := s%4 - r1;
    if r1*2 = r2 then
      s := s/2;
    else
      s := s/2 + 2^4;
    a[s+i] := i;
    i := i + 1;
  end;
end.
Example: Interval Analysis

• **Widening:**
  • If a bound increased in the last iteration set it to $\pm \infty$

• **Narrowing:**
  • If the old value of a bound is $\pm \infty$ and
  • The new one is better take the new one
Start

i := 0

s := 1

i <= 100

r1 := s%2

r2 := s%4 - r1

r1*2 = r2

s := s/2

a[s+i] := i

Exit

i <= 100

true

true

s := s/2
\[
\begin{align*}
    \text{Start} & \quad \Rightarrow \quad i := 0 \\
    \text{s := 1} & \quad \Rightarrow \quad i \in [0,0] \quad s \in [1,1] \\
    \text{i <= 100} & \quad \Rightarrow \quad \text{true} \\
    \text{r1 := s%2} & \quad \Rightarrow \quad r1 \in [0,1] \\
    \text{r2 := s%4 - r1} & \quad \Rightarrow \quad r2 \in [-1,3] \\
    \text{r1*2 = r2} & \quad \Rightarrow \quad \text{true} \\
    \text{a[s+i] := i} & \quad \Rightarrow \quad i \in [0,100] \\ & \quad \quad s \in [0,31] \\
    \text{s := s/2} & \quad \Rightarrow \quad s \in [16,31] \\
    \text{Exit} & \quad \Rightarrow \quad \text{i := 0} \\
\end{align*}
\]
Correctness of Program Analysis

A program analysis is correct if

1. The abstract domain $D_{\text{abs}}$ is a complete lattice

2. $(D_{\text{conc}}, \alpha, \gamma, D_{\text{abs}})$ form a Galois Connection

3. The transfer functions are locally consistent to the collection semantics

4. If narrowing and widening are used the must be operators in the sense of the definitions

5. The abstract initial value ($\text{i}$) is an abstraction of all possible concrete initial values ($\gamma (\text{i}) \sqsubseteq \text{Init}$)