Static Program Analysis
Part 11 – abstract interpretation

http://cs.au.dk/~amoeller/spa/

Anders Møller
Computer Science, Aarhus University
Abstract interpretation provides a solid mathematical foundation for reasoning about static program analyses

• Is my analysis sound? (Does it safely approximate the actual program behavior?)
• Is it as precise as possible for the currently used analysis lattice? If not, where can precision losses arise? Which precision losses can be avoided (without sacrificing soundness)?

Answering such questions requires a precise definition of the semantics of the programming language, and precise definitions of the analysis abstractions in terms of the semantics
Agenda

- Collecting semantics
- Abstraction and concretization
- Soundness
- Optimality
- Completeness
- Trace semantics
Sign analysis, recap

\[ \text{Sign} = + \quad - \quad 0 \]

\[ \text{States} = \text{Vars} \rightarrow \text{Sign} \]

\[ \text{States}^n \]

\[ \begin{align*}
[v_1] &= af_{v_1}([v_1], \ldots, [v_n]) \\
[v_2] &= af_{v_2}([v_1], \ldots, [v_n]) \\
&\vdots \\
[v_n] &= af_{v_n}([v_1], \ldots, [v_n])
\end{align*} \]

\[ af([v_1], \ldots, [v_n]) = (af_{v_1}([v_1], \ldots, [v_n]), \ldots, af_{v_n}([v_1], \ldots, [v_n])) \]

\[ af: \text{States}^n \rightarrow \text{States}^n \]

is the analysis representation of the given program

\[ [P] = \text{lfp}(af) \]
Program semantics as constraint systems

Concret States = Vars → ℤ

{v} ⊆ Concret States

This is called a reachable states collecting semantics
The semantics of expressions

\[ ceval : \text{ConcreteStates} \times \text{Exp} \rightarrow \mathcal{P}(\mathbb{Z}) \]

\[ ceval(\rho, X) = \{ \rho(X) \} \]
\[ ceval(\rho, I) = \{ I \} \]
\[ ceval(\rho, \text{input}) = \mathbb{Z} \]
\[ ceval(\rho, E_1 \text{ op } E_2) = \{ v_1 \text{ op } v_2 \mid v_1 \in ceval(\rho, E_1) \land v_2 \in ceval(\rho, E_2) \} \]

\[ ceval(R, E) = \bigcup_{\rho \in R} ceval(\rho, E) \]
Successors and joins

\[ \text{csucc: } \text{ConcreteStates} \times \text{Nodes} \rightarrow \mathcal{P}\text{(Nodes)} \]

\[ \text{csucc}(R, v) = \bigcup_{\rho \in R} \text{csucc}(\rho, v) \]

\[ \text{CJOIN}(v) = \{ \rho \in \text{ConcreteStates} \mid \exists w \in \text{Nodes}: \rho \in \{w\} \land v \in \text{csucc}(\rho, w) \} \]
Semantics of statements

$$\{X=E\} = \{ \rho[X \mapsto z] \mid \rho \in CJOIN(v) \land z \in ceval(\rho, E) \}$$

$$\{\text{var } X_1, \ldots, X_n\} =$$

$$\{ \rho[X_1 \mapsto z_1, \ldots, X_n \mapsto z_n] \mid \rho \in CJOIN(v) \land z_1 \in \mathbb{Z} \land \cdots \land z_n \in \mathbb{Z} \}$$

$$\{\text{entry}\} = \{[]\}$$

$$\{v\} = CJOIN(v)$$
The resulting constraint system

\[
\begin{align*}
\{v_1\} &= cf_{v_1}(\{v_1\}, \ldots, \{v_n\}) \\
\{v_2\} &= cf_{v_2}(\{v_1\}, \ldots, \{v_n\}) \\
& \vdots \\
\{v_n\} &= cf_{v_n}(\{v_1\}, \ldots, \{v_n\}) \\
\end{align*}
\]

\[
cf : (\mathcal{P}(\text{ConcreteStates}))^n \rightarrow (\mathcal{P}(\text{ConcreteStates}))^n
\]

\[
cf(x_1, \ldots, x_n) = (cf_{v_1}(x_1, \ldots, x_n), \ldots, cf_{v_n}(x_1, \ldots, x_n))
\]

is the semantic representation of the given program

\[
x = cf(x)
\]

\[
\{P\} = \text{lfp}(cf)
\]
Example

```
var x;
  x = 0;
  while (input) {
    x = x + 2;
  }
```

<table>
<thead>
<tr>
<th></th>
<th>solution 1</th>
<th>solution 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{entry}</code></td>
<td>`{[]}</td>
<td>`{[]}</td>
</tr>
<tr>
<td><code>{var x}</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>{x = 0}</code></td>
<td>`{[x → z]</td>
<td>z ∈ ℤ}`</td>
</tr>
<tr>
<td><code>{input}</code></td>
<td>`{[x → z]</td>
<td>z ∈ {0, 2, 4, ...}}`</td>
</tr>
<tr>
<td><code>{x = x + 2}</code></td>
<td>`{[x → z]</td>
<td>z ∈ {2, 4, ...}}`</td>
</tr>
<tr>
<td><code>{exit}</code></td>
<td>`{[x → z]</td>
<td>z ∈ {0, 2, 4, ...}}`</td>
</tr>
</tbody>
</table>

the least solution
Kleene’s fixed point theorem for complete join morphisms

If $f : L \rightarrow L$ is a complete join morphism if $f(\bigsqcup A) = \bigsqcup_{a \in A} f(a)$ for every $A \subseteq L$.

If $f$ is a complete join morphism:

$$lfp(f) = \bigsqcup_{i \geq 0} f^i(\bot)$$

(even when $L$ has infinite height!)

(Proof: Exercise 11.8)

$$(\mathcal{P}(ConcreteStates))^n$$ is a complete lattice (a product of a powerset lattice)

$cf$ is a complete join morphism (Exercise 11.9)
Tarski’s fixed-point theorem

In a complete lattice $L$, every monotone function $f : L \rightarrow L$ has a unique least fixed point given by $\bigcap \{ x \in L \mid f(x) \subseteq x \}$.

(Proof in Chapter 6)

$cf$ is monotone  (Exercise 11.3)
Semantics vs. analysis

```
var a, b, c;
a = 42;
b = 87;
if (input) {
c = a + b;
} else {
c = a - b;
}
```

\[ [b = 87] = \{ [a \mapsto 42, b \mapsto 87, c \mapsto z] \mid z \in \mathbb{Z} \} \]
\[ [c = a - b] = \{ [a \mapsto 42, b \mapsto 87, c \mapsto -45] \} \]
\[ [exit] = \{ [a \mapsto 42, b \mapsto 87, c \mapsto 129], [a \mapsto 42, b \mapsto 87, c \mapsto -45] \} \]

\[ [b = 87] = [a \mapsto \top, b \mapsto \top, c \mapsto \top] \]
\[ [c = a - b] = [a \mapsto \top, b \mapsto \top, c \mapsto \top] \]
\[ [exit] = [a \mapsto \top, b \mapsto \top, c \mapsto \top] \]
Agenda

- Collecting semantics
- Abstraction and concretization
- Soundness
- Optimality
- Completeness
- Trace semantics
Abstraction functions for sign analysis

$$\alpha_a : \mathcal{P}(\mathbb{Z}) \rightarrow \text{Sign}$$
$$\alpha_b : \mathcal{P}(\text{ConcreteStates}) \rightarrow \text{States}$$
$$\alpha_c : (\mathcal{P}(\text{ConcreteStates}))^n \rightarrow \text{States}^n$$

$$\alpha_a(D) = \begin{cases} 
\bot & \text{if } D \text{ is empty} \\
+ & \text{if } D \text{ is nonempty and contains only positive integers} \\
- & \text{if } D \text{ is nonempty and contains only negative integers} \\
0 & \text{if } D \text{ is nonempty and contains only the integer } 0 \\
\top & \text{otherwise} 
\end{cases}$$

for any $D \in \mathcal{P}(\mathbb{Z})$

$$\alpha_b(R) = \sigma \text{ where } \sigma(X) = \alpha_a(\{\rho(X) \mid \rho \in R\})$$

for any $R \subseteq \text{ConcreteStates}$ and $X \in \text{Vars}$

$$\alpha_c(R_1, \ldots, R_n) = (\alpha_b(R_1), \ldots, \alpha_b(R_n))$$

for any $R_1, \ldots, R_n \subseteq \text{ConcreteStates}$
Concretization functions for sign analysis

\[ \gamma_a : \text{Sign} \rightarrow \mathcal{P}(\mathbb{Z}) \]
\[ \gamma_b : \text{States} \rightarrow \mathcal{P}(\text{ConcreteStates}) \]
\[ \gamma_c : \text{States}^n \rightarrow (\mathcal{P}(\text{ConcreteStates}))^n \]

\[ \gamma_a(s) = \begin{cases} 
  \emptyset & \text{if } s = \bot \\
  \{1, 2, 3, \ldots\} & \text{if } s = + \\
  \{-1, -2, -3, \ldots\} & \text{if } s = - \\
  \{0\} & \text{if } s = \emptyset \\
  \mathbb{Z} & \text{if } s = \top 
\end{cases} \]

for any \( s \in \text{Sign} \)

\[ \gamma_b(\sigma) = \{ \rho \in \text{ConcreteStates} \mid \rho(X) \in \gamma_a(\sigma(X)) \text{ for all } X \in \text{Vars} \} \]

for any \( \sigma \in \text{States} \)

\[ \gamma_c(\sigma_1, \ldots, \sigma_n) = (\gamma_b(\sigma_1), \ldots, \gamma_b(\sigma_n)) \]

for any \( (\sigma_1, \ldots, \sigma_n) \in \text{States}^n \)
Monotonicity of abstraction and concretization functions

Concretatization functions are, like abstraction functions, naturally monotone.

(A larger set of concrete values should correspond to a larger abstract state, and conversely)
Galois connections

The pair of monotone functions, $\alpha$ and $\gamma$, is called a Galois connection if

$$
\gamma \circ \alpha \text{ is extensive}
$$

$$
\alpha \circ \gamma \text{ is reductive}
$$

all three pairs of abstraction and concretization functions $(\alpha_a, \gamma_a)$, $(\alpha_b, \gamma_b)$, and $(\alpha_c, \gamma_c)$ from the sign analysis example are Galois connections

(Exercise 11.13)
Galoiis connections

For Galois connections, the concretization function uniquely determines the abstraction function and vice versa:

\[
\gamma(y) = \bigsqcup \{ x \in L_1 \mid \alpha(x) \subseteq y \}
\]

\[
\alpha(x) = \bigsqcap \{ y \in L_2 \mid x \subseteq \gamma(y) \}
\]

(Proof: Exercise 11.20)
Galois connections

For this lattice, given the “obvious” concretization function, is there an abstraction function such that the concretization function and the abstraction function form a Galois connection?

how should we define $\alpha_a(\{0\})$?  
(Exercise 11.22)
Representation functions

\[ \beta : \mathbb{Z} \rightarrow \text{Sign} \]

\[ \beta(d) = \begin{cases} 
+ & \text{if } d > 0 \\
- & \text{if } d < 0 \\
0 & \text{if } d = 0 
\end{cases} \]

\[ \alpha_a(D) = \bigcup \{ \beta(d) \mid d \in D \} \]
Agenda

• Collecting semantics
• Abstraction and concretization
• **Soundness**
• Optimality
• Completeness
• Trace semantics
Soundness

\[ \alpha(\{P\}) \subseteq [P] \]
Soundness

\[ \{P\} \subseteq \gamma([P]) \]

\[
\begin{array}{c}
\{P\} \\
\gamma_c \\
(P(ConcreteStates))^n \\
\text{States}^n
\end{array}
\]

(Exercise 11.17)
Sound abstractions

$$\alpha_a(\text{ceval}(R, E)) \subseteq \text{eval}(\alpha_b(R), E)$$

$$c\text{succ}(R, v) \subseteq \text{succ}(v) \text{ for any } R \subseteq \text{ConcreteStates}$$

$$\alpha_b(\text{CJOIN}(v)) \subseteq \text{JOIN}(v)$$

if $$\alpha_b([\{w\}]) \subseteq [w] \text{ for all } w \in \text{Nodes}.$$

(Exercise 11.31 and Exercise 11.32)
Sound abstractions

if \( v \) represents an assignment statement \( X = E \):

\[
\begin{align*}
\text{cf}_v([v_1], \ldots, [v_n]) &= \{ \rho[X \mapsto z] \mid \rho \in CJOIN(v) \land z \in ceval(\rho, E) \} \\
\text{af}_v([v_1], \ldots, [v_n]) &= \sigma[X \mapsto eval(\sigma, E)] \text{ where } \sigma = JOIN(v)
\end{align*}
\]

\[
\alpha_b(cf_v(R_1, \ldots, R_n)) \subseteq af_v(\alpha_b(R_1), \ldots, \alpha_b(R_n))
\]

(Exercise 11.33)
The two constraint systems

\[ cf([v_1], \ldots, [v_n]) = (cf_{v_1}([v_1], \ldots, [v_n]), \ldots, cf_{v_n}([v_1], \ldots, [v_n])) \]

\[ af([v_1], \ldots, [v_n]) = (af_{v_1}([v_1], \ldots, [v_n]), \ldots, af_{v_n}([v_1], \ldots, [v_n])) \]
Sound abstractions

\[ \alpha_c(cf(R_1, \ldots, R_n)) \subseteq af(\alpha_c(R_1, \ldots, R_n)) \]
Sound abstractions

\[ \alpha \circ cf \subseteq af \circ \alpha \]

\[ cf \circ \gamma \subseteq \gamma \circ af \]

Equivalent, if \( \alpha \) and \( \gamma \) form a Galois connection

(Proof: Exercise 11.34)
The soundness theorem

If $L_1$ and $L_2$ are complete lattices with a concretization function $\gamma : L_2 \rightarrow L_1$, $cf : L_1 \rightarrow L_1$ and $af : L_2 \rightarrow L_2$ are monotone, and $af$ is a sound abstraction of $cf$ with respect to $\gamma$, i.e., $cf \circ \gamma \sqsubseteq \gamma \circ af$, then $lfp(cf) \sqsubseteq \gamma(lfp(af))$. 
Agenda

- Collecting semantics
- Abstraction and concretization
- Soundness
- Optimality
- Completeness
- Trace semantics
Optimal abstractions

$af$ is an optimal abstraction of $cf$ if

$$af = \alpha \circ cf \circ \gamma$$
Optimal abstractions in sign analysis?

\[ \hat{\star} \text{ is optimal:} \]
\[ s_1 \hat{\star} s_2 = \alpha_a(\gamma_a(s_1) \cdot \gamma_a(s_2)) \]

\[ \text{eval is not optimal:} \]
\[ \sigma(x) = \top \]
\[ \text{eval}(\sigma, x-x) = \top \]
\[ \alpha_b(ceval(\gamma_b(\sigma), x-x)) = \emptyset \]

Even if we could make \textit{eval} optimal, the analysis result is not always optimal:

```
x = input;
y = x;
z = x - y;
```
Agenda

- Collecting semantics
- Abstraction and concretization
- Soundness
- Optimality
- Completeness
- Trace semantics
Completeness

\[ \alpha([P]) \supseteq [P] \]

(compare with slide 23)

Sound and complete:

\[ \alpha([P]) = [P] \]

(Intuitively, the analysis result is the most precise possible for the currently used lattice)

Not the same as \[ \{P\} = \gamma([P]) \] (called “exact”)

(Intuitively, the analysis result exactly captures the semantics of the program)
Completeness in sign analysis?

\[ \hat{\star} \text{ is complete:} \]

\[ \alpha_a(D_1) \hat{\star} \alpha_a(D_2) \subseteq \alpha_a(D_1 \cdot D_2) \]

\[ \hat{\oplus} \text{ is not complete} \]

\[ \alpha_a(D_1) \hat{\oplus} \alpha_a(D_2) \not\subseteq \alpha_a(D_1 + D_2) \]

Sign analysis is sound and complete for *some* programs, but not for all programs.
Agenda

- Collecting semantics
- Abstraction and concretization
- Soundness
- Optimality
- Completeness
- Trace semantics
Limitations of the reachable states collecting semantics

The reachable states collecting semantics “collects” a set of concrete states for each program point:

\[ \text{ConcreteStates} = \text{Vars} \rightarrow \mathbb{Z} \]

\[ \{v\} \subseteq \text{ConcreteStates} \]

That is not always sufficient – a trivial example:

```plaintext
main(x) {
    return x;
}
```

For this program, the collecting semantics doesn’t allow us to express properties such as “in any execution, the return value is the same as the input value”
Trace semantics

\[\text{Traces} = (\text{Nodes} \times \text{ConcreteStates})^*\]

\[ct_v : \text{ConcreteStates} \rightarrow \mathcal{P}(\text{ConcreteStates})\]

\[ct_{X=E}(\rho) = \{\rho[X \mapsto z] | z \in \text{ceval}(\rho, E)\}\]

\[ct_v(\rho) = \{\rho\} \quad \text{for other kinds of nodes}\]

\[\{P\} \in \mathcal{P}(\text{Traces})\]

\[(\text{entry}, []) \in \{P\}\]

\[\pi \cdot (v, \rho) \in \{P\} \land v' \in \text{csucc}(\rho, v) \land \rho' \in ct_{v'}(\rho) \implies \pi \cdot (v, \rho) \cdot (v', \rho') \in \{P\}\]
Example

```c
main(x) {
    return x;
}

\langle P \rangle = \{(entry, [x \mapsto 0]) \cdot (return \ x, [x \mapsto 0]) \cdot (exit, [x \mapsto 0]),
        (entry, [x \mapsto 1]) \cdot (return \ x, [x \mapsto 1]) \cdot (exit, [x \mapsto 1]),
        \ldots\}
```
Reachable states is an abstraction of traces

the relation between the reachable states collecting semantics and the trace semantics can be expressed as a Galois connection induced by an abstraction function

$$\alpha_t : \mathcal{P}(\text{Traces}) \rightarrow (\mathcal{P}(\text{ConcreteStates}))^n$$

defined by

$$\alpha_t(T) = (R_1, \ldots, R_n)$$

where $R_i = \{\rho \mid \cdots (v_i, \rho) \cdots \in T\}$ for each $i = 1, \ldots, n$. 
Composition of Galois connections

**Exercise 11.57**: Let $\alpha_1 : L_1 \rightarrow L_2$, $\gamma_1 : L_2 \rightarrow L_1$, $\alpha_2 : L_2 \rightarrow L_3$, and $\gamma_2 : L_3 \rightarrow L_2$. Assume both $(\alpha_1, \gamma_1)$ and $(\alpha_2, \gamma_2)$ are Galois connections. Prove that $(\alpha_2 \circ \alpha_1, \gamma_1 \circ \gamma_2)$ is then also a Galois connection.

**Exercise 11.58**: Prove that the reachable states collecting semantics is sound with respect to the trace semantics. (Even though the collecting semantics is not a computable analysis, we can still apply the notion of soundness and the proof techniques from Section 11.3.)
Soundness of reaching definitions analysis

Instrumented trace semantics:

\[ \text{Traces}_{RD} = (\text{Nodes} \times \text{ConcreteStates} \times \text{ReachingDefs})^* \]

\[ \text{ReachingDefs} = Vars \leftrightarrow \text{Nodes} \]

\[(entry, [], []) \in \{P\}_{RD}\]

\[\pi \cdot (v, \rho, \delta) \in \{P\}_{RD} \land v' \in \text{csucc}(\rho, v) \land \rho' \in \text{ct}_{v'}(\rho) \implies \]

\[\pi \cdot (v, \rho, \delta) \cdot (v', \rho', \delta') \in \{P\}_{RD}\]

\[
\delta' = \begin{cases} 
\delta[X \mapsto v'] & \text{if } v' \text{ is an assignment } X = E \\
\delta & \text{otherwise}
\end{cases}
\]
Soundness of reaching definitions analysis

Abstraction:

\[ \alpha_{RD} : \mathcal{P}(\text{Traces}_{RD}) \rightarrow (\mathcal{P}(\text{Nodes}))^n \]

\[ \alpha_{RD}(T) = (R_1, \ldots, R_n) \]

\[ R_i = \{ w \mid \cdots (v_i, \rho, \delta) \cdots \in T \text{ where } \delta(X) = w \text{ for some } \rho, \delta, X \} \]

Soundness:

\[ \alpha_{RD}(\{P\}_{RD}) \sqsubseteq [P]_{RD} \]
Soundness proofs for other analyses

**Exercise 11.61:** Use the approach from Section 11.3 and an appropriate instrumented trace semantics to prove that the available expressions analysis from Section 5.5 is sound. (This is more tricky than Exercise 11.60, because available expressions analysis is a “must” analysis!)

**Exercise 11.62:** Use the approach from Section 11.3 and an appropriate instrumented trace semantics to prove that the live variables analysis from Section 5.4 is sound. As part of this, you need to specify an appropriate collecting semantics that formally captures what it means for a variable to be live (see the informal definition in Section 5.4). (This is more tricky than Exercise 11.60, because live variables analysis is a “backward” analysis!)
Conclusions

Abstract interpretation provides a solid mathematical foundation for reasoning about soundness and precision of static program analyses.

We need
- the static analysis (the analysis lattices and constraint rules)
- the language semantics (a suitable collecting semantics)
- abstraction/concretization functions that specify the meaning of the elements in the analysis lattice in terms of the semantic lattice

... and then
- if each constituent of the analysis is a sound abstraction of its semantic counterpart, then the analysis is sound (according to the soundness theorem)
- if an abstraction is optimal, then it is as precise as possible (yet sound), relative to the choice of analysis lattice
- if the analysis is sound and complete, then the analysis result is as precise as possible (yet sound), relative to the choice of analysis lattice