Recap: static vs. dynamic analysis

**Static analysis**
- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but conservative.

**Dynamic analysis**
- Reason about the program based on some program executions.
- Observe concrete behavior at run time.
- Improve confidence in correctness.
- Unsound* but precise.

* Some static analyses are unsound; dynamic analyses can be sound.

---

Recap: Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Analysis result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Pos</td>
</tr>
<tr>
<td>Neg</td>
<td>Neg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Analysis result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>TP</td>
</tr>
<tr>
<td>Neg</td>
<td>FN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Analysis result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>FP</td>
</tr>
<tr>
<td>Neg</td>
<td>TN</td>
</tr>
</tbody>
</table>
Recap: Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Analysis result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>TP</td>
</tr>
<tr>
<td>Pos</td>
<td>FN</td>
</tr>
<tr>
<td>Neg</td>
<td>FP</td>
</tr>
<tr>
<td>Neg</td>
<td>TN</td>
</tr>
</tbody>
</table>

**Precision:**
\[
\frac{|TP|}{|TP| + |FP|}
\]

**Recall:**
\[
\frac{|TP|}{|TP| + |FN|}
\]

2. Soundness vs. completeness

**Soundness:**
no FNs

**Completeness:**
no FPs

3. Abstract domain

Concrete domain: 0, 1, 2, 3, 4, ...

Abstract domain: even, odd, unknown
Recap: Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)
2. Soundness vs. completeness
3. Abstract domain
4. Accurate vs. precise analysis

Concrete domain

Abstract domain

- Precision
- even, odd, unknown

Accuracy

Today

- Abstract interpretation
  - Introduction
  - Abstraction functions
  - Transfer functions
  - Lattices

Abstract Interpretation: applications

Compiler checks and optimizations
- Liveness analysis (register reallocation)
- Reachability analysis (dead code elimination)
- Code motion (while(cond){x = comp(); ...})

Properties of an ideal program analysis

- Soundness
- Completeness
- Termination

```c
int x = 0;
while (!isDone()) {
  x = x + 1;
}
```

...
Properties of an ideal program analysis

- Soundness
- Completeness
- Termination

```
int x = 0;
while (!isDone()) {
  x = x + 1;
}
```

Abstract interpretation sacrifices completeness/precision

**Analogy: analytical vs. numerical solutions**
Abstract interpretation: example

Are all statements necessary?

Program
```
x = 0;
y = read_even();
x = y + 1;
y = 2 * x;
x = y - 2;
y = x / 2;
```

Concrete execution
```
{x=0; y=undef}
{x=0; y=8}
{x=9; y=8}
{x=9; y=18}
{x=16; y=18}
```

Abstract interpretation: example

SSA form
```
x1 = 0;
y1 = read_even();
x2 = y1 + 1;
y2 = 2 * x2;
x3 = y2 - 2;
y4 = x3 / 2;
```

Concrete execution
```
{x=0; y=8}
{x=9; y=8}
{x=9; y=18}
{x=16; y=18}
```
Abstract interpretation: example

Program:

```plaintext
x = 0;
y = read_even();
x = y + 1;
y = 2 * x;
x = y - 2;
y = x / 2;
```

Concrete execution:

<table>
<thead>
<tr>
<th>Concrete state</th>
<th>Mapping to abstract domain (even, odd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x=0; y=undef}</td>
<td>{x=e; y=e}</td>
</tr>
<tr>
<td>{x=0; y=8}</td>
<td>{x=o; y=e}</td>
</tr>
<tr>
<td>{x=9; y=8}</td>
<td>{x=o; y=e}</td>
</tr>
<tr>
<td>{x=9; y=18}</td>
<td>{x=e; y=e}</td>
</tr>
<tr>
<td>{x=16; y=18}</td>
<td>{x=e; y=e}</td>
</tr>
<tr>
<td>{x=16; y=8}</td>
<td>{x=e; y=e}</td>
</tr>
</tbody>
</table>

Abstract interpretation: safety

Concrete state
 Concrete exec
 Concrete state

Abstract state
 Abstract exec
 Abstract state

Transfer function
 Abstraction function

Abstract interpretation: safety

Concrete state
 Concrete exec
 Concrete state

Abstract state
 Abstract exec
 Abstract state
Abstract interpretation: safety example

Abstract domain: \{odd, even_{2}, even_{4}, ?\}

\[ \begin{align*}
x &= 16 \\
16 / 2 & \rightarrow x = 8 \\
8 / 2 & \rightarrow x = 4
\end{align*} \]

\[ x = \text{even}_{4} \]

Abstract interpretation: safety example

\[ \begin{align*}
x &= 16 \\
16 / 2 & \rightarrow x = 8 \\
8 / 2 & \rightarrow x = 4
\end{align*} \]

\[ x = \text{even}_{4} \]
Abstract interpretation: safety example

Abstract domain: \{odd, even, even, ?\}

\[
\begin{align*}
    x &= 16 \\
    \text{even}_4 / 2 &\rightarrow x = \text{even}_2 \\
    \text{even}_2 / 2 &\rightarrow x = ?
\end{align*}
\]

Abstract interpretation: safety example

Abstract domain: \{odd, even, even, ?\}

\[
\begin{align*}
    x &= 8 \\
    \text{even}_4 / 2 &\rightarrow x = \text{even}_2 \\
    \text{even}_2 / 2 &\rightarrow x = ?
\end{align*}
\]

Abstract interpretation: safety example

Concrete state \rightarrow \text{Concrete exec} \rightarrow \text{Concrete state}

Abstract state \rightarrow \text{Abstract exec} \rightarrow \text{Abstract state}
Abstract interpretation: lattices

Abstract domain: even, odd, unknown (τ), ∅ (⊥)

\[
\begin{array}{ccc}
\top & \Downarrow & \text{even} \\
\downarrow & & \downarrow \\
\text{odd} & & \bot
\end{array}
\]

Goal: approximate the values of x after the loop

Possible abstract domains:
- Powerset of set of integers
- Intervals
- ...