Cryptographic Constant-Time Verification of C programs by Abstract Interpretation

David Pichardie
Cache timing attacks

• Common side-channel: Cache timing attacks
• Exploit the latency between cache hits and misses
• Attackers can recover cryptographic keys
  • Tromer et al (2010), Gullasch et al (2011) show efficient attacks on AES implementations
• Based on the use of look-up tables
  • Access to memory addresses that depend on the key
Constant-time programs

Characterization

• Constant-time programs do not:
  • branch on secrets
  • perform memory accesses that depend on secrets

• There are constant-time implementations of many cryptographic algorithms: AES, DES, RSA, etc
Constant-time programs

Example
Constant-time programs
Example

```java
boolean testPIN(int code[]) {
    for (int i=0; i<N; i++) {
        if (code[i] != secret[i]) return false;
    }
    return true;
}
```
Constant-time programs

Example

```java
boolean testPIN(int code[]) {
    for (int i=0; i<N; i++) {
        if (code[i] != secret[i]) return false;
    }
    return true;
}
```

Not constant-time
Example

```java
boolean testPIN(int code[]) {
    for (int i=0; i<N; i++) {
        if (code[i] != secret[i]) return false;
    }
    return true;
}
```

```java
boolean testPIN(int code[]) {
    int diff = 0;
    for (int i=0; i<N; i++) {
        diff = diff | (code[i] ^ secret[i]);
    }
    return (diff == 0);
}
```

Not constant-time
Constant-time programs

Example

```java
boolean testPIN(int code[]) {
    for (int i=0; i<N; i++) {
        if (code[i] != secret[i]) return false;
    }
    return true;
}
```

Not constant-time

```java
boolean testPIN(int code[]) {
    int diff = 0;
    for (int i=0; i<N; i++) {
        diff = diff | (code[i] ^ secret[i]);
    }
    return (diff == 0);
}
```

Constant-time
This lecture

1. Presentation of recent works on the topic

2. Introductory course on abstract interpretation

3. Back to recent research works and conclusion
Verification of constant-time programs

Challenges

• Provide a mechanism to formally check that a program is constant-time
  • static tainting analysis for implementations of cryptographic algorithms

• At low level implementation (C, assembly), advanced static analysis is required
  • secrets depends on data, data depends on control flow, control flow depends on data…

• A high level of reliability is required
  • semantic justifications, Coq mechanizations…

• Attackers exploit executable code, not source code
  • we need guaranties at the assembly level using a compiler toolchain
Background: verifying a compiler

CompCert, a moderately optimizing C compiler usable for critical embedded software

= compiler + proof that the compiler does not introduce bugs

Using the Coq proof assistant, X. Leroy proves the following semantic preservation property:

For all source programs S and compiler-generated code C, if the compiler generates machine code C from source S, without reporting a compilation error, then «C behaves like S». 
Background: verifying a compiler

CompCert, a moderately optimizing C compiler usable for critical embedded software

= compiler + proof that the compiler does not introduce bugs

Using the Coq proof assistant, X. Leroy proves the following semantic preservation property:

For all source programs S and compiler-generated code C, if the compiler generates machine code C from source S, without reporting a compilation error, then «C behaves like S». 

does not deal with the constant-time security property!
CompCert: 1 compiler, 11 languages

- CompCert C
- Clight
- C#minor
- RTL
- CminorSel
- Cminor
- LTL
- LTLin
- Linear
- ASM
- Mach

Optimizations: constant prop., CSE, tail calls, (LCM), (software pipelining)

- side-effects out of expressions
- type elimination loop simplifications
- stack allocation of «&» variables
- instruction selection
- (instruction scheduling)
- asm code generation
- layout of stack frames
- spilling, reloading calling conventions
- register allocation (IRC)
- linearization of the CFG
- format of the CFG

Diagram:

1. CompCert C → Clight
2. Clight → C#minor
3. RTL → CminorSel
4. CminorSel → Cminor
5. LTL → LTLin
6. LTLin → Linear
7. Linear → Mach
8. ASM → Mach
CompCert: 1 compiler, 11 languages

Optimizations: constant prop., CSE, tail calls, (LCM), (software pipelining)

Where should we perform the constant time analysis?
Our approach

1. Analyse the program at source level

Sandrine Blazy, David Pichardie, Alix Trieu. 
*Verifying Constant-Time Implementations by Abstract Interpretation.*
ESORICS 2017.

2. Make the compiler preserve the property

*Formal verification of a constant-time preserving C compiler.*
POPL 2020.
Constant-time analysis at source level

We perform static analysis at (almost) C level
  • Based on previous work with a value analyser, Verasco
  • We mix Verasco memory tracking with fine-grained tainting
  • Main difficulty: alias analysis taking into account pointer arithmetic

Sandrine Blazy, David Pichardie, Alix Trieu.
Verifying Constant-Time Implementations by Abstract Interpretation.
ESORICS 2017.

Cf next part on Abstract Interpretation
Preserving the property through compilation


- Makes precise what secure compilation means for cryptographic constant-time
- Provides a machine checked-proof that a mildly modified version of the CompCert compiler preserves cryptographic constant-time
- Explains how to turn a pre-existing formally-verified compiler into a formally-verified secure compiler
- Provides a proof toolkit for proving security preservation with simulation diagrams
CompCert: 1 compiler, 11 languages

**Optimizations:**
- constant prop., CSE, tail calls, (LCM), (software pipelining)
- side-effects out of expressions
- CFG construction expr. decomp.
- register allocation (IRC)
- linearization of the CFG

**Flowchart:**
- CompCert C → Clight
- Clight → C#minor
- RTL → CminorSel
- CminorSel → Cminor
- LTL → LTLin
- LTLin → Linear
- Linear → Mach
- Mach → ASM
- ASM → CompCert C

**Specifics:**
- type elimination loop simplifications
- spilling, reloading calling conventions
- stack allocation of «&» variables
- instruction selection
- instruction scheduling
- layout of stack frames
- asm code generation
CompCert preservation proof methodology

- Each language is given an operational semantics $s \xrightarrow{t} s'$ that models a small step transition from a state $s$ to a state $s'$ by emitting a trace of external events $t$.

- From this stems a notion of program behavior (event trace) for complete (possibly infinite) executions.

- Behavior preservation is proved via backward and forward simulation, but thanks to language determinism, forward simulation is enough.
CompCert preservation proof methodology

- Each language is given an operational semantics $s \xrightarrow{t} s'$ that models a small step transition from a state $s$ to a state $s'$ by emitting a trace of external events $t$.

- From this stems a notion of program behavior (event trace) for complete (possibly infinite) executions.

- Behavior preservation is proved via backward and forward simulation, but thanks to language determinism, forward simulation is enough.
CompCert preservation proof methodology

• Each language is given an **operational semantics** $s \xrightarrow{t} s'$ that models a small step transition from a state $s$ to a state $s'$ by emitting a trace of external events $t$.

• From this stems a notion of **program behavior** (event trace) for complete (possibly infinite) executions.

• Behavior preservation is proved via backward and forward simulation, but thanks to language determinism, **forward simulation** is enough.
CompCert preservation proof methodology

- Each language is given an **operational semantics** $s \xrightarrow{t} s'$ that models a small step transition from a state $s$ to a state $s'$ by emitting a trace of external events $t$.

- From this stems a notion of **program behavior** (event trace) for complete (possibly infinite) executions.

- Behavior preservation is proved via backward and forward simulation, but thanks to language determinism, **forward simulation** is enough.

\[
\begin{align*}
\sigma_1 \xrightarrow{t} \sigma_2 \quad &\text{or} \quad \sigma_1 \xrightarrow{t} \sigma_2 \approx \sigma_1 \approx \sigma_2 \quad \text{with } t = \epsilon \\
\text{and } m(s_2) < m(s_1) \quad &\text{well founded measure}
\end{align*}
\]
### CompCert: 17 Preservations Proofs

<table>
<thead>
<tr>
<th>Compiler pass</th>
<th>Explanation on the pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cshmgen</td>
<td>Type elaboration, simplification of control</td>
</tr>
<tr>
<td>Cminorgen</td>
<td>Stack allocation</td>
</tr>
<tr>
<td>Selection</td>
<td>Recognition of operators and addr. modes</td>
</tr>
<tr>
<td>RTLgen</td>
<td>Generation of CFG and 3-address code</td>
</tr>
<tr>
<td>Tailcall</td>
<td>Tailcall recognition</td>
</tr>
<tr>
<td>Inlining</td>
<td>Function inlining</td>
</tr>
<tr>
<td>Renumber</td>
<td>Renumbering CFG nodes</td>
</tr>
<tr>
<td>ConstProp</td>
<td>Constant propagation</td>
</tr>
<tr>
<td>CSE</td>
<td>Common subexpression elimination</td>
</tr>
<tr>
<td>Deadcode</td>
<td>Redundancy elimination</td>
</tr>
<tr>
<td>Allocation</td>
<td>Register allocation</td>
</tr>
<tr>
<td>Tunneling</td>
<td>Branch tunneling</td>
</tr>
<tr>
<td>Linearize</td>
<td>Linearization of CFG</td>
</tr>
<tr>
<td>CleanupLabels</td>
<td>Removal of unreferenced labels</td>
</tr>
<tr>
<td>Debugvar</td>
<td>Synthesis of debugging information</td>
</tr>
<tr>
<td>Stacking</td>
<td>Laying out stack frames</td>
</tr>
<tr>
<td>Asmgen</td>
<td>Emission of assembly code</td>
</tr>
</tbody>
</table>
Cryptographic constant-time property: defining leakages

- We enrich the CompCert traces of events with leakages of two types
  - either the truth value of a condition,
  - or a pointer representing the address of
    - either a memory access (i.e., a load or a store)
    - or a called function
- Using event erasure, from \( s \xrightarrow{t} s' \) we can extract
  - the compile-only judgment \( s \xrightarrow{t}\text{comp} s' \)
  - the leak-only judgment \( s \xrightarrow{t}\text{leak} s' \)
- **Program leakage** is defined as the behavior of the \( \xrightarrow{\text{leak}} \) semantics
Cryptographic constant-time property: preservation

- We note $\varphi(s, s')$ the fact that two initial states $s$ and $s'$ share the same values for public inputs, but may differ on the values of secret inputs.

- A program is **constant-time secure w.r.t.** $\varphi$ if for two initial states $s$ and $s'$ such that $\varphi(s, s')$ holds, then both leak-only executions starting from $s$ and $s'$ observe the same leakage.

\[ \varphi \quad \text{implies} \quad t = t' \]
Cryptographic constant-time property: preservation

- We note $\varphi(s, s')$ the fact that two initial states $s$ and $s'$ share the same values for public inputs, but may differ on the values of secret inputs.

- A program is **constant-time secure w.r.t.** $\varphi$ if for two initial states $s$ and $s'$ such that $\varphi(s, s')$ holds, then both leak-only executions starting from $s$ and $s'$ observe the same leakage.

---

**Main Theorem (Constant-Time security preservation):** Let $P$ be a safe Clight source program that is compiled into an x86 assembly program $P'$. If $P$ is constant-time w.r.t. $\varphi$, then so is $P'$. 
Take-away message
Take-away message

- Abstract Interpretation can secure program art source level
Take-away message

- Abstract Interpretation can secure program art source level

- But we must make sure the compiler will preserve the security policy
Take-away message

• Abstract Interpretation can secure program art source level

• But we must make sure the compiler will preserve the security policy
Take-away message

- Abstract Interpretation can secure program art source level

- But we must make sure the compiler will preserve the security policy

- Let’s start the Abstract Interpretation lecture!
Abstract Interpretation (an introduction)
Static program analysis

The goals of static program analysis

- to prove properties about the run-time behaviour of a program
- in a fully automatic way
- without actually executing this program

Applications

- code optimisation
- error detection (array out of bound access, null pointers)
- proof support (invariant extraction)
Abstract Interpretation

[A theory which unifies a large variety of static analysis

formalises the approximated analyse of programs

allows to compare relative precision of analyses

facilitates the conception of sophisticated analyses

1. See http://www.di.ens.fr/~cousot/
Static analysis computes approximations\(^2\)

- \( P \) is safe w.r.t. \( \phi_1 \) and the analyser proves it
  \[ [[P]] \cap \phi_1 = \emptyset \quad \text{and} \quad [[P]]_{\text{approx}} \cap \phi_1 = \emptyset \]

- \( P \) is unsafe w.r.t. \( \phi_2 \) and the analyser warns about it
  \[ [[P]] \cap \phi_2 \neq \emptyset \quad \text{and} \quad [[P]]_{\text{approx}} \cap \phi_2 \neq \emptyset \]

- **but** \( P \) is safe w.r.t. \( \phi_3 \) and the analyser can’t prove it (this is called a *false alarm*)
  \[ [[P]] \cap \phi_3 = \emptyset \quad \text{and} \quad [[P]]_{\text{approx}} \cap \phi_3 \neq \emptyset \]

\[ [[P]]: \quad \text{concrete semantics (e.g. set of reachable states)} \quad \text{(not computable)} \]
\[ \phi_1, \phi_2, \phi_3: \quad \text{erroneous/dangerous set of states} \quad \text{(computable)} \]
\[ [[P]]_{\text{approx}}: \quad \text{analyser result (here over-approximation)} \quad \text{(computable)} \]

---

2. see [https://www.di.ens.fr/~cousot/AI/IntroAbsInt.html](https://www.di.ens.fr/~cousot/AI/IntroAbsInt.html)
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in \( P(\mathbb{Z}^2) \) of \((x, y)\) values.
- When a point is reached for a second time we make an union with the previous property.
- We “execute” the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```plaintext
x = 0; y = 0;
{
}
while (x<6) {
  if (?) {
    {
      }
    y = y+2;
    {
    }
  }
};
{
  }
  x = x+1;
  {
  }
}
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(x, y)$ values.
- When a point is reached for a second time we make an union with the previous property.
- We “execute” the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```plaintext
x = ∅; y = ∅;
{(0,0)}
while (x<6) {
  if (?) {
    {  
      {y = y+2; }  
    }
  }  
  {x = x+1; }  
}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(x, y)$ values.
- When a point is reached for a second time we make an union with the previous property.
- We ”execute” the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```c
x = 0; y = 0;
{ (0, 0) }
while (x < 6) {
  if (?) {
    { (0, 0) }
    y = y + 2;
    {
    }
  };
  {
  }
  x = x + 1;
  { }
}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(x, y)$ values.
- When a point is reached for a second time we make an union with the previous property.
- We “execute” the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```plaintext
x = 0; y = 0;
{(0,0)}

while (x<6) {
  if (?) {
    {(0,0)}
    y = y+2;
    {(0,2)}
  };

  x = x+1;
  {(1,0), (1,2), (2,0), (2,2), (2,4),...}
}
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

**Collecting semantics**

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(x, y)$ values.
- When a point is reached for a second time we make an union with the previous property.
- We "execute" the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```cpp
x = 0; y = 0;
{(0,0)}
while (x<6) {
  if (?) {
    {(0,0)}
    y = y+2;
    {(0,2)}
  };
    {(0,0),(0,2)}
  x = x+1;
    { }
}  ```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in \( \mathcal{P}(\mathbb{Z}^2) \) of \((x, y)\) values.
- When a point is reached for a second time we make an union with the previous property.
- We "execute" the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```
x = 0; y = 0;
{x,
 ((0, 0),
 ((0, 0),
 (0, 2)
 )
 }
while (x<6) {
  if (?) {
    y = y+2;
    {((0, 0), (0, 2))
    }
  }
  x = x+1;
  {((1, 0), (1, 2))
  }
}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(x, y)$ values.
- When a point is reached for a second time we make an union with the previous property.
- We “execute” the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```cpp
x = 0; y = 0;  
{ (0,0), (1,0), (1,2) }  
while (x<6) {  
  if (?) {  
    { (0,0) }  
    y = y+2;  
    { (0,2) }  
  };  
  { (0,0), (0,2) }  
  x = x+1;  
  { (1,0), (1,2) }  
}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in \( \mathcal{P}(\mathbb{Z}^2) \) of \((x, y)\) values.
- When a point is reached for a second time we make an union with the previous property.
- We "execute" the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```plaintext
x = \emptyset; y = \emptyset;
{(0,0),(1,0),(1,2) }
while (x<6) {
  if (?) {
    {(0,0),(1,0),(1,2) }
    y = y+2;
    {(0,2) }
  }
};
{(0,0),(0,2) }
x = x+1;
{(1,0),(1,2) }
}```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(x, y)$ values.
- When a point is reached for a second time we make an union with the previous property.
- We "execute" the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```plaintext
x = 0; y = 0;
{ (0,0), (1,0), (1,2) }

while (x<6) {
  if (?) {
    { (0,0), (1,0), (1,2) }
    y = y+2;
    { (0,2), (1,2), (1,4) }
  }
  { (0,0), (0,2) }
  x = x+1;
  { (1,0), (1,2) }
}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(x, y)$ values.
- When a point is reached for a second time we make an union with the previous property.
- We "execute" the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```plaintext
x = 0; y = 0;
{(0,0), (1,0), (1,2) }
while (x<6) {
  if (?) {
    {(0,0), (1,0), (1,2) }
    y = y+2;
    {(0,2), (1,2), (1,4) }
  }
  {(0,0), (0,2), (1,0), (1,2), (1,4) }
  x = x+1;
  {(1,0), (1,2) }
}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in \( \mathcal{P}(\mathbb{Z}^2) \) of \((x, y)\) values.
- When a point is reached for a second time we make an union with the previous property.
- We “execute” the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```c
x = 0; y = 0;
{(0,0), (1,0), (1,2) }
while (x<6) {
  if (?) {
    {(0,0), (1,0), (1,2) }
    y = y+2;
    {(0,2), (1,2), (1,4) }
  }
  {(0,0), (0,2), (1,0), (1,2), (1,4) }
  x = x+1;
  {(1,0), (1,2), (2,0), (2,2), (2,4) }
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Collecting semantics

- A state property is a subset in \( \mathcal{P}(\mathbb{Z}^2) \) of \((x, y)\) values.
- When a point is reached for a second time we make an union with the previous property.
- We “execute” the program until stability
  - It may take an infinite number of steps...
  - But the limit always exists (explained later)

```plaintext
x = 0; y = 0;
{(0,0), (1,0), (1,2), ...}

while (x<6) {
  if (?) {
    {(0,0), (1,0), (1,2), ...}
    y = y+2;
    {(0,2), (1,2), (1,4), ...}
  };
    {(0,0), (0,2), (1,0), (1,2), (1,4), ...}
  x = x+1;
    {(1,0), (1,2), (2,0), (2,2), (2,4), ...}
}
{(6,0), (6,2), (6,4), (6,6), ...}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[
P ::= x \in C \ 0 \land y \in C \ 0
\]

\[
C ::= < | \leq | = | > | \geq
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

\[
x = 0; \ y = 0;
\]

\[
x = 0 \land y = 0
\]

\[
\text{while } (x < 6) \{ \\
\quad \text{if } (?) \{ \\
\quad \quad y = y + 2; \\
\quad \}; \\
\quad x = x + 1; \\
\};
\]
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

**Approximation**

- The set of manipulated properties may be restricted to ensure computability of the semantics.

**Example: sign of variables**

\[
P := x C 0 \land y C 0
\]
\[
C := < \mid \leq \mid = \mid > \mid \geq
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

\[
x = 0; y = 0;
x = 0 \land y = 0
\]
\[
\textbf{while} \ (x<6) \ {\}
\]
\[
\quad \textbf{if} \ (?) \ {\}
\]
\[
\quad \quad x = 0 \land y = 0
\]
\[
\quad y = y+2;
\]
\[
\}
\]
\[
\]
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[
P ::= x \in C 0 \land y \in C 0
\]

\[
C ::= < | \leq | = | > | \geq
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

```plaintext
x = 0; y = 0;
  x = 0 \land y = 0

while (x<6) {
  if (?) {
    x = 0 \land y = 0
    y = y+2;
    x = 0 \land y > 0 \text{ over-approximation!}
  }
  x = x+1;
}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[
\begin{align*}
P & ::= x \lor 0 \land y \lor 0 \\
C & ::= < | | | | > | | \\
\end{align*}
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

\[
\begin{align*}
x & = 0; y = 0; \\
x & = 0 \land y = 0 \\
\textbf{while} (x < 6) \{ \\
\quad \textbf{if} (?) \{ \\
\qquad x & = 0 \land y = 0 \\
\qquad y & = y + 2; \\
\qquad x & = 0 \land y > 0 \\
\quad \}; \\
\quad x & = 0 \land y \geq 0 \\
\qquad x & = x + 1; \\
\}; \\
\end{align*}
\]
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[
\begin{align*}
P & ::= \ x C 0 \land y C 0 \\
C & ::= \ < | \leq | = | > | \geq
\end{align*}
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

\[
x = 0; y = 0; \\
x = 0 \land y = 0
\]

\[
\textbf{while}\ (x<6)\{ \\
\quad \textbf{if}\ (\)\{ \\
\quad \quad x = 0 \land y = 0 \\
\quad y = y+2; \\
\quad \quad x = 0 \land y > 0 \\
\quad \}; \\
\quad x = x+1; \\
\quad x > 0 \land y \geq 0 \text{ over-approximation!}
\]

}
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[
P ::= x \in [0, 0] \land y \in [0, 0]
\]

\[
C ::= < | \leq | = | > | \geq
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

```plaintext
x = 0; y = 0;

x \geq 0 \land y \geq 0

while (x < 6) {
  if (?) {
    x = 0 \land y = 0
    y = y + 2;
    x = 0 \land y > 0
  }

  x = x + 1;
  x > 0 \land y \geq 0
}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[
P ::= x \geq 0 \land y \geq 0
\]

\[
C ::= < | \leq | = | > | \geq
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

\[
x = 0; y = 0;
x \geq 0 \land y \geq 0
\]

while \( x < 6 \) {

\[
if (\text{?}) {

\]

\[
x \geq 0 \land y \geq 0
\]

\[
y = y + 2;
\]

\[
x = 0 \land y > 0
\]

\[
};
\]

\[
x = 0 \land y \geq 0
\]

\[
x = x + 1;
\]

\[
x > 0 \land y \geq 0
\]

\]
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[
P ::= \ x \ C \ 0 \ \land \ y \ C \ 0 \\
C ::= \ < \mid \leq \mid = \mid > \mid \geq
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

\[
x = 0; \ y = 0; \\
x \geq 0 \ \land \ y \geq 0 \\
\textbf{while} \ (x<6) \{ \\
\quad \textbf{if} \ (?) \{ \\
\quad \quad x \geq 0 \ \land \ y \geq 0 \\
\quad \quad y = y+2; \\
\quad \quad x \geq 0 \ \land \ y > 0 \\
\quad \}; \\
\quad x = 0 \ \land \ y \geq 0 \\
\quad x = x+1; \\
\quad x > 0 \ \land \ y \geq 0 \\
\}
\]
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[
P ::= \quad x \in \mathbb{C} \land y \in \mathbb{C}
\]

\[
C ::= \quad < | \leq | = | > | \geq
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

```plaintext
x = 0; y = 0;
\]
x \geq 0 \land y \geq 0
\]
while (x < 6) {
\]
if (?) {
\]
x \geq 0 \land y \geq 0
\]
y = y + 2;
\]
x \geq 0 \land y > 0
\];
\]
x \geq 0 \land y \geq 0
\]
x = x + 1;
\]
x > 0 \land y \geq 0
\}
```
A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[
\begin{align*}
P & ::= x \mathcal{C} 0 \land y \mathcal{C} 0 \\
C & ::= < | \leq | = | > | \geq
\end{align*}
\]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

\[
\begin{align*}
x & = 0; \ y = 0; \\
x & \geq 0 \land y \geq 0 \\
\text{while} \ (x < 6) \ {\{} \\
\text{if} \ (?) \ {\{} \\
\quad x & \geq 0 \land y \geq 0 \\
\quad y & = y + 2; \\
\quad x & \geq 0 \land y > 0 \\
\}; \\
\quad x & \geq 0 \land y \geq 0 \\
x & = x + 1; \\
x & > 0 \land y \geq 0 \\
\}
\]

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of states.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.

Example: sign of variables

\[ P ::= x \leq 0 \land y \leq 0 \]

\[ C ::= < | \leq | = | > | \geq \]

- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
  x \geq 0 \land y \geq 0
while (x<6) {
  if (?) {
    x \geq 0 \land y \geq 0
    y = y+2;
    x \geq 0 \land y > 0
  }
};
  x \geq 0 \land y \geq 0
x = x+1;
  x > 0 \land y \geq 0
}  x \geq 0 \land y \geq 0
```
An other example: the interval analysis

For each point $k$ and each numeric variable $x$, we infer an interval in which $x$ must belong to.

Example: insertion sort, array access verification

```plaintext
assume(T.length=100); i=1; {i ∈ [1,100]}

while (i<T.length) {
    p = T[i]; j = i-1; {i ∈ [1,99]}
    while (0<=j and T[j]>p) {
        T[j]=T[j+1]; j = j-1; {i ∈ [1,99], j ∈ [−1,97]}
    }
    T[j+1]=p; i = i+1; {i ∈ [2,100], j ∈ [−1,98]}
}
```
An other example: the polyhedral analysis

For each point $k$ and we infer invariant linear equality and inequality relationships among variables.

Example: insertion sort, array access verification

```plaintext
assume(T.length>=1); i=1;

while i<T.length {
    p = T[i]; j = i-1;
    while 0<=j and T[j]>$p$ {
        T[j]=T[j+1]; j = j-1;
    }
    T[j+1]=p; i = i+1;
}
```
This lecture

1. Introduction
2. Intermediate representation: syntax and semantics
3. Collecting semantics
4. Just put some #...
5. Building a generic abstract interpreter
6. Numeric abstraction by intervals
7. Widening/Narrowing
8. Polyhedral abstract interpretation
9. Readings
Outline

1. Introduction

2. Intermediate representation: syntax and semantics

3. Collecting semantics

4. Just put some #...

5. Building a generic abstract interpreter

6. Numeric abstraction by intervals

7. Widening/Narrowing

8. Polyhedral abstract interpretation

9. Readings
A flowchart representation of program

The standard model of program in static analysis is \textit{control flow graph}. The graph model used here:

- the nodes are program point $k \in \mathcal{P}$,
- the edges are labeled with \textit{basic instructions}.

\begin{align*}
\text{Instr} & : = \ x \ := \ Exp \quad \text{assignment} \\
& \quad | \quad \text{nop} \\
& \quad | \quad \text{assume } \text{Test} \quad \text{execution continues only if} \\
& \quad \quad \quad \text{the test succeeds}
\end{align*}

(\text{Exp} \text{ and } \text{Test} \text{ to be defined in the next slide})

- formally a cfg is a couple $(k_{\text{init}}, S)$ with
  - $k_{\text{init}} \in \mathcal{P}$ : the entry point,
  - $S \subseteq \mathcal{P} \times \text{Instr} \times \mathcal{P}$ the set of edges.

Remark : data-flow analyses are generally based on other versions of control flow graph (nodes are put in instructions).
x = read_input()
if x<0 {
    while (x<0) x++
    y = x
} else {
    y = 0
}
Expression and test language for today

In OCaml syntax

We will restrict our study to a simple numeric subset of Java expressions

```ocaml
type binop =
  | Add | Sub | Mult

type expr =
  | Const of int
  | Var of var
  | Binop of binop * expr * expr

type comp = Eq | Neq | Le | Lt

type test =
  | Cond of expr * comp * expr (* e1 cmp e2 *)
  | And of test * test (* t1 && t2 *)
  | Or of test * test (* t1 || t2 *)

type instr =
  | Nop
  | Forget of var (* x := ? *)
  | Assign of var * expr (* x := e *)
  | Assume of test (* assume t *)
```
Semantics

Semantic domains

\[ Env \overset{\text{def}}{=} V \rightarrow \mathbb{Z} \]
\[ State \overset{\text{def}}{=} \mathbb{P} \times Env \]

Semantics of expressions (standard then omitted)

\[ A[e] \rho \in \mathbb{Z}, \quad e \in \text{Exp}, \quad \rho \in \text{Env} \]

Semantics of tests (standard then omitted)

\[ B[t] \rho \in \mathbb{B}, \quad t \in \text{Test}, \quad \rho \in \text{Env} \]
Small-step semantics of cfg

We first define the semantics of instructions: $\xrightarrow{i} \subseteq \mathit{Env} \times \mathit{Env}$

\[
\begin{align*}
v \in \mathbb{Z} & \quad \rho \xrightarrow{x := ?} \rho[x \leftarrow v] \\
\rho \xrightarrow{x := a} \rho[x \leftarrow A[a] \rho] & \quad \rho \xrightarrow{t} \rho
\end{align*}
\]

Then a small-step relation $\xrightarrow{cfg} \subseteq \mathit{State} \times \mathit{State}$ for a $\text{cfg} = (k_{\text{init}}, S)$

\[
(k_1, i, k_2) \in S \quad \rho_1 \xrightarrow{i} \rho_2 \quad \Rightarrow \quad (k_1, \rho_1) \xrightarrow{cfg} (k_2, \rho_2)
\]

Reachable states for control flow graphs

\[
\llbracket \text{cfg} \rrbracket = \{ (k, \rho) \mid \exists \rho_0 \in \mathit{Env}, (k_{\text{init}}, \rho_0) \xrightarrow{cfg}^* (k, \rho) \}
\]

where $\text{cfg} = (k_{\text{init}}, S)$
Starting from an other semantics?

Remark: for the purpose of the talk, we directly start with a \textit{cfg}-semantics. We could have started from a more conventional operational semantics. See


Outline

1. Introduction
2. Intermediate representation: syntax and semantics
3. Collecting semantics
4. Just put some # ...
5. Building a generic abstract interpreter
6. Numeric abstraction by intervals
7. Widening/Narrowing
8. Polyhedral abstract interpretation
9. Readings
Collecting Semantics

We will consider a collecting semantics that give us the set of reachable states $\llbracket p \rrbracket^\text{col}_k$ at each program points $k$.

$$\forall k \in P, \llbracket p \rrbracket^\text{col}_k = \{ \rho \mid (k, \rho) \in \llbracket p \rrbracket \}$$

**Theorem**

$\llbracket p \rrbracket^\text{col}$ may be characterized as the least fixpoint of the following equation system.

$$\forall k \in \text{labels}(p), \quad X_k = X_k^{\text{init}} \cup \bigcup_{(k',i,k) \in p} \llbracket i \rrbracket (X_{k'})$$

with $X_k^{\text{init}} = \begin{cases} \text{Env} & \text{if } k = k_{\text{init}} \\ \emptyset & \text{otherwise} \end{cases}$

and

$$\forall i \in \text{Instr}, \forall X \subseteq \text{Env}, \llbracket i \rrbracket (X) = \left\{ \rho_2 \mid \exists \rho_1 \in X, \rho_1 \xrightarrow{i} \rho_2 \right\} = \text{post} \left[ \xrightarrow{i} \right] (X)$$
Example

For the following program, $\llbracket P \rrbracket^{\text{col}}$ is the least solution of the following equation system:

$$\begin{align*}
X_0 &= Env \\
X_1 &= \llbracket x := ? \rrbracket (X_0) \\
X_2 &= \llbracket x < 0 \rrbracket (X_1) \cup X_4 \\
X_3 &= \llbracket x < 0 \rrbracket (X_2) \\
X_4 &= \llbracket x := x + 1 \rrbracket (X_3) \\
X_5 &= \llbracket x \geq 0 \rrbracket (X_2) \\
X_6 &= \llbracket y := x \rrbracket (X_5) \\
X_7 &= \llbracket x \geq 0 \rrbracket (X_1) \\
X_8 &= \llbracket y := 0 \rrbracket (X_7) \\
X_9 &= X_6 \cup X_8
\end{align*}$$
**Theorem (Knaster-Tarski)**

In a complete lattice \((A, \sqsubseteq, \sqcup)\), for all monotone functions \(f \in A \rightarrow A\), the least fixpoint \(\text{lfp}(f)\) of \(f\) exists and is \(\cap \{x \in A \mid f(x) \sqsubseteq x\}\).

**Theorem (Kleene fixpoint theorem)**

In a complete lattice \((A, \sqsubseteq, \sqcup)\), for all continuous function \(f \in A \rightarrow A\), the least fixpoint \(\text{lfp}(f)\) of \(f\) is equal to \(\cup \{f^n(\bot) \mid n \in \mathbb{N}\}\).

**Theorem**

Let \((A, \sqsubseteq)\) a poset that verifies the ascending chain condition and \(f\) a monotone function. The sequence \(\bot, f(\bot), \ldots, f^n(\bot), \ldots\) eventually stabilises. Its limit is the least fixpoint of \(f\).
Collecting semantics and exact analysis

The \((X_k)_{i=1..N}\) are hence specified as the least solution of a fixpoint equation system

\[ X_k = F_k(X_1, X_2, \ldots, X_N) , \ k \in \text{labels}(p) \]

or, equivalently \(\vec{X} = \vec{F}(\vec{X})\).

Exact analysis:

- Thanks to Knaster-Tarski, the least solution exists (complete lattice, \(F_k\) are monotone functions),
- Kleen fixpoint theorem (\(F_k\) are continuous functions) says it is the limit of

\[
X_k^0 = \emptyset , \ X_k^{n+1} = F_k(X_1^n, X_2^n, \ldots, X_N^n)
\]

Uncomputable problem:

- Representing the \(X_k\) may be hard (infinite sets)
- The limit may not be reachable in a finite number of steps
Approximate analysis

Exact analysis:
Least solution of $X = F(X)$ in the complete lattice $(\mathcal{P}(Env)^N, \subseteq, \cup, \cap)$
or limit of $X^0 = \perp, X^{n+1} = F(X^n)$

Approximate analysis:

- **Static approximation**: we replace the concrete lattice $(\mathcal{P}(Env), \subseteq, \cup, \cap)$ by an abstract lattice $(L^#, \subseteq^#, \cup^#, \cap^#)$
  - whose elements can be (efficiently) represented in computers,
  - in which we know how to compute $\cup^#, \cap^#, \subseteq^#, \ldots$

and we “transpose” the equation $X = F(X)$ of $\mathcal{P}(Env)^N$ into $(L^#)^N$.

- **Dynamic approximation**: when $L^#$ does not verify the ascending chain condition, the iterative computation may not terminate in a finite number of steps (or sometimes too slowly). In this case, we can only approximate the limit (see widening/narrowing).
Outline

1. Introduction
2. Intermediate representation: syntax and semantics
3. Collecting semantics
4. Just put some # ...
5. Building a generic abstract interpreter
6. Numeric abstraction by intervals
7. Widening/Narrowing
8. Polyhedral abstract interpretation
9. Readings
Just put some …

From $\mathcal{P}(\text{Env})$ to $\text{Env}^\#$

control flow graph

collecting semantics

abstract semantics

\[
\begin{align*}
X_0 &= \text{Env} & X_0^\# &= \top_{\text{Env}} \\
X_1 &= \llbracket x :=? \rrbracket (X_0) & X_1^\# &= \llbracket x :=? \rrbracket^\# (X_0^\#) \\
X_2 &= \llbracket x < 0 \rrbracket (X_1) \cup X_4 & X_2^\# &= \llbracket x < 0 \rrbracket^\# (X_1^\#) \cup^\# X_4^\# \\
X_3 &= \llbracket x < 0 \rrbracket (X_2) & X_3^\# &= \llbracket x < 0 \rrbracket^\# (X_2^\#) \\
X_4 &= \llbracket x := x + 1 \rrbracket (X_3) & X_4^\# &= \llbracket x := x + 1 \rrbracket^\# (X_3^\#) \\
X_5 &= \llbracket x \geq 0 \rrbracket (X_2) & X_5^\# &= \llbracket x \geq 0 \rrbracket^\# (X_2^\#) \\
X_6 &= \llbracket y := x \rrbracket (X_5) & X_6^\# &= \llbracket y := x \rrbracket^\# (X_5^\#) \\
X_7 &= \llbracket x \geq 0 \rrbracket (X_1) & X_7^\# &= \llbracket x \geq 0 \rrbracket^\# (X_1^\#) \\
X_8 &= \llbracket y := 0 \rrbracket (X_7) & X_8^\# &= \llbracket y := 0 \rrbracket^\# (X_7^\#) \\
X_9 &= X_6 \cup X_8 & X_9^\# &= X_6^\# \cup^\# X_8^\#
\end{align*}
\]
Abstract semantics: the ingredients

- A lattice structure \((Env^#, \sqsubseteq_{Env^#}, \sqcup_{Env^#}, \sqcap_{Env^#}, \bot_{Env^#}, \top_{Env^#})\)
  - \(\sqsubseteq_{Env^#}\) is an approximation of \(\subseteq\)
  - \(\sqcup_{Env^#}\) is an approximation of \(\cup\)
  - \(\sqcap_{Env^#}\) is an approximation of \(\cap\)
  - \(\bot_{Env^#}\) is an approximation of \(\emptyset\)
  - \(\top_{Env^#}\) is an approximation of \(Env\)

- For all \(x \in \mathbb{V}\),
  \[\llbracket x := ? \rrbracket^# \in Env^# \rightarrow Env^#\] an approximation of \(\llbracket x := ? \rrbracket\)

- For all \(x \in \mathbb{V}, e \in Exp\),
  \[\llbracket x := e \rrbracket^# \in Env^# \rightarrow Env^#\] an approximation of \(\llbracket x := e \rrbracket\)

- For all \(t \in Test\),
  \[\llbracket t \rrbracket^# \in Env^# \rightarrow Env^#\] an approximation of \(\llbracket t \rrbracket\)

- A concretisation \(\gamma \in Env^# \rightarrow \mathcal{P}(Env)\) that explains which property \(\gamma(x^#) \in \mathcal{P}(Env)\) is represented by each abstract element \(x^# \in Env^#\).
An abstraction by signs

\[
\begin{array}{c}
\top \\
-0 \\
\downarrow \\
- \\
0 \\
\downarrow \\
-0 \\
+0 \\
\downarrow \\
0 \\
\downarrow \\
+ \\
+0 \\
\end{array}
\]

\[
\begin{array}{ll}
\bot & \text{represents the property } \emptyset \\
- & \text{represents the property } \{ z \mid z < 0 \} \\
0 & \text{represents the property } \{0\} \\
+ & \text{represents the property } \{ z \mid z > 0 \} \\
-0 & \text{represents the property } \{ z \mid z \leq 0 \} \\
+0 & \text{represents the property } \{ z \mid z \geq 0 \} \\
\top & \text{represents the property } \mathbb{Z}
\end{array}
\]

\[\text{Env}^\# \overset{\text{def}}{=} \mathcal{V} \rightarrow \text{Sign} : \text{a sign is associated to each variable.}\]
An abstraction by signs: example

\[
\begin{align*}
X_0^\# &= \top_{Env}^\
X_1^\# &= \llbracket x := ? \rrbracket^\# (X_0^\#) \\
X_2^\# &= \llbracket x < 0 \rrbracket^\# (X_1^\#) \sqcup^\# X_4^\# \\
X_3^\# &= \llbracket x < 0 \rrbracket^\# (X_2^\#) \\
X_4^\# &= \llbracket x := x + 1 \rrbracket^\# (X_3^\#) \\
X_5^\# &= \llbracket x \geq 0 \rrbracket^\# (X_2^\#) \\
X_6^\# &= \llbracket y := x \rrbracket^\# (X_5^\#) \\
X_7^\# &= \llbracket x \geq 0 \rrbracket^\# (X_1^\#) \\
X_8^\# &= \llbracket y := 0 \rrbracket^\# (X_7^\#) \\
X_9^\# &= X_6^\# \sqcup^\# X_8^\#
\end{align*}
\]

which simplifies into

\[
\begin{align*}
X_0^\# &= \llbracket x : \top; y : \top \rrbracket \\
X_1^\# &= X_0^\# [x \mapsto \top] \\
X_2^\# &= X_1^\# [x \mapsto -] \sqcup^\# X_4^\# \\
X_3^\# &= X_2^\# [x \mapsto -] \\
X_4^\# &= X_3^\# [x \mapsto \text{succ}^\#(X_3^\#(x))] \\
X_5^\# &= X_2^\# [x \mapsto +0] \\
X_6^\# &= X_5^\# [y \mapsto X_5^\#(x)] \\
X_7^\# &= X_1^\# [x \mapsto +0] \\
X_8^\# &= X_7^\# [y \mapsto 0] \\
X_9^\# &= X_6^\# \sqcup^\# X_8^\#
\end{align*}
\]

with

\[
\begin{align*}
\text{succ}^\#(\bot) &= \bot \\
\text{succ}^\#(-) &= -0 \\
\text{succ}^\#(0) &= \text{succ}^\#(+) = \text{succ}^\#(+0) = + \\
\text{succ}^\#(-0) &= \text{succ}^\#(\top) = \top
\end{align*}
\]
Abstraction by intervals

\[
\text{Int} \stackrel{\text{def}}{=} \{ [a, b] \mid a, b \in \overline{\mathbb{Z}}, a \leq b \} \cup \{ \bot \}
\]

with \( \overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\} \).

\( \bot \) represents \( \emptyset \) and \([a, b]\) the property \([z \mid a \leq z \leq b]\).

\[\begin{align*}
[-3, -1] & \quad [-2, 0] \quad [-1, 1] \quad [0, 2] \quad [1, 3] \\
[-3, -2] & \quad [-2, -1] \quad [-1, 0] \quad [0, 1] \quad [1, 2] \quad [2, 3] \\
[-3, -3] & \quad [-2, -2] \quad [-1, -1] \quad [0, 0] \quad [1, 1] \quad [2, 2] \quad [3, 3]
\end{align*}\]

\( Env^\# \stackrel{\text{def}}{=} V \rightarrow \text{Int} : \) an interval is associated to each variable.
Abstraction by intervals: example

\[
\begin{align*}
X_0^\# &= \top_{\text{Env}}^\# \\
X_1^\# &= \llbracket x := \text{?} \rrbracket^\# (X_0^\#) \\
X_2^\# &= \llbracket x < 0 \rrbracket^\# (X_1^\#) \sqcup^\#X_4^\# \\
X_3^\# &= \llbracket x < 0 \rrbracket^\# (X_2^\#) \\
X_4^\# &= \llbracket x := x + 1 \rrbracket^\# (X_3^\#) \\
X_5^\# &= \llbracket x \geq 0 \rrbracket^\# (X_2^\#) \\
X_6^\# &= \llbracket y := x \rrbracket^\# (X_5^\#) \\
X_7^\# &= \llbracket x \geq 0 \rrbracket^\# (X_1^\#) \\
X_8^\# &= \llbracket y := 0 \rrbracket^\# (X_7^\#) \\
X_9^\# &= X_6^\# \sqcup^\#X_8^\#
\end{align*}
\]

with

\[
\begin{align*}
\text{succ}^\#(\bot) &= \bot \\
\text{succ}^\#([a, b]) &= [a + 1, b + 1]
\end{align*}
\]
Outline

1. Introduction
2. Intermediate representation: syntax and semantics
3. Collecting semantics
4. Just put some #...
5. Building a generic abstract interpreter
6. Numeric abstraction by intervals
7. Widening/Narrowing
8. Polyhedral abstract interpretation
9. Readings
Given an environment concretisation function $\gamma_{\text{Env}} \in \text{Env}^\# \to \mathcal{P}(\text{Env})$, we want to compute an abstract semantics $\llbracket P \rrbracket^\# : \mathcal{P} \to \text{Env}^\#$ that is a conservative approximation of $\llbracket P \rrbracket^\text{col}$.

$$\forall k \in \mathcal{P}, \llbracket P \rrbracket^\text{col}(k) \subseteq \gamma(\llbracket P \rrbracket^\#(k))$$

This leads to a sound over-approximation of $\llbracket P \rrbracket$ since $\llbracket P \rrbracket$ and $\llbracket P \rrbracket^\text{col}$ are equivalents.

$$\llbracket P \rrbracket = \{ (k, \rho) \mid \rho \in \llbracket p \rrbracket^\text{col}(k) \}$$
Function approximation

When some computations in the concrete world are uncomputable or too costly, the abstract world can be used to execute a simplified version of these computations.

- the abstract computation must always give a conservative answer w.r.t. the concrete computation

Let \( f \in \mathcal{A} \rightarrow \mathcal{A} \) in the concrete world and \( f^\# \in \mathcal{A}^\# \rightarrow \mathcal{A}^\# \) which correctly approximates each concrete computation.

\[
\begin{align*}
\mathcal{A} & \xrightarrow{f} \mathcal{A} \\
\uparrow \gamma & \quad \uparrow \gamma \\
\mathcal{A}^\# & \xrightarrow{f^\#} \mathcal{A}^\#
\end{align*}
\]

Correctness criterion : \( f \circ \gamma \sqsubseteq \gamma \circ f^\# \)
Theorem

Given a monotone concretisation between two complete lattices \((A^\#, \sqsubseteq^\#, \sqcup^\#, \sqcap^\#) \rightarrow (A, \sqsubseteq, \sqcup, \sqcap)\), a function \(f^\# \in A^\# \rightarrow A^\#\) and a monotone function \(f \in A \rightarrow A\) which verify \(f \circ \gamma \sqsubseteq \gamma \circ f^\#\), we have

\[
lfp(f) \sqsubseteq \gamma(lfp(f^\#))
\]

It means it is generally sound to mimic fixpoint computation in the abstract.
Environment abstraction : sufficient elements

Thanks to the previous theorem, it is sufficient to design an abstraction domain $Env^\#$ with a correct approximation $\llbracket i \rrbracket^\#$ of $\llbracket i \rrbracket$ for all instructions $i$.

$$\forall \rho^\# \in Env^\#, \llbracket i \rrbracket (\gamma_{Env}(\rho^\#)) \subseteq \gamma_{Env}(\llbracket i \rrbracket^\# (\rho^\#))$$

And $\llbracket P \rrbracket^\#$ is defined as the least fixpoint of the system :

$$\forall k \in \text{labels}(P), \ X^\#_k = X^\#_{k}^{\text{init}} \sqcup \bigsqcup_{(k',i,k) \in P} \llbracket i \rrbracket^\# (X^\#_{k'})$$

with $X^\#_{k}^{\text{init}} = \begin{cases} \top_{Env} & \text{if } k = k_{\text{init}} \\ \emptyset & \text{otherwise} \end{cases}$
A Generic Abstract Interpreter

Non-relational Environment Abstraction

Numeric Abstraction

Generic fixpoint solver
Non-relational environment abstraction

We start with the description of a non-relational abstraction: each variable is abstracted independently.

\[ Env^\# \overset{\text{def}}{=} V \rightarrow \text{Num}^\# \]

\[ \forall \rho_1^\#, \rho_2^\# \in Env^\#, \ \rho_1^\# \sqsubseteq_{Env}^\# \rho_2^\# \overset{\text{def}}{=} \forall x \in V, \ \rho_1^\#(x) \sqsubseteq_{\text{Num}}^\# \rho_2^\#(x) \]

\[ \forall \rho^\# \in Env^\#, \ \gamma_{Env}(\rho^\#) \overset{\text{def}}{=} \{ \rho \mid \forall x \in V, \ \rho(x) \in \gamma_{\text{Num}}(\rho^\#(x)) \} \]

See the end of the lecture for a relational abstraction.
Sign abstraction

We will use this abstract domain as running example but you should keep in mind this is just an example among other numerical abstract domains.

\[
\begin{align*}
\gamma_{\text{Num}}(\bot) &= \emptyset \\
\gamma_{\text{Num}}(-) &= \{ z \mid z < 0 \} \\
\gamma_{\text{Num}}(0) &= \{ 0 \} \\
\gamma_{\text{Num}}(+) &= \{ z \mid z > 0 \} \\
\gamma_{\text{Num}}(-0) &= \{ z \mid z \leq 0 \} \\
\gamma_{\text{Num}}(+0) &= \{ z \mid z \geq 0 \} \\
\gamma_{\text{Num}}(\top) &= \mathbb{Z}
\end{align*}
\]
Construction of $\llbracket x := ? \rrbracket^*$

$$\llbracket x := ? \rrbracket^* (\rho^*) = \rho^*[x \mapsto T_{\text{Num}}], \ \forall \rho^* \in Env^*$$

with $T_{\text{Num}} \in \text{Num}^*$ such that $\mathbb{Z} \subseteq \gamma_{\text{Num}}(T_{\text{Num}})$. 

Construction of $\llbracket x := e \rrbracket^\#$

$\llbracket x := e \rrbracket^\# (\rho^\#) = \rho^\#[x \mapsto A[\llbracket e \rrbracket^\# (\rho^\#)], \forall \rho^\# \in Env^\#$

with

$\forall e \in \text{Expr}, A[\llbracket e \rrbracket^\# \in Env^\# \rightarrow \text{Num}^\#$

a (forward) abstract evaluation of expressions

$A[\llbracket n \rrbracket^\# (\rho^\#)] = \text{const}^\# (n)$

$A[\llbracket x \rrbracket^\# (\rho^\#)] = \rho^\# (x)$

$A[\llbracket e_1 \circ e_2 \rrbracket^\# (\rho^\#)] = o^\# (A[\llbracket e_1 \rrbracket^\# (\rho^\#), A[\llbracket e_2 \rrbracket^\# (\rho^\#)])$
Required operators on the numeric abstraction

- $\mathsf{const}^\# \in \mathsf{Num} \rightarrow \mathsf{Num}^\#$ computes an approximation of constants

$$
\forall n \in \mathbb{Z}, \{n\} \subseteq \gamma_{\mathsf{Num}}(\mathsf{const}^\#(n))
$$

- $\top_{\mathsf{Num}} \in \mathsf{Num}^\#$ approximates any numeric value

$$
\mathbb{Z} \subseteq \gamma_{\mathsf{Num}}(\top_{\mathsf{Num}})
$$

- $o^\# \in \mathsf{Num}^\# \times \mathsf{Num}^\# \rightarrow \mathsf{Num}^\#$ is a correct approximation of the arithmetic operators $o \in \{+,-,\times\}$

$$
\forall n_1^\#, n_2^\# \in \mathsf{Num}^\#,
\{ n_1 \circ n_2 \mid n_1 \in \gamma_{\mathsf{Num}}(n_1^\#), n_2 \in \gamma_{\mathsf{Num}}(n_2^\#) \} \subseteq \gamma_{\mathsf{Num}}(o^\#(n_1^\#, n_2^\#))
$$
Example: sign abstract domain

\[ \text{const}^#(n) = \begin{cases} \end{cases} \]

<table>
<thead>
<tr>
<th>+</th>
<th>‑</th>
<th>+</th>
<th>0</th>
<th>‑0</th>
<th>+0</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>‑</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‑0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>×</th>
<th>‑</th>
<th>+</th>
<th>0</th>
<th>‑0</th>
<th>+0</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>‑</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‑0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example: Sign abstract domain

The sign abstraction function `\( \text{const}(n) \)` is defined as:

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

<table>
<thead>
<tr>
<th>( \text{const}(n) )</th>
<th>( \bot )</th>
<th>( - )</th>
<th>( + )</th>
<th>( 0 )</th>
<th>( -0 )</th>
<th>( +0 )</th>
<th>( \top )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bot )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( - )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( + )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( -0 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( +0 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \top )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table:

<table>
<thead>
<tr>
<th>(-#)</th>
<th>(\bot)</th>
<th>(-)</th>
<th>(+)</th>
<th>(0)</th>
<th>(-0)</th>
<th>(+0)</th>
<th>(\top)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bot)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(+)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(+0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\top)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table:

<table>
<thead>
<tr>
<th>(\times#)</th>
<th>(\bot)</th>
<th>(-)</th>
<th>(+)</th>
<th>(0)</th>
<th>(-0)</th>
<th>(+0)</th>
<th>(\top)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bot)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(+)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(+0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\top)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: sign abstract domain

\[ \text{const}^\#(n) = \begin{cases} 
+ & \text{if } n > 0 \\
0 & \text{if } n = 0 \\
- & \text{if } n < 0 
\end{cases} \]

\[
\begin{array}{cccccccc}
+ & - & + & 0 & -0 & +0 & T \\
\hline
\perp & \perp & \perp & \perp & \perp & \perp & \perp \\
- & \perp & - & T & - & - & T & T \\
+ & \perp & T & + & + & T & + & T \\
0 & \perp & - & + & 0 & -0 & +0 & T \\
-0 & \perp & - & T & -0 & -0 & T & T \\
+0 & \perp & T & + & +0 & T & +0 & T \\
T & \perp & T & T & T & T & T & T \\
\end{array}
\]
**Example : sign abstract domain**

\[
\text{const}^\#(n) = \begin{cases} 
+ & \text{if } n > 0 \\
0 & \text{if } n = 0 \\
- & \text{if } n < 0 
\end{cases}
\]

<table>
<thead>
<tr>
<th>(\text{const}^#(n))</th>
<th>(\text{+})</th>
<th>(-)</th>
<th>(0)</th>
<th>(-0)</th>
<th>(+0)</th>
<th>(T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(-)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(T)</td>
<td>(-)</td>
<td>(T)</td>
<td>(T)</td>
</tr>
<tr>
<td>(+)</td>
<td>(\bot)</td>
<td>(+)</td>
<td>(T)</td>
<td>(+)</td>
<td>(T)</td>
<td>(T)</td>
</tr>
<tr>
<td>(0)</td>
<td>(+)</td>
<td>(-)</td>
<td>(0)</td>
<td>(+0)</td>
<td>(-0)</td>
<td>(T)</td>
</tr>
<tr>
<td>(-0)</td>
<td>(\bot)</td>
<td>(T)</td>
<td>(-0)</td>
<td>(T)</td>
<td>(-0)</td>
<td>(T)</td>
</tr>
<tr>
<td>(+0)</td>
<td>(\bot)</td>
<td>(+)</td>
<td>(+0)</td>
<td>(T)</td>
<td>(+0)</td>
<td>(T)</td>
</tr>
<tr>
<td>(T)</td>
<td>(\bot)</td>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(-#)</th>
<th>(\bot)</th>
<th>(-)</th>
<th>(\bot)</th>
<th>(+)</th>
<th>(0)</th>
<th>(-0)</th>
<th>(+0)</th>
<th>(T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(-)</td>
<td>(\bot)</td>
<td>(T)</td>
<td>(-)</td>
<td>(T)</td>
<td>(-)</td>
<td>(T)</td>
<td>(-)</td>
<td>(T)</td>
</tr>
<tr>
<td>(+)</td>
<td>(\bot)</td>
<td>(+)</td>
<td>(T)</td>
<td>(+)</td>
<td>(T)</td>
<td>(+)</td>
<td>(T)</td>
<td>(+)</td>
</tr>
<tr>
<td>(0)</td>
<td>(+)</td>
<td>(-)</td>
<td>(0)</td>
<td>(+0)</td>
<td>(-0)</td>
<td>(T)</td>
<td>(-0)</td>
<td>(T)</td>
</tr>
<tr>
<td>(-0)</td>
<td>(\bot)</td>
<td>(T)</td>
<td>(-0)</td>
<td>(T)</td>
<td>(-0)</td>
<td>(T)</td>
<td>(-0)</td>
<td>(T)</td>
</tr>
<tr>
<td>(+0)</td>
<td>(\bot)</td>
<td>(+)</td>
<td>(+0)</td>
<td>(+0)</td>
<td>(+0)</td>
<td>(T)</td>
<td>(+0)</td>
<td>(T)</td>
</tr>
<tr>
<td>(T)</td>
<td>(\bot)</td>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\times#)</th>
<th>(\bot)</th>
<th>(-)</th>
<th>(\bot)</th>
<th>(+)</th>
<th>(0)</th>
<th>(-0)</th>
<th>(+0)</th>
<th>(T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(-)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(+)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(0)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(-0)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(+0)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>(T)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
</tbody>
</table>
Example: sign abstract domain

\[ \text{const}^\#(n) = \begin{cases} 
+ & \text{if } n > 0 \\
0 & \text{if } n = 0 \\
- & \text{if } n < 0 
\end{cases} \]

<table>
<thead>
<tr>
<th>-#</th>
<th>⊥</th>
<th>-</th>
<th>+</th>
<th>0</th>
<th>-0</th>
<th>+0</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>-</td>
<td>⊥</td>
<td>T</td>
<td>-</td>
<td>-</td>
<td>T</td>
<td>-</td>
<td>T</td>
</tr>
<tr>
<td>+</td>
<td>⊥</td>
<td>+</td>
<td>T</td>
<td>+</td>
<td>+</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>⊥</td>
<td>+</td>
<td>-</td>
<td>0</td>
<td>+0</td>
<td>-0</td>
<td>T</td>
</tr>
<tr>
<td>-0</td>
<td>⊥</td>
<td>T</td>
<td>-</td>
<td>-0</td>
<td>T</td>
<td>-0</td>
<td>T</td>
</tr>
<tr>
<td>+0</td>
<td>⊥</td>
<td>+</td>
<td>T</td>
<td>+0</td>
<td>+0</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>⊥</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>×#</th>
<th>⊥</th>
<th>-</th>
<th>+</th>
<th>0</th>
<th>-0</th>
<th>+0</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>-</td>
<td>⊥</td>
<td>+</td>
<td>+</td>
<td>0</td>
<td>+0</td>
<td>-0</td>
<td>T</td>
</tr>
<tr>
<td>+</td>
<td>⊥</td>
<td>-</td>
<td>+</td>
<td>0</td>
<td>-0</td>
<td>+0</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>⊥</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0</td>
<td>⊥</td>
<td>+0</td>
<td>-0</td>
<td>0</td>
<td>+0</td>
<td>-0</td>
<td>T</td>
</tr>
<tr>
<td>+0</td>
<td>⊥</td>
<td>-0</td>
<td>+0</td>
<td>0</td>
<td>-0</td>
<td>+0</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>⊥</td>
<td>T</td>
<td>T</td>
<td>0</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Construction of $\llbracket t \rrbracket^\#$

More difficult, because ideally such a refinement should be possible...

$$[x \mapsto +; y \mapsto -0] \xrightarrow{\llbracket (0-y)-x>0 \rrbracket^\#} [x \mapsto +; y \mapsto -]$$
Construction of $[[t]]^\#$

\[
[[e_1 \cdot e_2]]^\#(\rho^\#) = \left( [[e_1]]_{expr}^\#(\rho^\#, n_1^\#) \cap_{Env}^\# [[e_2]]_{expr}^\#(\rho^\#, n_2^\#) \right) \\
\text{with } (n_1^\#, n_2^\#) = [[c]]_{\text{comp}}^\#(A[[e_1]]^\#(\rho^\#), A[[e_2]]^\#(\rho^\#))
\]

- $[[c]]_{\text{comp}}^\# \in \text{Num}^\# \times \text{Num}^\# \rightarrow \text{Num}^\# \times \text{Num}^\#$ computes a refinement of two numeric abstract values, knowing that they verify condition $c$

- $[[e]]_{\text{expr}}^\# \in \text{Env}^\# \times \text{Num}^\# \rightarrow \text{Env}^\#$ : $[[e]]_{\text{expr}}^\#(\rho^\#, n^\#)$ computes a refinement of the abstract environment $\rho^\#$, knowing that the expression $e$ evaluates into a value that is approximated by $n^\#$ in this environment.
\[ \llbracket = \rrbracket^# (x^#, y^#) = \]

<table>
<thead>
<tr>
<th>[ \llbracket \not= \rrbracket^# ]</th>
<th>(\perp)</th>
<th>(\neg)</th>
<th>(+)</th>
<th>(0)</th>
<th>(-0)</th>
<th>(+0)</th>
<th>(\top)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\perp)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\neg)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(+)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(+0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\top)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ \llbracket = \rrbracket^\# (x^\#, y^\#) = (x^\# \sqcap^\# y^\#, x^\# \sqcup^\# y^\#) \]

<table>
<thead>
<tr>
<th><img src="https://via.placeholder.com/150" alt="image" /></th>
<th>( \bot )</th>
<th>( - )</th>
<th>( + )</th>
<th>( 0 )</th>
<th>( -0 )</th>
<th>( +0 )</th>
<th>( \top )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bot )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( - )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( + )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( -0 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( +0 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \top )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[\llbracket = \rrbracket_{\#} (x^\#, y^\#) = (x^\# \sqcap_{\#} y^\#, x^\# \sqcap_{\#} y^\#)\]

<table>
<thead>
<tr>
<th>$\llbracket \neq \rrbracket_{#}$</th>
<th>$\bot$</th>
<th>$-$</th>
<th>$+$</th>
<th>$0$</th>
<th>$-0$</th>
<th>$+0$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
</tr>
<tr>
<td>$-$</td>
<td>$(\bot, \bot)$</td>
<td>$(-, -)$</td>
<td>$(-, +)$</td>
<td>$(-, 0)$</td>
<td>$(-, -0)$</td>
<td>$(-, +0)$</td>
<td>$(-, T)$</td>
</tr>
<tr>
<td>$+$</td>
<td>$(\bot, \bot)$</td>
<td>$(+, -)$</td>
<td>$(+, +)$</td>
<td>$(+, 0)$</td>
<td>$(+, -0)$</td>
<td>$(+, +0)$</td>
<td>$(+, T)$</td>
</tr>
<tr>
<td>$0$</td>
<td>$(\bot, \bot)$</td>
<td>$(0, -)$</td>
<td>$(0, +)$</td>
<td>$(\bot, \bot)$</td>
<td>$(0, -)$</td>
<td>$(0, +)$</td>
<td>$(0, T)$</td>
</tr>
<tr>
<td>$-0$</td>
<td>$(\bot, \bot)$</td>
<td>$(-0, -)$</td>
<td>$(-0, +)$</td>
<td>$(-0, 0)$</td>
<td>$(-0, -0)$</td>
<td>$(-0, +0)$</td>
<td>$(-0, T)$</td>
</tr>
<tr>
<td>$+0$</td>
<td>$(\bot, \bot)$</td>
<td>$(+0, -)$</td>
<td>$(+0, +)$</td>
<td>$(+, 0)$</td>
<td>$(+, -0)$</td>
<td>$(+, +0)$</td>
<td>$(+, T)$</td>
</tr>
<tr>
<td>$T$</td>
<td>$(\bot, \bot)$</td>
<td>$(T, -)$</td>
<td>$(T, +)$</td>
<td>$(T, 0)$</td>
<td>$(T, -0)$</td>
<td>$(T, +0)$</td>
<td>$(T, T)$</td>
</tr>
<tr>
<td>$\ll$</td>
<td>$\bot$</td>
<td>$-$</td>
<td>$+$</td>
<td>0</td>
<td>$-0$</td>
<td>$+0$</td>
<td>$\top$</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>$\bot$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$-$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$-0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\top$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\leq$</th>
<th>$\bot$</th>
<th>$-$</th>
<th>$+$</th>
<th>0</th>
<th>$-0$</th>
<th>$+0$</th>
<th>$\top$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$-$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$-0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\top$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ll$</td>
<td>$\bot$</td>
<td>$-$</td>
<td>$+$</td>
<td>$0$</td>
<td>$-0$</td>
<td>$+0$</td>
<td>$\top$</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
</tr>
<tr>
<td>$-$</td>
<td>$\bot,\bot$</td>
<td>$-,-$</td>
<td>$-,+0$</td>
<td>$-0$</td>
<td>$-,-0$</td>
<td>$-,+0$</td>
<td>$-,+\top$</td>
</tr>
<tr>
<td>$+$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$+0,+0$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$+0,+0$</td>
<td>$+0,+\top$</td>
</tr>
<tr>
<td>$0$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$0,0$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$0,0$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$-0$</td>
<td>$\bot,\bot$</td>
<td>$-0,-0$</td>
<td>$-0,0$</td>
<td>$-0,-0$</td>
<td>$-0,0$</td>
<td>$-0,0$</td>
<td>$-0,\top$</td>
</tr>
<tr>
<td>$+0$</td>
<td>$\bot,\bot$</td>
<td>$+0,+0$</td>
<td>$\bot,\bot$</td>
<td>$\bot,\bot$</td>
<td>$+0,+0$</td>
<td>$+0,+0$</td>
<td>$+0,\top$</td>
</tr>
<tr>
<td>$\top$</td>
<td>$\bot,\bot$</td>
<td>$-,-0$</td>
<td>$\top,\bot$</td>
<td>$-0$</td>
<td>$\top,\bot$</td>
<td>$\top,\bot$</td>
<td>$\top,\top$</td>
</tr>
<tr>
<td>$[\llbracket &lt; \rrbracket]$</td>
<td>$\bot$</td>
<td>$-$</td>
<td>$+$</td>
<td>$0$</td>
<td>$-0$</td>
<td>$+0$</td>
<td>$\top$</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
</tr>
<tr>
<td>$-$</td>
<td>$(\bot, \bot)$</td>
<td>$(-, -)$</td>
<td>$(-, +)$</td>
<td>$(-, 0)$</td>
<td>$(-, -0)$</td>
<td>$(-, +0)$</td>
<td>$(-, \top)$</td>
</tr>
<tr>
<td>$+$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(+, +)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(+, +)$</td>
<td>$(+, +)$</td>
</tr>
<tr>
<td>$0$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(0, +)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(0, +)$</td>
<td>$(0, +)$</td>
</tr>
<tr>
<td>$-0$</td>
<td>$(\bot, \bot)$</td>
<td>$(-0, -)$</td>
<td>$(-0, +)$</td>
<td>$(-0, 0)$</td>
<td>$(-0, -0)$</td>
<td>$(-0, +0)$</td>
<td>$(-0, \top)$</td>
</tr>
<tr>
<td>$+0$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(+0, +)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(+0, +0)$</td>
<td>$(+0, +)$</td>
</tr>
<tr>
<td>$\top$</td>
<td>$(\bot, \bot)$</td>
<td>$(-, -)$</td>
<td>$(\top, +)$</td>
<td>$(-, 0)$</td>
<td>$(-, -0)$</td>
<td>$(\top, +0)$</td>
<td>$(\top, \top)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$[\llbracket \leq \rrbracket]$</th>
<th>$\bot$</th>
<th>$-$</th>
<th>$+$</th>
<th>$0$</th>
<th>$-0$</th>
<th>$+0$</th>
<th>$\top$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
</tr>
<tr>
<td>$-$</td>
<td>$(\bot, \bot)$</td>
<td>$(-, -)$</td>
<td>$(-, +)$</td>
<td>$(-, 0)$</td>
<td>$(-, -0)$</td>
<td>$(-, +0)$</td>
<td>$(-, \top)$</td>
</tr>
<tr>
<td>$+$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(+, +)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(+, +)$</td>
<td>$(+, +)$</td>
</tr>
<tr>
<td>$0$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(0, +)$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(0, +0)$</td>
<td>$(0, +0)$</td>
</tr>
<tr>
<td>$-0$</td>
<td>$(\bot, \bot)$</td>
<td>$(-0, -)$</td>
<td>$(-0, +)$</td>
<td>$(-0, 0)$</td>
<td>$(-0, -0)$</td>
<td>$(-0, +0)$</td>
<td>$(-0, \top)$</td>
</tr>
<tr>
<td>$+0$</td>
<td>$(\bot, \bot)$</td>
<td>$(\bot, \bot)$</td>
<td>$(+0, +)$</td>
<td>$(0, 0)$</td>
<td>$(0, 0)$</td>
<td>$(+0, +0)$</td>
<td>$(+0, +)$</td>
</tr>
<tr>
<td>$\top$</td>
<td>$(\bot, \bot)$</td>
<td>$(-, -)$</td>
<td>$(\top, +)$</td>
<td>$(-, 0)$</td>
<td>$(-, -0)$</td>
<td>$(\top, +0)$</td>
<td>$(\top, \top)$</td>
</tr>
</tbody>
</table>
Required operators on the numeric abstraction

\[ \{(n_1, n_2) \mid n_1 \in \gamma_{\text{Num}}(n_1^\#), \ n_2 \in \gamma_{\text{Num}}(n_2^\#), \ n_1 \ast n_2 \} \subseteq \gamma_{\text{Num}}(m_1^\#) \times \gamma_{\text{Num}}(m_2^\#) \]

with \((m_1^\#, m_2^\#) = \llbracket c \rrbracket_{\text{comp}} (n_1^\#, n_2^\#)\)

\[
\llbracket n \rrbracket_{\text{expr}}^\# (\rho^\#, n^\#) = \begin{cases} 
\bot_{\text{Env}} & \text{if } \text{const}^\#(n) \cap^\#_{\text{Num}} n^\# = \bot_{\text{Num}} \\
\rho^\# & \text{otherwise}
\end{cases}
\]

\[
\llbracket x \rrbracket_{\text{expr}}^\# (\rho^\#, n^\#) = (\rho^#[x \mapsto \rho^#(x) \cap^\#_{\text{Num}} n^#])
\]

\[
\llbracket e_1 \circ e_2 \rrbracket_{\text{expr}}^\# (\rho^\#, n^\#) = \left( \llbracket e_1 \rrbracket_{\text{expr}}^\# (\rho^\#, n_1^\#) \cap^\#_{\text{Env}} \llbracket e_2 \rrbracket_{\text{expr}}^\# (\rho^\#, n_2^\#) \right)
\]

with \((n_1^\#, n_2^\#) = \llbracket o \rrbracket_{\text{op}} (n^\#, A[\llbracket e_1 \rrbracket^\# (\rho^#), A[\llbracket e_2 \rrbracket^\# (\rho^#)]\)
Required operators on the numeric abstraction

\[ [o]_{\text{op}} \in \text{Num}^\# \times \text{Num}^\# \times \text{Num}^\# \rightarrow \text{Num}^\# \times \text{Num}^\# \]

\[ [o]_{\text{op}} (n^\#, n_1^\#, n_2^\#) \] computes a refinement of two numeric values \( n_1^\# \) and \( n_2^\# \) knowing that the result of the binary operation \( o \) is approximated by \( n^\# \) on their concretisations.

\[ \forall n^\#, n_1^\#, n_2^\# \in \text{Num}^\#, \]
\[ \{ (n_1, n_2) \mid n_1 \in \gamma_{\text{Num}}(n_1^\#), n_2 \in \gamma_{\text{Num}}(n_2^\#), (n_1 \circ n_2) \in \gamma_{\text{Num}}(n^\#) \} \]
\[ \subseteq \gamma_{\text{Num}}(m_1^\#) \times \gamma_{\text{Num}}(m_2^\#) \]

with \((m_1^\#, m_2^\#) = [o]_{\text{op}} (n^\#, n_1^\#, n_2^\#)\)
<table>
<thead>
<tr>
<th>$\parallel + \parallel$ $(+,\cdot,\cdot)$</th>
<th>$\bot$</th>
<th>$-$</th>
<th>$+$</th>
<th>0</th>
<th>$-0$</th>
<th>$+0$</th>
<th>$\top$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$-$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$-0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\top$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\parallel \times \parallel$ $(0,\cdot,\cdot)$</th>
<th>$\bot$</th>
<th>$-$</th>
<th>$+$</th>
<th>0</th>
<th>$-0$</th>
<th>$+0$</th>
<th>$\top$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$-$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$-0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\top$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[
\begin{array}{llllllll}
\oplus & \neg \quad + & 0 & -0 & +0 & \top \\
\bot & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) \\
\neg & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) \\
+ & (\bot, \bot) & (+, -) & (+, +) & (+, 0) & (+, -0) & (+, +0) & (+, \top) \\
0 & (\bot, \bot) & (\bot, \bot) & (0, +) & (\bot, \bot) & (\bot, \bot) & (0, +) & (0, +) \\
-0 & (\bot, \bot) & (\bot, \bot) & (-0, +) & (\bot, \bot) & (\bot, \bot) & (-0, +) & (-0, +) \\
+0 & (\bot, \bot) & (+, -) & (+0, +) & (+, 0) & (+, -0) & (+0, +0) & (+0, \top) \\
\top & (\bot, \bot) & (+, -) & (\top, +) & (+, 0) & (+, -0) & (\top, +0) & (\top, \top) \\
\end{array}
\]

\[
\begin{array}{llllllll}
\otimes & \neg \quad + & 0 & -0 & +0 & \top \\
\bot & & & & & & \\
\neg & & & & & & \\
+ & & & & & & \\
0 & & & & & & \\
-0 & & & & & & \\
+0 & & & & & & \\
\top & & & & & & \\
\end{array}
\]
\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\llbracket + \rrbracket \downarrow^\# \ (+, \cdot , \cdot) & \bot & - & + & 0 & -0 & +0 & T \\
\hline
\bot & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) \\
- & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) \\
+ & (\bot, \bot) & (+, -) & (+, +) & (+, 0) & (+, -0) & (+, +0) & (+, T) \\
0 & (\bot, \bot) & (\bot, \bot) & (0, +) & (\bot, \bot) & (\bot, \bot) & (0, +) & (0, +) \\
-0 & (\bot, \bot) & (\bot, \bot) & (-0, +) & (\bot, \bot) & (\bot, \bot) & (-0, +) & (-0, +) \\
+0 & (\bot, \bot) & (+, -) & (+0, +) & (+, 0) & (+, -0) & (+0, +0) & (+0, T) \\
T & (\bot, \bot) & (+, -) & (T, +) & (+, 0) & (+, -0) & (T, +0) & (T, T) \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\llbracket \times \rrbracket \downarrow^\# \ (0, \cdot , \cdot) & \bot & - & + & 0 & -0 & +0 & T \\
\hline
\bot & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) \\
- & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (-0) & (-0) & (-0) & (-0) \\
+ & (\bot, \bot) & (\bot, \bot) & (\bot, \bot) & (+, 0) & (+, 0) & (+, 0) & (+, 0) \\
0 & (\bot, \bot) & (0, ) & (0, +) & (0, 0) & (0, -0) & (0, +0) & (0, T) \\
-0 & (\bot, \bot) & (0, -) & (0, +) & (-0, 0) & (-0, -0) & (-0, +0) & (-0, T) \\
+0 & (\bot, \bot) & (0, -) & (0, +) & (+0, 0) & (+0, -0) & (+0, +0) & (+0, T) \\
T & (\bot, \bot) & (0, -) & (0, +) & (T, 0) & (T, 0) & (T, 0) & (T, T) \\
\hline
\end{array}
\]
module type NumAbstraction =
   sig
     module L : Lattice

     val backTest : comp -> L.t -> L.t -> L.t * L.t

     val semOp : op -> L.t -> L.t -> L.t

     val back_semOp : op -> L.t -> L.t -> L.t -> L.t * L.t

     val const : int -> L.t

     val top : L.t

     val to_string : string -> L.t -> string
   end

module EnvNotRelational = functor (AN:NumAbstraction) ->
   (struct ... end : EnvAbstraction)
Outline

1. Introduction
2. Intermediate representation: syntax and semantics
3. Collecting semantics
4. Just put some #...
5. Building a generic abstract interpreter
6. Numeric abstraction by intervals
7. Widening/Narrowing
8. Polyhedral abstract interpretation
9. Readings
Abstraction by intervals

\[
\text{Int} \overset{\text{def}}{=} \{ [a, b] \mid a, b \in \overline{\mathbb{Z}}, a \leq b \} \cup \{ \bot \} \quad \text{with } \overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}
\]

Lattice:

\[
\begin{array}{c}
I \in \text{Int} \\
\bot \sqsubseteq_{\text{Int}} I
\end{array}
\quad
\begin{array}{ccc}
c \leq a & b \leq d & a, b, c, d \in \overline{\mathbb{Z}} \\
[a, b] \sqsubseteq_{\text{Int}} [c, d]
\end{array}
\]

\[
\begin{align*}
I \uplus_{\text{Int}} \bot & \overset{\text{def}}{=} I, \ \forall I \in \text{Int} \\
\bot \uplus_{\text{Int}} I & \overset{\text{def}}{=} I, \ \forall I \in \text{Int} \\
[a, b] \uplus_{\text{Int}} [c, d] & \overset{\text{def}}{=} [\min(a, c), \max(b, d)]
\end{align*}
\]

\[
\begin{align*}
I \sqcap_{\text{Int}} \bot & \overset{\text{def}}{=} \bot, \ \forall I \in \text{Int} \\
\bot \sqcap_{\text{Int}} I & \overset{\text{def}}{=} \bot, \ \forall I \in \text{Int} \\
[a, b] \sqcap_{\text{Int}} [c, d] & \overset{\text{def}}{=} \rho_{\text{Int}}([\max(a, c), \min(b, d)])
\end{align*}
\]
with $\rho_{\text{Int}} \in (\overline{\mathbb{Z} \times \mathbb{Z}}) \to \text{Int}$ defined by

$$
\rho_{\text{Int}}(a, b) = \begin{cases} [a, b] & \text{if } a \leq b, \\ \bot & \text{otherwise} \end{cases}
$$

$$
\bot_{\text{Int}} \overset{\text{def}}{=} \bot
$$

$$
\top_{\text{Int}} \overset{\text{def}}{=} [-\infty, +\infty]
$$

$$
\gamma_{\text{Int}}(\bot) \overset{\text{def}}{=} \emptyset
$$

$$
\gamma_{\text{Int}}([a, b]) \overset{\text{def}}{=} \{ z \in \mathbb{Z} \mid a \leq z \text{ and } z \leq b \}
$$
All the other operators are stricts: they return ⊥ if one of their arguments is ⊥.

\[
\begin{align*}
+\# ([a, b], [c, d]) &= [a + c, b + d] \\
-\# ([a, b], [c, d]) &= [a - d, b - c] \\
\times\# ([a, b], [c, d]) &= [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]
\end{align*}
\]

\[
\begin{align*}
\llbracket + \rrbracket_{\text{op}}\downarrow\# ([a, b], [c, d], [e, f]) &= (\rho(\max(c, a - f), \min(d, b - e)), \\
& \quad \rho(\max(e, a - d), \min(f, b - c))) \\
\llbracket - \rrbracket_{\text{op}}\downarrow\# ([a, b], [c, d], [e, f]) &= (\rho(\max(c, a + e), \min(d, b + f)), \\
& \quad \rho(\max(e, c - b), \min(f, d - a))) \\
\llbracket \times \rrbracket_{\text{op}}\downarrow\# ([a, b], [c, d], [e, f]) &= ([c, d], [e, f])
\end{align*}
\]

\[
\begin{align*}
\llbracket = \rrbracket_{\text{comp}}\downarrow\# ([a, b], [c, d]) &= ([a, b] \cap_{\text{Int}} [c, d], [a, b] \cap_{\text{Int}} [c, d]) \\
\llbracket < \rrbracket_{\text{comp}}\downarrow\# ([a, b], [c, d]) &= ([a, b] \cap_{\text{Int}} [-\infty, d - 1], [a + 1, +\infty] \cap_{\text{Int}} [c, d]) \\
\llbracket \leq \rrbracket_{\text{comp}}\downarrow\# ([a, b], [c, d]) &= ([a, b] \cap_{\text{Int}} [-\infty, d], [a, +\infty] \cap_{\text{Int}} [c, d]) \\
\llbracket \neq \rrbracket_{\text{comp}}\downarrow\# ([a, b], [c, d]) &= ? \text{ exercise...} \\
\text{const}(n)^\# &= [n, n]
\end{align*}
\]
Convergence problem

Such a lattice does not satisfy the ascending chain condition.

Example of infinite increasing chain:

$$\bot \sqsubseteq [0,0] \sqsubseteq [0,1] \sqsubseteq \cdots \sqsubseteq [0,n] \sqsubseteq \cdots$$

Solution: dynamic approximation

- we extrapolate the limit thanks to a **widening operator** $\triangledown$

$$\bot \sqsubseteq [0,0] \sqsubseteq [0,1] \sqsubseteq [0,2] \sqsubseteq [0,\infty] = [0,2] \triangledown [0,3]$$
Outline

1. Introduction
2. Intermediate representation: syntax and semantics
3. Collecting semantics
4. Just put some ...   
5. Building a generic abstract interpreter
6. Numeric abstraction by intervals
7. Widening/Narrowing
8. Polyhedral abstract interpretation
9. Readings
Fixpoint approximation

**Lemma**

Let $(A, \sqsubseteq, \sqcup, \sqcap)$ a complete lattice and $f$ a monotone operator on $A$. If $a$ is a post-fixpoint of $f$ (i.e. $f(a) \sqsubseteq a$), then $\text{lfp}(f) \sqsubseteq a$.

We may want to compute an over-approximation of $\text{lfp}(f)$ in the following cases:

- The lattice does not satisfy the ascending chain condition, the iteration $\bot, f(\bot), \ldots, f^n(\bot), \ldots$ may never terminate.

- The ascending chain condition is satisfied but the iteration chain is too long to allow an efficient computation.

- If the underlying lattice is not complete, the limits of the ascending iterations do not necessarily belong to the abstraction domain.
Widening

Idea: the standard iteration is of the form

\[ x^0 = \bot, x^{n+1} = F(x^n) = x^n \sqcap F(x^n) \]

We will replace it by something of the form

\[ y^0 = \bot, y^{n+1} = y^n \triangledown F(y^n) \]

such that

(i) \((y^n)\) is increasing,
(ii) \(x^n \sqsubseteq y^n\), for all \(n\),
(iii) and \((y^n)\) stabilizes after a finite number of steps.

But we also want a \(\triangledown\) operator that is independent of \(F\).
A **widening** is an operator $\nabla : L \times L \to L$ such that

- $\forall x, x' \in L, x \sqcup x' \sqsubseteq x \nabla x'$ (implies (i) & (ii))
- If $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$ is an increasing chain, then the increasing chain $y^0 = x^0, y^{n+1} = y^n \nabla x^{n+1}$ stabilizes after a finite number of steps (implies (iii)).

**Usage**: we replace $x^0 = \bot, x^{n+1} = F(x^n)$

by $y^0 = \bot, y^{n+1} = y^n \nabla F(y^n)$
Widening: theorem

Theorem

Let $L$ a complete lattice, $F : L \to L$ a monotone function and $\nabla : L \times L \to L$ a widening operator. The chain $y^0 = \bot, y^{n+1} = y^n \nabla F(y^n)$ stabilizes after a finite number of steps towards a post-fixpoint $y$ of $F$.

Corollary: $\text{lfp}(F) \sqsubseteq y$. 
Scheme

\[ \text{decreasing \ iteration \ with } \Delta \]

\[ \text{increasing \ iteration \ with } \nabla \]

\[ \text{Ifp}(f) \]
Example: widening on intervals

Idea: as soon as a bound is not stable, we extrapolate it by $+\infty$ (or $-\infty$). After such an extrapolation, the bound can't move any more.

Definition:

$$[a, b] \nabla_{\text{Int}} [a', b'] = \begin{cases} 
-\infty & \text{if } a' < a \\
\text{else if } b' > b & \text{then } +\infty \\
a & \text{else}
\end{cases}$$

$$\perp \nabla_{\text{Int}} [a', b'] = [a', b']$$

$$I \nabla_{\text{Int}} \perp = I$$

Examples:

$$[-3, 4] \nabla_{\text{Int}} [-3, 2] = [-3, 4]$$

$$[-3, 4] \nabla_{\text{Int}} [-3, 5] = [-3, +\infty]$$
Example

\[ x := 100; \]

\[
\text{while } 0 < x \{
\]

\[ x := x - 1; \]

\[
\}
\]

\[ X_1 = [100, 100] \sqcup_{\text{Int}} (X_2 - \# [1, 1]) \]

\[ X_2 = [1, +\infty] \cap_{\text{Int}} X_1 \]

\[ X_3 = [-\infty, 0] \cap_{\text{Int}} X_1 \]
Example: without widening

\[ X_1 = [100, 100] \sqcup_{\text{Int}} (X_2 - \# [1, 1]) \]
\[ X_2 = [1, +\infty] \cap_{\text{Int}} X_1 \]
\[ X_3 = [-\infty, 0] \cap_{\text{Int}} X_1 \]

Iteration strategy: \(1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow \cdots\)

\[ X_1^0 = \bot \quad X_1^{n+1} = [100, 100] \sqcup_{\text{Int}} (X_2^n - \# [1, 1]) \]
\[ X_2^0 = \bot \quad X_2^{n+1} = [1, +\infty] \cap_{\text{Int}} X_1^{n+1} \]
\[ X_3^0 = \bot \quad X_3^{n+1} = [-\infty, 0] \cap_{\text{Int}} X_1^{n+1} \]
Example: without widening

\[
X_1 = [100, 100] \sqcup_{\text{Int}} (X_2 - \# [1, 1]) \\
X_2 = [1, +\infty] \cap_{\text{Int}} X_1 \\
X_3 = [-\infty, 0] \cap_{\text{Int}} X_1
\]

Iteration strategy: \( 1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow \cdots \)

\[
X_1^0 = \bot \\
X_1^{n+1} = [100, 100] \sqcup_{\text{Int}} (X_2^n - \# [1, 1])
\]

\[
X_2^0 = \bot \\
X_2^{n+1} = [1, +\infty] \cap_{\text{Int}} X_1^{n+1}
\]

\[
X_3^0 = \bot \\
X_3^{n+1} = [-\infty, 0] \cap_{\text{Int}} X_1^{n+1}
\]

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_1)</td>
<td>(\bot)</td>
<td>[100, 100]</td>
<td>[99, 100]</td>
<td>[98, 100]</td>
<td>[97, 100]</td>
<td>\cdots</td>
<td>[1, 100]</td>
<td>[0, 100]</td>
<td></td>
</tr>
<tr>
<td>(X_2)</td>
<td>(\bot)</td>
<td>[100, 100]</td>
<td>[99, 100]</td>
<td>[98, 100]</td>
<td>[97, 100]</td>
<td>\cdots</td>
<td>[1, 100]</td>
<td>[1, 100]</td>
<td></td>
</tr>
<tr>
<td>(X_3)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\cdots)</td>
<td>(\bot)</td>
<td>[0, 0]</td>
<td></td>
</tr>
</tbody>
</table>
Example: with widening at each nodes of the cfg

\[ X_1 = [100, 100] \sqcup \text{Int} \left( X_2 - \# [1, 1] \right) \]
\[ X_2 = [1, +\infty] \cap \text{Int} X_1 \]
\[ X_3 = [-\infty, 0] \cap \text{Int} X_1 \]

Iteration strategy: \( 1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow \cdots \)

\[ X_1^0 = \bot \quad X_1^{n+1} = X_1^n \sqcup \text{Int} \left( [100, 100] \sqcup \text{Int} \left( X_2^n - \# [1, 1] \right) \right) \]
\[ X_2^0 = \bot \quad X_2^{n+1} = X_2^n \sqcup \text{Int} \left( [1, +\infty] \cap \text{Int} X_1^{n+1} \right) \]
\[ X_3^0 = \bot \quad X_3^{n+1} = X_3^n \sqcup \text{Int} \left( [-\infty, 0] \cap \text{Int} X_1^{n+1} \right) \]

<table>
<thead>
<tr>
<th>( X_1 )</th>
<th>( \bot )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_2 )</td>
<td>( \bot )</td>
</tr>
<tr>
<td>( X_3 )</td>
<td>( \bot )</td>
</tr>
</tbody>
</table>
Example: with widening at each nodes of the cfg

\[
X_1 = [100, 100] \cup_{\text{Int}} (X_2 - \# [1, 1]) \\
X_2 = [1, +\infty] \cap_{\text{Int}} X_1 \\
X_3 = [-\infty, 0] \cap_{\text{Int}} X_1
\]

Iteration strategy: \(1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow \cdots\)

\[
X_1^0 = \bot \\
X_1^{n+1} = X_1^n \triangledown_{\text{Int}} ([100, 100] \cup_{\text{Int}} (X_2^n - \# [1, 1]))
\]

\[
X_2^0 = \bot \\
X_2^{n+1} = X_2^n \triangledown_{\text{Int}} ([1, +\infty] \cap_{\text{Int}} X_1^{n+1})
\]

\[
X_3^0 = \bot \\
X_3^{n+1} = X_3^n \triangledown_{\text{Int}} ([-\infty, 0] \cap_{\text{Int}} X_1^{n+1})
\]

<table>
<thead>
<tr>
<th>(X_1)</th>
<th>(X_2)</th>
<th>(X_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bot)</td>
<td>[100, 100]</td>
<td>[-\infty, 100]</td>
</tr>
<tr>
<td>(\bot)</td>
<td>[100, 100]</td>
<td>[-\infty, 100]</td>
</tr>
<tr>
<td>(\bot)</td>
<td>(\bot)</td>
<td>[-\infty, 0]</td>
</tr>
</tbody>
</table>
Improving fixpoint approximation

Idea : iterating a little more may help...

**Theorem**

Let \((A, \sqsubseteq, \sqcup, \sqcap)\) a complete lattice, \(f\) a monotone operator on \(A\) and \(a\) a post-fixpoint of \(f\). The chain \((x_n)_n\) defined by \[
\begin{align*}
x_0 &= a \\
x_{k+1} &= f(x_k)
\end{align*}
\] admits for limit \(\bigcup \{x_n\}\) the greatest fixpoint of \(f\) lower than \(a\) (written \(\text{gfp}_a(f)\)). In particular, \(\text{lfp}(f) \subseteq \bigcup \{x_n\}\). Each intermediate step is a correct approximation:

\[
\forall k, \; \text{lfp}(f) \subseteq \text{gfp}_a(f) \subseteq x_k \subseteq a
\]
A *narrowing* is an operator $\Delta : L \times L \to L$ such that

- $\forall x, x' \in L, x' \sqsubseteq x \Delta x' \sqsubseteq x$
- If $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$ is a decreasing chain, then the increasing chain $y^0 = x^0, y^{n+1} = y^n \Delta x^{n+1}$ stabilizes after a finite number of steps.
Theorem

If $\Delta$ is a narrowing operator on a poset $(A, \sqsubseteq)$, if $f$ is a monotone operator on $A$ and $a$ is a post-fixpoint of $f$ then the chain $(x_n)_n$ defined by

\[
\begin{align*}
x_0 &= a \\
x_{k+1} &= x_k \Delta f(x_k)
\end{align*}
\]

stabilizes after a finite number of steps on a post-fixpoint of $f$ lower than $a$. 
Narrowing on intervals

\[ [a, b] \Delta_{\text{Int}} [c, d] = [\text{if } a = -\infty \text{ then } c \text{ else } a ; \text{ if } b = +\infty \text{ then } d \text{ else } b] \]

\[
\begin{align*}
I & \Delta_{\text{Int}} \bot = \bot \\
\bot & \Delta_{\text{Int}} I = \bot 
\end{align*}
\]

Intuition: we only improve infinite bounds.

In practice: a few standard iterations already improve a lot the result that has been obtained after widening...

- Assignments by constants and conditional guards make the decreasing iterations efficient: they \textit{filter} the (too big) approximations computed by the widening
Example: with narrowing at each nodes of the cfg

\[ X_1 = [100, 100] \cup_{\text{Int}} (X_2 - \# [1, 1]) \]
\[ X_2 = [1, +\infty] \cap_{\text{Int}} X_1 \]
\[ X_3 = [-\infty, 0] \cap_{\text{Int}} X_1 \]

Iteration strategy: 1 → 2 → 3 → 1 → 2 → · · ·

\[ X^0_1 = [-\infty, 100] \quad X^{n+1}_1 = X^n_1 \Delta_{\text{Int}} ([100, 100] \cup_{\text{Int}} (X^n_2 - \# [1, 1])) \]
\[ X^0_2 = [-\infty, 100] \quad X^{n+1}_2 = X^n_2 \Delta_{\text{Int}} ([1, +\infty] \cap_{\text{Int}} X^{n+1}_1) \]
\[ X^0_3 = [-\infty, 0] \quad X^{n+1}_3 = X^n_3 \Delta_{\text{Int}} ([-\infty, 0] \cap_{\text{Int}} X^{n+1}_1) \]
Example: with narrowing at each nodes of the cfg

\[ X_1 = [100, 100] \cup_{\text{Int}} (X_2 - \# [1, 1]) \]
\[ X_2 = [1, +\infty] \cap_{\text{Int}} X_1 \]
\[ X_3 = [-\infty, 0] \cap_{\text{Int}} X_1 \]

Iteration strategy: 1 → 2 → 3 → 1 → 2 → ⋯

\[ X_1^0 = [-\infty, 100] \quad X_1^{n+1} = X_1^n \Delta_{\text{Int}} \left( [100, 100] \cup_{\text{Int}} \left( X_2^n - \# [1, 1] \right) \right) \]
\[ X_2^0 = [-\infty, 100] \quad X_2^{n+1} = X_2^n \Delta_{\text{Int}} \left( [1, +\infty] \cap_{\text{Int}} X_1^{n+1} \right) \]
\[ X_3^0 = [-\infty, 0] \quad X_3^{n+1} = X_3^n \Delta_{\text{Int}} \left( [-\infty, 0] \cap_{\text{Int}} X_1^{n+1} \right) \]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_1)</td>
<td>([-\infty, 100])</td>
<td>([-\infty, 100])</td>
<td>([0, 100])</td>
</tr>
<tr>
<td>(X_2)</td>
<td>([-\infty, 100])</td>
<td>([1, 100])</td>
<td>([1, 100])</td>
</tr>
<tr>
<td>(X_3)</td>
<td>([-\infty, 0])</td>
<td>([-\infty, 0])</td>
<td>([0, 0])</td>
</tr>
</tbody>
</table>
The particular case of an equation system

Consider a system

\[
\begin{align*}
    x_1 &= f_1(x_1, \ldots, x_n) \\
    \vdots \\
    x_n &= f_n(x_1, \ldots, x_n)
\end{align*}
\]

with \( f_1, \ldots, f_n \) monotones.

Standard iteration:

\[
\begin{align*}
    x_1^{i+1} &= f_1(x_1^i, \ldots, x_n^i) \\
    x_2^{i+1} &= f_2(x_1^i, \ldots, x_n^i) \\
    \vdots \\
    x_n^{i+1} &= f_n(x_1^i, \ldots, x_n^i)
\end{align*}
\]

Standard iteration with widening:

\[
\begin{align*}
    x_1^{i+1} &= x_1^i \bigtriangleup f_1(x_1^i, \ldots, x_n^i) \\
    x_2^{i+1} &= x_2^i \bigtriangleup f_2(x_1^i, \ldots, x_n^i) \\
    \vdots \\
    x_n^{i+1} &= x_n^i \bigtriangleup f_n(x_1^i, \ldots, x_n^i)
\end{align*}
\]
The particular case of an equation system

\[
\begin{align*}
x_1 &= f_1(x_1, \ldots, x_n) \\
    &\vdots \\
x_n &= f_n(x_1, \ldots, x_n)
\end{align*}
\]

It is sufficient (and generally more precise) to use \(\nabla\) for a selection of index \(W\) such that each dependence cycle in the system goes through at least one point in \(W\).

\[
\forall k = 1..n, \ x_k^{i+1} = \begin{cases} 
  x_k^i \nabla f_k(x_k^i, \ldots, x_n^i) & \text{if } k \in W \\
  f_k(x_1^i, \ldots, x_n^i) & \text{otherwise}
\end{cases}
\]

Chaotic iteration: at each step, we use only one equation, without forgetting one for ever.

Contrary, to what happen in a standard dataflow framework (with monotone functions and ascending chain condition), the iteration strategy may affect a lot the precision of the result. See F. Bourdoncle, *Efficient Chaotic Iteration Strategies with Widenings*, 1993.
Direct iteration on program syntax tree

Is is also possible to directly iterate over the program syntax tree.

\[
\llbracket \text{while } t \text{ do } s \rrbracket (m_0^\#) = \llbracket \neg t \rrbracket \left( \text{lfp} \left( \lambda m^\#. m_0^\# \cup \llbracket t \rrbracket \circ \llbracket s \rrbracket (m^\#) \right) \right)
\]

or

\[
\llbracket \text{while } t \text{ do } s \rrbracket (m_0^\#) = \llbracket \neg t \rrbracket \left( \text{lfp}_\downarrow \left( \lambda m^\#. m_0^\# \cup \llbracket t \rrbracket \circ \llbracket s \rrbracket (m^\#) \right) \right)
\]

Outline

1 Introduction
2 Intermediate representation: syntax and semantics
3 Collecting semantics
4 Just put some #...
5 Building a generic abstract interpreter
6 Numeric abstraction by intervals
7 Widening/Narrowing
8 Polyhedral abstract interpretation
9 Readings
Polyhedral abstract interpretation

Automatic discovery of linear restraints among variables of a program. P. Cousot and N. Halbwachs. POPL’78.

Patrick Cousot
Nicolas Halbwachs

Polyhedral analysis seeks to discover invariant linear equality and inequality relationships among the variables of an imperative program.
Convex polyhedra

A convex polyhedron can be defined algebraically as the set of solutions of a system of linear inequalities. Geometrically, it can be defined as a finite intersection of half-spaces.
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $Q^2$.

```plaintext
x = 0; y = 0;

while (x<6) {
    if (?) {
        y = y+2;
    }
    x = x+1;
}
```
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\[
\begin{align*}
x &= 0; \quad y = 0; \\
\{x = 0 \land y = 0\}
\end{align*}
\]

\[
\begin{align*}
\textbf{while} \ (x < 6) \ {\{ \\
\textbf{if} \ (?) \ {\{ \\
\{x = 0 \land y = 0\} \\
y &= y + 2;
\}} \\
\}}
\end{align*}
\]

\[
\begin{align*}
x &= x + 1;
\}
\end{align*}
\]
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\[
x = 0; \quad y = 0;
\{x = 0 \land y = 0\}
\]

\[
\text{while (x<6) \{} \\
\quad \text{if (?)} \{} \\
\quad \quad \{x = 0 \land y = 0\} \\
\quad \quad y = y+2; \\
\quad \quad \{x = 0 \land y = 2\} \\
\quad \}; \\
\{x = 0 \land y = 0\} \uplus \{x = 0 \land y = 2\}
\]

At junction points, we over-approximates union by a convex union.

\[
x = x+1;
\]
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $Q^2$.

\[
\begin{align*}
  x = 0; & \quad y = 0; \\
  \{x = 0 \land y = 0\} \\
\end{align*}
\]

\[
\text{while } (x<6) \{ \\
  \text{if } (?) \{ \\
    \{x = 0 \land y = 0\} \\
    y = y+2; \\
    \{x = 0 \land y = 2\} \\
  \}; \\
  \{x = 0 \land 0 \leq y \leq 2\} \\
\}
\]

At junction points, we over-approximates union by a convex union.

\[
\begin{align*}
  x = x+1; \\
\end{align*}
\]
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\[
x = 0; \ y = 0;
\{x = 0 \land y = 0\}
\]

\[
\text{while} \ (x<6) \ {\{\}
\text{if} \ (? \ {\{\}
\text{if} \ (? \ {\{x = 0 \land y = 0\}
\ y = y+2;
\{x = 0 \land y = 2\}
\};; \{x = 0 \land 0 \leq y \leq 2\}
\} \} \ x = x+1;
\{x = 1 \land 0 \leq y \leq 2\}
\}
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $Q^2$.

\[
\begin{align*}
x &= 0; \ y &= 0; \\
\{x = 0 \land y = 0\} \cup \{x = 1 \land 0 \leq y \leq 2\}
\end{align*}
\]

\[
\text{while (x<6) {}
\]
\[
\text{if (?) {}
\]
\[
\{x = 0 \land y = 0\}
\]
\[
y = y+2;
\]
\[
\{x = 0 \land y = 2\}
\]
\[
} \}
\]
\[
\{x = 0 \land 0 \leq y \leq 2\}
\]
\[
x = x+1;
\]
\[
\{x = 1 \land 0 \leq y \leq 2\}
\]
\[
}
\]
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\begin{align*}
x &= 0; \quad y = 0; \\
&\quad \{ x \leq 1 \land 0 \leq y \leq 2x \}
\end{align*}

\begin{verbatim}
while (x<6) {
    if (?) {
        {x = 0 \land y = 0}
        y = y+2;
        {x = 0 \land y = 2}
    };
    {x = 0 \land 0 \leq y \leq 2}
    x = x+1;
    {x = 1 \land 0 \leq y \leq 2}
}
\end{verbatim}
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\[
x = 0; \ y = 0;
\{x \leq 1 \land 0 \leq y \leq 2x\}
\]

\[
\textbf{while} \ (x<6) \ {\}
\textbf{if} \ (?) \ {\}
\textbf{if} \ (?) \ {\}
\{x \leq 1 \land 0 \leq y \leq 2x\}
\]
\[
y = y+2;
\{x = 0 \land y = 2\}
\];
\{x = 0 \land 0 \leq y \leq 2\}
\]

\[
x = x+1;
\{x = 1 \land 0 \leq y \leq 2\}
\]
\]
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

```plaintext
x = 0; y = 0;
{x ≤ 1 ∧ 0 ≤ y ≤ 2x}

while (x<6) {
    if (?) {
        if (?) {
            {x ≤ 1 ∧ 0 ≤ y ≤ 2x}
            y = y+2;
            {x ≤ 1 ∧ 2 ≤ y ≤ 2x + 2}
        }
        {x = 0 ∧ 0 ≤ y ≤ 2}
    }
    x = x+1;
    {x = 1 ∧ 0 ≤ y ≤ 2}
}
```
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\begin{align*}
x &= 0; \quad y = 0; \\
\{ & x \leq 1 \land 0 \leq y \leq 2x \}
\end{align*}

\begin{verbatim}
while (x<6) {
    if (?) {
        \{ x \leq 1 \land 0 \leq y \leq 2x \}
        y = y+2;
        \{ x \leq 1 \land 2 \leq y \leq 2x + 2 \}
    }
}
\end{verbatim}

\begin{align*}
\{ & x \leq 1 \land 0 \leq y \leq 2x \} \\
\cup \{ & x \leq 1 \land 2 \leq y \leq 2x + 2 \}
\end{align*}
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\[
x = 0; \quad y = 0;
\{x \leq 1 \land 0 \leq y \leq 2x\}
\]

\[
\textbf{while} \ (x < 6) \{ \\
\quad \textbf{if} \ (?) \{ \\
\quad\quad \{x \leq 1 \land 0 \leq y \leq 2x\} \\
\quad\quad y = y + 2; \\
\quad\quad \{x \leq 1 \land 2 \leq y \leq 2x + 2\} \\
\quad\}; \\
\quad \{0 \leq x \leq 1 \land 0 \leq y \leq 2x + 2\} \\
\}
\]

\[
x = x + 1; \\
\{x = 1 \land 0 \leq y \leq 2\}
\]
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\[
x = 0; \ y = 0;
\{x \leq 1 \land 0 \leq y \leq 2x\}
\]

\textbf{while} (x<6) {
  \textbf{if} (?) {
    \textbf{if} (?) {
      \{x \leq 1 \land 0 \leq y \leq 2x\}
      y = y+2;
      \{x \leq 1 \land 2 \leq y \leq 2x + 2\}
    };
    \{0 \leq x \leq 1 \land 0 \leq y \leq 2x + 2\}
  };
  x = x+1;
  \{1 \leq x \leq 2 \land 0 \leq y \leq 2x\}
}\}
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\[
\begin{align*}
x & = 0; \
y & = 0; \\
\{ & x \leq 1 \land 0 \leq y \leq 2x \} \quad \lor \{ x \leq 2 \land 0 \leq y \leq 2x \} \\
\textbf{while} \; (x < 6) \; \{ \\
\; \textbf{if} \; (?) \; \{ \\
\; \{ x \leq 1 \land 0 \leq y \leq 2x \} \\
\; y & = y + 2; \\
\; \{ x \leq 1 \land 2 \leq y \leq 2x + 2 \} \\
\; \} \\
\; \} \\
\textbf{end while} \\
\; x & = x + 1; \\
\; \{ 1 \leq x \leq 2 \land 0 \leq y \leq 2x \} \\
\} 
\end{align*}
\]

At loop headers, we use heuristics (widening) to ensure finite convergence.
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\[
x = 0; \quad y = 0; \\
\{0 \leq y \leq 2x\}
\]

\[
\text{while} \ (x<6) \{ \\
\quad \text{if} \ (?) \{ \\
\quad \quad \{x \leq 1 \land 0 \leq y \leq 2x\} \\
\quad \quad y = y+2; \quad \{x \leq 1 \land 2 \leq y \leq 2x + 2\} \\
\quad \}; \\
\quad \{0 \leq x \leq 1 \land 0 \leq y \leq 2x + 2\}
\]

\[
x = x+1; \\
\quad \{1 \leq x \leq 2 \land 0 \leq y \leq 2x\}
\]

At loop headers, we use heuristics (widening) to ensure finite convergence.
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

\[
x = 0; \ y = 0;
\{0 \leq y \leq 2x\}
\]

\[
\text{while } (x<6) \{
\text{if } (?)
\{0 \leq y \leq 2x \land x \leq 5\}
\]
\[
y = y+2;
\{2 \leq y \leq 2x + 2 \land x \leq 5\}
\}
\{0 \leq y \leq 2x + 2 \land 0 \leq x \leq 5\}
\]
\[
x = x+1;
\{0 \leq y \leq 2x \land 1 \leq x \leq 6\}
\}
\{0 \leq y \leq 2x \land 6 \leq x\}
\]

By propagation we obtain a post-fixpoint
Polyhedral analysis

State properties are over-approximated by convex polyhedra in $\mathbb{Q}^2$.

$$x = 0; \ y = 0;$$

$$\{0 \leq y \leq 2x \land x \leq 6\}$$

By propagation we obtain a post-fixpoint which is enhanced by downward iteration.

$$\textbf{while} \ (x<6) \ {\{ \} \ \textbf{if} \ (?) \ {\{ \} \ y = y+2; \ \{2 \leq y \leq 2x + 2 \land x \leq 5\} \}; \ \{0 \leq y \leq 2x + 2 \land 0 \leq x \leq 5\} \}$$

$$x = x+1;$$

$$\{0 \leq y \leq 2x \land 1 \leq x \leq 6\}$$

$$\} \ \{0 \leq y \leq 2x \land 6 = x\}$$
Polyhedral analysis

A more complex example.

The analysis accepts to replace some constants by parameters.

```plaintext
x = 0; y = A;
{A \leq y \leq 2x + A \land x \leq N}

while (x<N) {
    if (?) {
        if (?) {
            {A \leq y \leq 2x + A \land x \leq N - 1}
            y = y+2;
            {A + 2 \leq y \leq 2x + A + 2 \land x \leq N - 1}
        }
    }
    x = x+1;
    {A \leq y \leq 2x + A \land 1 \leq x \leq N}
}
{A \leq y \leq 2x + A \land N = x}
```
The four polyhedra operations

- $\cup \in \mathbb{P}_n \times \mathbb{P}_n \to \mathbb{P}_n$: convex union
  - over-approximates the concrete union at junction points

- $\cap \in \mathbb{P}_n \times \mathbb{P}_n \to \mathbb{P}_n$: intersection
  - over-approximates the concrete intersection after a conditional instruction

- $[x := e] \in \mathbb{P}_n \to \mathbb{P}_n$: affine transformation
  - over-approximates the assignment of a variable by a linear expression

- $\nabla \in \mathbb{P}_n \times \mathbb{P}_n \to \mathbb{P}_n$: widening
  - ensures (and accelerates) convergence of (post-)fixpoint iteration
  - includes heuristics to infer loop invariants

---

x = 0; y = 0;

$P_0 = [y := 0] [x := 0] (Q^2) \cup P_4$

while (x<6) {
  if (?) {
    $P_1 = P_0 \cap \{x < 6\}$
    y = y+2;
    $P_2 = [y := y + 2] (P_1)$
  }
  $P_3 = P_1 \cup P_2$
  x = x+1;
  $P_4 = [x := x + 1] (P_3)$
}

$P_5 = P_0 \cap \{x \geq 6\}$
Library for manipulating polyhedra

- Parma Polyhedra Library\(^3\) (PPL), NewPolka: complex C/C++ libraries
- They rely on the Double Description Method
  - polyhedra are managed using two representations in parallel
  
  \[ P = \begin{cases} 
  (x, y) \in \mathbb{Q}^2 & | \begin{aligned} 
  x &\geq -1 \\
  x - y &\geq -3 \\
  2x + y &\geq -2 \\
  x + 2y &\geq -4 
  \end{aligned} 
  \end{cases} \]

- by set of inequalities

- by set of generators
  \[ P = \left\{ \lambda_1 s_1 + \lambda_2 s_2 + \lambda_3 s_3 + \mu_1 r_1 + \mu_2 r_2 \in \mathbb{Q}^2 \mid \begin{aligned} 
  \lambda_1, \lambda_2, \lambda_3, \mu_1, \mu_2 &\in \mathbb{R}^+ \\
  \lambda_1 + \lambda_2 + \lambda_3 &= 1 
  \end{aligned} \right\} \]

- operations efficiency strongly depends on the chosen representations, so they keep both

Outline

1. Introduction
2. Intermediate representation: syntax and semantics
3. Collecting semantics
4. Just put some #...
5. Building a generic abstract interpreter
6. Numeric abstraction by intervals
7. Widening/Narrowing
8. Polyhedral abstract interpretation
9. Readings
References (1)

A few articles

- a short formal introduction
  
  \textit{P. Cousot and R. Cousot. Basic Concepts of Abstract Interpretation.} \url{http://www.di.ens.fr/~cousot/COUSOTpapers/WCC04.shtml}

- technical but very complete (the logic programming part is optional):
  
  \textit{P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs.} \url{http://www.di.ens.fr/~cousot/COUSOTpapers/JLP92.shtml}

- application of abstract interpretation theory to verify airbus flight commands
  
  \textit{P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE Analyser.} \url{http://www.di.ens.fr/~cousot/COUSOTpapers/ESOP05.shtml}
References (2)

On the web:

- informal presentation of AI with nice pictures
  http://www.di.ens.fr/~cousot/AI/IntroAbsInt.html

- a short abstract of various works around AI
  http://www.di.ens.fr/~cousot/AI/

- very complete lecture notes
  http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www/
Back to constant-time static analysis
We perform static analysis at (almost) C level

- Based on previous work with a value analyser, Verasco
- We mix Verasco memory tracking with fine-grained tainting
- Main difficulty: alias analysis taking into account pointer arithmetic
Goal: develop and verify in Coq a realistic static analyzer by abstract interpretation

- Language analyzed: the CompCert subset of C
- Nontrivial abstract domains, including relational domains
- Modular architecture inspired from Astrée’s
- To prove the absence of undefined behaviors in C source programs

Slogan:
- if « CompCert $\approx 1/10^{th}$ of GCC but formally verified »,
- likewise « Verasco $\approx1/10^{th}$ of Astrée but formally verified »
Verified Static Analysis
Verified Static Analysis

Logical Framework
(Coq)
Verified Static Analysis

Logical Framework
(Coq)

Language Semantics
(CompCert C)
Verified Static Analysis

Analyzer Implementation (manual)

Language Semantics (CompCert C)

Logical Framework (Coq)
Verified Static Analysis

- Analyzer Implementation (manual)
- Analyzer Spec. (abstract interp. methodology)
- Language Semantics (CompCert C)

Logical Framework (Coq)
Verified Static Analysis

- **Analyzer Implementation (manual)**
- **Analyzer Spec. (abstract interp. methodology)**
- **Language Semantics (CompCert C)**
- **Soundness Proof (mostly interactive)**
- **Logical Framework (Coq)**
Verified Static Analysis

Analyzer Implementation (manual)
Analyzer Spec. (abstract interp. methodology)
Language Semantics (CompCert C)
Soundness Proof (mostly interactive)
Logical Framework (Coq)

extraction

analyzer .exe
Verasco
A Formally-Verified C Static Analyzer


Verasco
Abstract numerical domains
Verasco
Abstract numerical domains

CompCert C → Clight → C#minor → ... → CompCert compiler

Alarms ← Abstract interpreter → control flow

Memory & value domain

Z → int

Convex polyhedra ← Symbolic equalities ← Nonrel → Rel ← Nonrel → Rel

VERIMAG work

Integer congruences

Integer & F.P. intervals
Verasco
Abstract numerical domains

VERIMAG work
Convex polyhedra
Symbolic equalities
Nonrel → Rel
Integer congruences
Nonrel → Rel
Integer & F.P. intervals
Verasco
Abstract numerical domains

CompCert C → Clight → C#minor → ... → CompCert compiler

Alarms ← Abstract interpreter → control flow

Memory & value domain → states

Z → int

conjunctions of linear inequalities \( \sum ai xi \leq c \) [SAS’13]

Convex polyhedra
Symbolic equalities
Nonrel → Rel
Integer congruences
Nonrel → Rel
Integer & F.P. intervals

VERIMAG work
Verasco
Abstract numerical domains

VERIMAG work

symbolic conditional expressions
(improve precision of assume commands)

Convex polyhedra
Symbolic equalities

Z → int
Nonrel → Rel
Integer congruences
Nonrel → Rel
Integer & F.P. intervals

CompCert C
Clight
C#minor
...
CompCert compiler

Alarms
Abstract interpreter
control flow

Memory & value domain
states

7
Verasco
Abstract numerical domains

CompCert C → Clight → C#minor → ... → CompCert compiler

Alarms → Abstract interpreter → control flow

Memory & value domain → states

Z → int → numbers

Convex polyhedra
Symbolic equalities
Nonrel → Rel
Integer congruences
Nonrel → Rel
Integer & F.P. intervals

VERIMAG work

transforms any non-rel. domain into a (reduced) rel. domain
Verasco
Abstract numerical domains

CompCert C → Clight → C#minor → ... → CompCert compiler

Alarms ← Abstract interpreter → control flow

Memory & value domain → states

Z → int

crucial to analyze the safety of memory accesses (memory alignment)

Convex polyhedra
Symbolic equalities

Nonrel → Rel
Nonrel → Rel

Integer congruences
Integer & F.P. intervals

VERIMAG work
Verasco
Abstract numerical domains

CompCert C → Clight → C#minor → ... → CompCert compiler

Alarms ← Abstract interpreter → control flow

Memory & value domain ← Z → int → states

VERIMAG work

Convex polyhedra
Symbolic equalities
Nonrel → Rel
Nonrel → Rel
Integer congruences
Integer & F.P. intervals

requires reasoning on double-precision floating-point numbers (IEEE754)
Verasco
Abstract numerical domains

CompCert C → Clight → C#minor → ... → CompCert compiler

Alarms
Abstract interpreter
control flow

Memory & value domain
states

Z → int

custom reduced product

Convex polyhedra
Symbolic equalities
Nonrel → Rel
Nonrel → Rel

VERIMAG work
Integer congruences
Integer & F.P. intervals
Verasco
Abstract numerical domains

CompCert C → Clight → C#minor → ... → CompCert compiler

control flow

Alarms ← Abstract interpreter

states

Memory & value domain

Z → int

numbers

Convex polyhedra ← Symbolic equalities ← Nonrel → Rel ← Nonrel → Rel

Nonrel → Rel

Integer congruences

Integer & F.P. intervals

VERIMAG work
Verasco
Implementation

34 000 lines of Coq, excluding blanks and comments
• half proof, half code & specs
• plus parts reused from CompCert

Bulk of the development: abstract domains for states and for numbers (involve large case analyses and difficult proofs over integer and floating points arithmetic)

Except for the operations over polyhedra, the algorithms are implemented directly in Coq’s specification language.
Constant-time analysis at source level
Constant-time analysis at source level

We design an *abstract functor*
Constant-time analysis at source level

We design an abstract functor

- takes as input an abstract memory domain

\[
\begin{align*}
[e] & : M^\# \to V^# \\
[x \to e] & : M^# \to M^# \\
[*e_1 \to e_2] & : M^# \to M^# \\
[x \to *e] & : M^# \to M^# \\
\text{assert}(e) & : M^# \to M^# \\
\text{concretize} & : V^# \to \mathcal{P}(\mathbb{L})
\end{align*}
\]
Constant-time analysis at source level

We design an abstract functor
• takes as input an abstract memory domain

\[
\begin{align*}
[e] & : M \rightarrow V \\
[x \rightarrow e] & : M \rightarrow M \\
[*e_1 \rightarrow e_2] & : M \rightarrow M \\
[x \rightarrow *e] & : M \rightarrow M \\
\text{assert}(e) & : M \rightarrow M \\
\text{concretize} & : V \rightarrow \mathcal{P}(L)
\end{align*}
\]

abstract memory abstract value

set of concrete memory locations
Constant-time analysis at source level

We design an abstract functor

- takes as input an abstract memory domain
  - abstract memory
  - abstract value
  - set of concrete memory locations

- returns an abstract domain that taints every memory cells

\[
\begin{align*}
[T[e]]^* & : M^\text{taint} \rightarrow V^\text{taint} \\
[T[x \rightarrow e]]^* & : M^\text{taint} \rightarrow M^\text{taint} \\
[T[*e_1 \rightarrow e_2]]^* & : M^\text{taint} \rightarrow M^\text{taint} \\
[T[x \rightarrow *e]]^* & : M^\text{taint} \rightarrow M^\text{taint} \\
\text{assert(e)}^* & : M^\text{taint} \rightarrow M^\text{taint} \\
\text{concretize}^* & : V^\text{taint} \rightarrow \mathcal{P}(L)
\end{align*}
\]
Constant-time analysis at source level

We design an abstract functor

- takes as input an abstract memory domain
- returns an abstract domain that taints every memory cells

\[
\begin{align*}
[e] & : \mathbb{M} \rightarrow \mathbb{V} \\
[x \rightarrow e] & : \mathbb{M} \rightarrow \mathbb{M} \\
[*e_1 \rightarrow e_2] & : \mathbb{M} \rightarrow \mathbb{M} \\
[x \rightarrow *e] & : \mathbb{M} \rightarrow \mathbb{M} \\
\text{assert}(e) & : \mathbb{M} \rightarrow \mathbb{M} \\
\text{concretize} & : \mathbb{V} \rightarrow \mathcal{P}(\mathcal{L})
\end{align*}
\]

\[
\begin{align*}
\mathcal{T}[e] & : \mathbb{M}_{\text{taint}} \rightarrow \mathbb{V}_{\text{taint}} \\
\mathcal{T}[x \rightarrow e] & : \mathbb{M}_{\text{taint}} \rightarrow \mathbb{M}_{\text{taint}} \\
\mathcal{T}[*e_1 \rightarrow e_2] & : \mathbb{M}_{\text{taint}} \rightarrow \mathbb{M}_{\text{taint}} \\
\mathcal{T}[x \rightarrow *e] & : \mathbb{M}_{\text{taint}} \rightarrow \mathbb{M}_{\text{taint}}
\end{align*}
\]

\(\{\text{MustBeLow, MayBeHigh}\}\)
Constant-time analysis at source level

We design an abstract functor

- takes as input an abstract memory domain
- returns an abstract domain that taints every memory cell

\[
\begin{align*}
\lbrack e \rbrack^\#: & \quad M^\# \rightarrow V^\# \\
\lbrack x \rightarrow e \rbrack^\#: & \quad M^\# \rightarrow M^\# \\
\lbrack \ast e_1 \rightarrow e_2 \rbrack^\#: & \quad M^\# \rightarrow M^\# \\
\lbrack x \rightarrow \ast e \rbrack^\#: & \quad M^\# \rightarrow M^\# \\
\text{assert(e)}^\#: & \quad M^\# \rightarrow M^\# \\
\text{concretize}^\#: & \quad V^\# \rightarrow \mathcal{P}(\mathbb{L})
\end{align*}
\]

Example:

\[
\begin{align*}
\mathcal{T}\lbrack \ast e_1 \rightarrow e_2 \rbrack^\#(m^\#, t^\#) =
& \quad t^\#[l \mapsto \mathcal{T}\lbrack e_2 \rbrack^\#] \\
& \forall l \in \text{concretize}^\# \circ \lbrack e_1 \rbrack^\#(m^\#)
\end{align*}
\]
We report in Table 1 our results on a set of cryptographic algorithms, all executions times reported were obtained on a 3.1GHz Intel i7 with 16GB of RAM. Sizes are reported in terms of numbers of C\# minor statements (i.e., close to C statements), lines of code are measured with cloc and execution times are reported in seconds.

<table>
<thead>
<tr>
<th>Example</th>
<th>Size</th>
<th>Loc</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes</td>
<td>1171</td>
<td>1399</td>
<td>41.39</td>
</tr>
<tr>
<td>curve25519-donna</td>
<td>1210</td>
<td>608</td>
<td>586.20</td>
</tr>
<tr>
<td>des</td>
<td>229</td>
<td>436</td>
<td>2.28</td>
</tr>
<tr>
<td>rlwe_sample</td>
<td>145</td>
<td>1142</td>
<td>30.76</td>
</tr>
<tr>
<td>salsa20</td>
<td>341</td>
<td>652</td>
<td>0.04</td>
</tr>
<tr>
<td>sha3</td>
<td>531</td>
<td>251</td>
<td>57.62</td>
</tr>
<tr>
<td>snow</td>
<td>871</td>
<td>460</td>
<td>3.37</td>
</tr>
<tr>
<td>tea</td>
<td>121</td>
<td>109</td>
<td>3.47</td>
</tr>
<tr>
<td>nacl_chacha20</td>
<td>384</td>
<td>307</td>
<td>0.34</td>
</tr>
<tr>
<td>nacl_sha256</td>
<td>368</td>
<td>287</td>
<td>0.04</td>
</tr>
<tr>
<td>nacl_sha512</td>
<td>437</td>
<td>314</td>
<td>1.02</td>
</tr>
<tr>
<td>mbedtls_sha1</td>
<td>544</td>
<td>354</td>
<td>0.19</td>
</tr>
<tr>
<td>mbedtls_sha256</td>
<td>346</td>
<td>346</td>
<td>0.38</td>
</tr>
<tr>
<td>nbedtls_sha512</td>
<td>310</td>
<td>399</td>
<td>0.26</td>
</tr>
<tr>
<td>mee-cbc</td>
<td>1959</td>
<td>939</td>
<td>933.37</td>
</tr>
</tbody>
</table>

The first block of lines gathers test cases for the implementations of a representative set of cryptographic primitives including TEA [36], an implementation of sampling in a discrete Gaussian distribution by Bos et al. [10] (rlwe_sample) taken from the OpenQum library [30], a implementation of elliptic curve arithmetic operations over Curve25519 [6] by Langley [16] (curve25519-donna), and various primitives such as AES, DES, etc. The second block reports on different implementations from the NaCl library [7]. The third block reports on implementations from the mbedTLS [26] library. Finally, the last result corresponds to an implementation of MAC-then-Encode-then-CBC-Encrypt (MEE-CBC).

All these examples are proven constant time, except for AES and DES. Our prototype rightfully reports memory accesses depending on secrets, so these two programs are not constant time. Similarly to [2], rlwe_sample is only proven constant time, provided that the core random generator is also constant time, thus showing that it is the only possible source of leakage. The last example mee-cbc is a full implementation of the MEE-CBC construction using low-level primitives taken from the NaCl library. Our prototype is able to verify the constant-time property of this example, showing that it scales to large code bases (939 loc).
Experiments at source level (ESORICS’17)

<table>
<thead>
<tr>
<th>Example</th>
<th>Size</th>
<th>Loc</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes</td>
<td>1171</td>
<td>1399</td>
<td>41.39</td>
</tr>
<tr>
<td>curve25519-donna</td>
<td>1210</td>
<td>608</td>
<td>586.20</td>
</tr>
<tr>
<td>des</td>
<td>229</td>
<td>436</td>
<td>2.28</td>
</tr>
<tr>
<td>rlwe_sample</td>
<td>145</td>
<td>1142</td>
<td>30.76</td>
</tr>
<tr>
<td>salsa20</td>
<td>341</td>
<td>652</td>
<td>0.04</td>
</tr>
<tr>
<td>sha3</td>
<td>531</td>
<td>251</td>
<td>57.62</td>
</tr>
<tr>
<td>snow</td>
<td>871</td>
<td>460</td>
<td>3.37</td>
</tr>
<tr>
<td>tea</td>
<td>121</td>
<td>109</td>
<td>3.47</td>
</tr>
<tr>
<td>nacl_chacha20</td>
<td>384</td>
<td>307</td>
<td>0.34</td>
</tr>
<tr>
<td>nacl_sha256</td>
<td>368</td>
<td>287</td>
<td>0.04</td>
</tr>
<tr>
<td>nacl_sha512</td>
<td>437</td>
<td>314</td>
<td>1.02</td>
</tr>
<tr>
<td>mbedtls_sha1</td>
<td>544</td>
<td>354</td>
<td>0.19</td>
</tr>
<tr>
<td>mbedtls_sha256</td>
<td>346</td>
<td>346</td>
<td>0.38</td>
</tr>
<tr>
<td>nbedtls_sha512</td>
<td>310</td>
<td>399</td>
<td>0.26</td>
</tr>
<tr>
<td>mee-cbc</td>
<td>1959</td>
<td>939</td>
<td>933.37</td>
</tr>
</tbody>
</table>
Conclusion
Conclusion

- We can build secure programming abstractions at source level (C-like) using Abstract Interpretation

this lecture focused on Crypto-Constant-Time security property
Conclusion

- We can build secure programming abstractions at source level (C-like) using Abstract Interpretation.

- We make sure the compiler will generate executables that are as secure.

This lecture focused on Crypto-Constant-Time security property.
Conclusion

We can build secure programming abstractions at source level (C-like) using Abstract Interpretation.

We make sure the compiler will generate executables that are as secure.

We reduce as much as possible the TCB (Trusted Computing Base) with formal proofs.

This lecture focused on Crypto-Constant-Time security property.