Abstract Interpretation
EJCP 2007

David Pichardie

David Pichardie

IRISA/INRIA

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

[Cousot&Cousot 75, 76, 77, 79, 80, 81, 82, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 00, 01, 02, 03, 04, 05, 06, 07,. . .][1]

Patrick Cousot

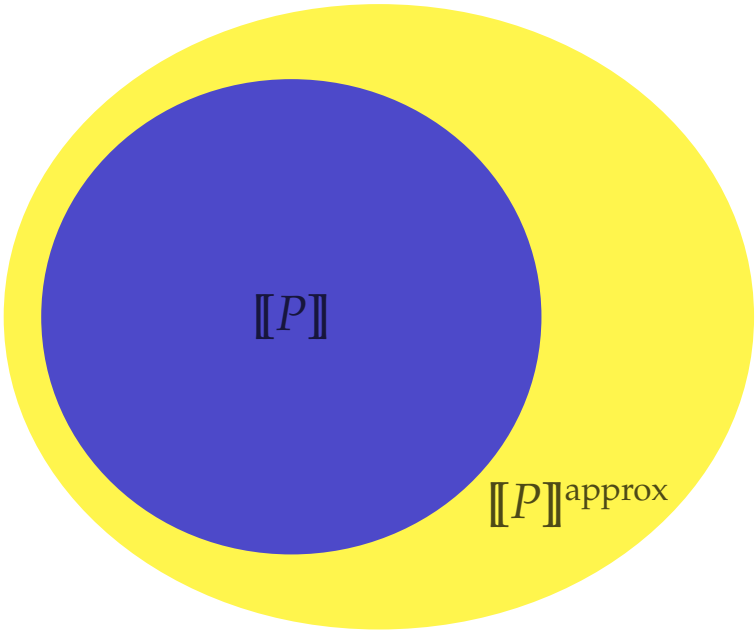
Radhia Cousot

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]

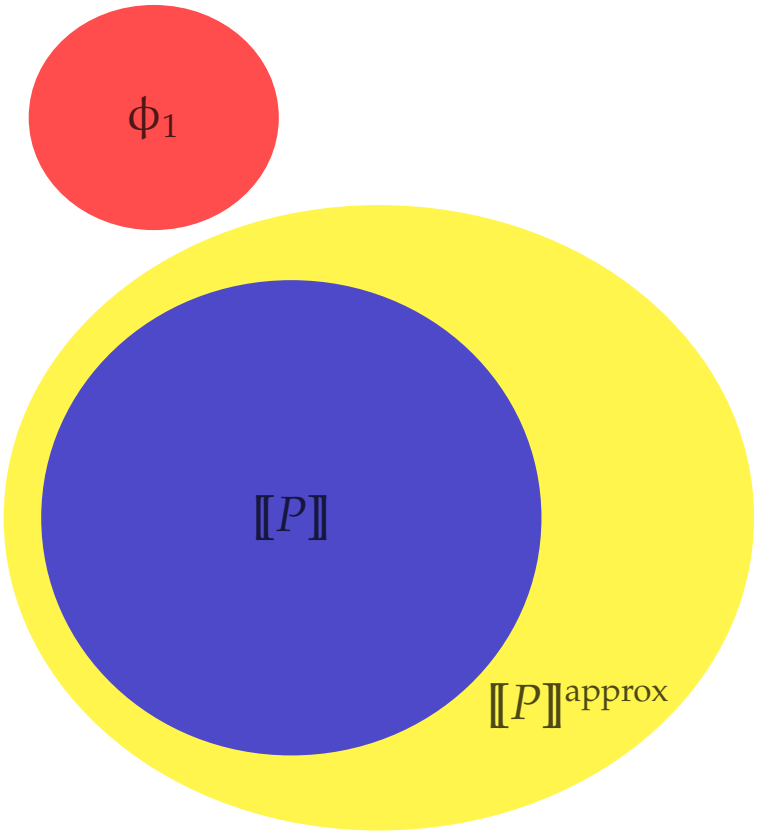


$[\![P]\!]$: concrete semantics (e.g. set of reachable states) (not computable)

$[\![P]\!]^{\text{approx}}$: analyser result (here over-approximation) (computable)

[2] see `http://www.astree.ens.fr/IntroAbsInt.html`

Static analysis computes approximations[2]



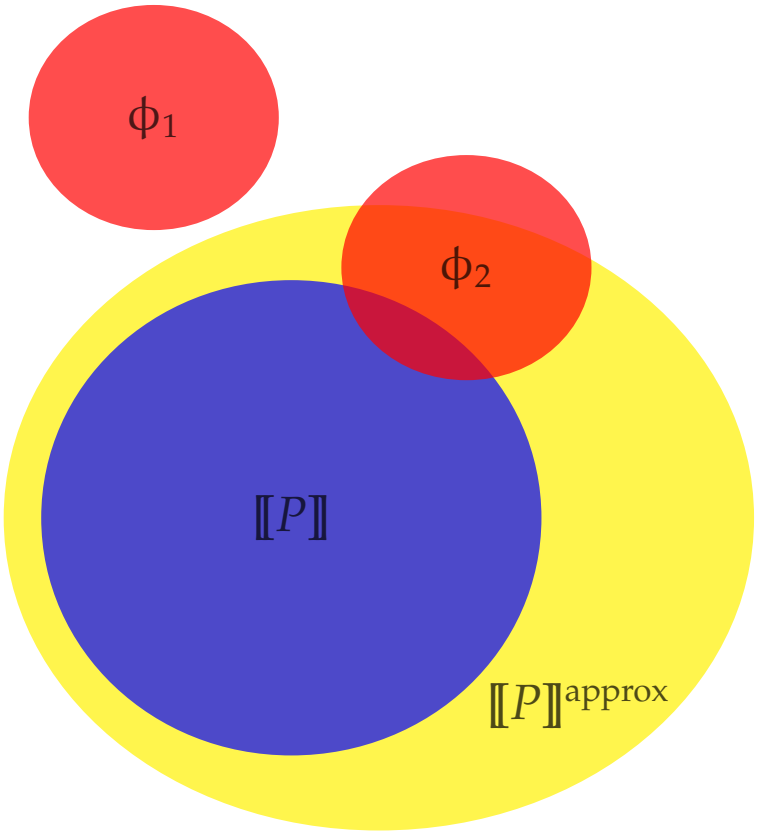
- P is safe w.r.t. ϕ_1 and the analyser proves it

$$[\![P]\!] \cap \phi_1 = \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_1 = \emptyset$$

$[\![P]\!]$:	concrete semantics (e.g. set of reachable states)	(not computable)
ϕ_1 :	erroneous/dangerous set of states	(computable)
$[\![P]\!]^{\text{approx}}$:	analyser result (here over-approximation)	(computable)

[2] see `http://www.astree.ens.fr/IntroAbsInt.html`

Static analysis computes approximations[2]



- P is safe w.r.t. ϕ_1 and the analyser proves it

$$[\![P]\!] \cap \phi_1 = \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_1 = \emptyset$$

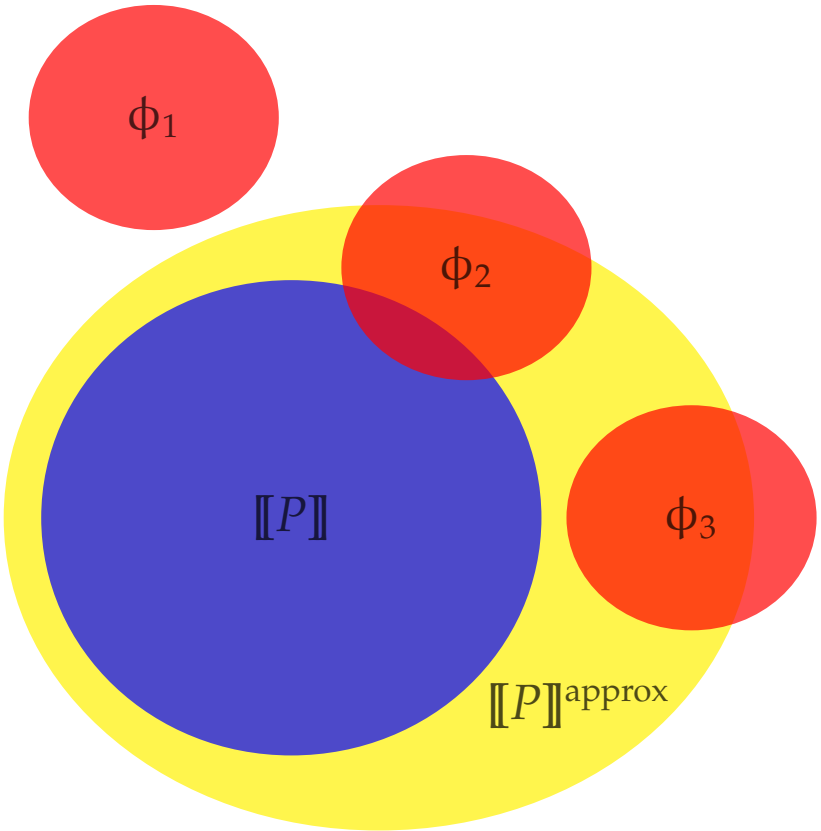
- P is unsafe w.r.t. ϕ_2 and the analyser warns about it

$$[\![P]\!] \cap \phi_2 \neq \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_2 \neq \emptyset$$

$[\![P]\!]$:	concrete semantics (e.g. set of reachable states)	(not computable)
ϕ_1, ϕ_2 :	erroneous/dangerous set of states	(computable)
$[\![P]\!]^{\text{approx}}$:	analyser result (here over-approximation)	(computable)

[2] see `http://www.astree.ens.fr/IntroAbsInt.html`

Static analysis computes approximations[2]



- P is safe w.r.t. ϕ_1 and the analyser proves it

$$[\![P]\!] \cap \phi_1 = \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_1 = \emptyset$$

- P is unsafe w.r.t. ϕ_2 and the analyser warns about it

$$[\![P]\!] \cap \phi_2 \neq \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_2 \neq \emptyset$$

- but P is safe w.r.t. ϕ_3 and the analyser can't prove it (this is called a *false alarm*)

$$[\![P]\!] \cap \phi_3 = \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_3 \neq \emptyset$$

$[\![P]\!]$:	concrete semantics (e.g. set of reachable states)	(not computable)
ϕ_1, ϕ_2, ϕ_3 :	erroneous/dangerous set of states	(computable)
$[\![P]\!]^{\text{approx}}$:	analyser result (here over-approximation)	(computable)

[2] see `http://www.astree.ens.fr/IntroAbsInt.html`

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
x = 0; y = 0;
       {(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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
x = 0; y = 0;
       {(0,0)                }
while (x<6) {
  if (?) {
       {(0,0)                }
    y = y+2;
       {(0,2)                }
  };
       {                     }
  x = x+1;
       {                     }
}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
x = 0; y = 0;
      {(0,0)                          }
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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
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)                }
  };
        {(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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
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 values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

```
x = 0; y = 0;
```
$\{(0,0),(1,0),(1,2),\ldots\}$
```
while (x<6) {
  if (?) {
```
$\{(0,0),(1,0),(1,2),\ldots\}$
```
    y = y+2;
```
$\{(0,2),(1,2),(1,4),\ldots\}$
```
  };
```
$\{(0,0),(0,2),(1,0),(1,2),(1,4),\ldots\}$
```
  x = x+1;
```
$\{(1,0),(1,2),(2,0),(2,2),(2,4),\ldots\}$
```
}
```
$\{(6,0),(6,2),(6,4),(6,6),\ldots\}$

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
```
$\mathtt{x} = 0 \wedge \mathtt{y} = 0$
```
while (x<6) {
  if (?) {

    y = y+2;

  };

  x = x+1;

}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
```
$\mathtt{x} = 0 \wedge \mathtt{y} = 0$
```
while (x<6) {
  if (?) {
```
$\mathtt{x} = 0 \wedge \mathtt{y} = 0$
```
    y = y+2;

  };

  x = x+1;

}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
```
$\mathtt{x} = 0 \wedge \mathtt{y} = 0$
```
while (x<6) {
  if (?) {
```
$\mathtt{x} = 0 \wedge \mathtt{y} = 0$
```
    y = y+2;
```
$\mathtt{x} = 0 \wedge \mathtt{y} > 0$
```
  };

  x = x+1;

}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
```
$\mathtt{x} = 0 \wedge \mathtt{y} = 0$
```
while (x<6) {
  if (?) {
```
$\mathtt{x} = 0 \wedge \mathtt{y} = 0$
```
    y = y+2;
```
$\mathtt{x} = 0 \wedge \mathtt{y} > 0$
```
  };
```
$\mathtt{x} = 0 \wedge \mathtt{y} \geqslant 0$
```
  x = x+1;

}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
```
$\mathtt{x} = 0 \wedge \mathtt{y} = 0$
```
while (x<6) {
  if (?) {
```
$\mathtt{x} = 0 \wedge \mathtt{y} = 0$
```
    y = y+2;
```
$\mathtt{x} = 0 \wedge \mathtt{y} > 0$
```
  };
```
$\mathtt{x} = 0 \wedge \mathtt{y} \geqslant 0$
```
  x = x+1;
```
$\mathtt{x} > 0 \wedge \mathtt{y} \geqslant 0$
```
}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
        x ⩾ 0 ∧ y ⩾ 0
while (x<6) {
  if (?) {
        x = 0 ∧ y = 0
    y = y+2;
        x = 0 ∧ y > 0
  };
        x = 0 ∧ y ⩾ 0
  x = x+1;
        x > 0 ∧ y ⩾ 0
}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
        x ⩾ 0 ∧ y ⩾ 0
while (x<6) {
  if (?) {
        x ⩾ 0 ∧ y ⩾ 0
    y = y+2;
        x = 0 ∧ y > 0
  };
        x = 0 ∧ y ⩾ 0
  x = x+1;
        x > 0 ∧ y ⩾ 0
}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
        x ⩾ 0 ∧ y ⩾ 0
while (x<6) {
  if (?) {
        x ⩾ 0 ∧ y ⩾ 0
    y = y+2;
        x ⩾ 0 ∧ y ⩾ 0
  };
        x = 0 ∧ y ⩾ 0
  x = x+1;
        x > 0 ∧ y ⩾ 0
}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

```
x = 0; y = 0;
        x ⩾ 0 ∧ y ⩾ 0
while (x<6) {
  if (?) {
        x ⩾ 0 ∧ y ⩾ 0
    y = y+2;
        x ⩾ 0 ∧ y ⩾ 0
  };
        x ⩾ 0 ∧ y ⩾ 0
  x = x+1;
        x > 0 ∧ y ⩾ 0
}
```

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

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

A flavor of abstract interpretation

Abstract interpretation executes programs on state properties instead of values.

Collecting semantics

- A state property is a subset in $\mathcal{P}(\mathbb{Z}^2)$ of $(\mathtt{x}, \mathtt{y})$ values.
- When a point is reached for a second time we make an union with the previous property.

Approximation

- The set of manipulated properties may be restricted to ensure computability of the semantics.
  Example : sign of variables
- To stay in the domain of selected properties, we over-approximate the concrete properties.

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

```
assert(T.length=100); i=1;
                                        {i ∈ [1,100]}
while i<T.length {
                                        {i ∈ [1,99]}
    p = T[i]; j = i-1;
                                        {i ∈ [1,99], j ∈ [−1,98]}
    while 0<=j and T[j]>p {
                                        {i ∈ [1,99], j ∈ [0,98]}
        T[j]=T[j+1]; j = j-1;
                                        {i ∈ [1,99], j ∈ [−1,97]}
    };
                                        {i ∈ [1,99], j ∈ [−1,98]}
    T[j+1]=p; i = i+1;
                                        {i ∈ [2,100], j = [−1,98]}
};
                                        {i = 100}
```

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

```
assert(T.length>=1); i=1;
```
$\{1 \leqslant i \leqslant T.length\}$

```
while i<T.length {
```
$\{1 \leqslant i \leqslant T.length - 1\}$

```
    p = T[i]; j = i-1;
```
$\{1 \leqslant i \leqslant T.length - 1 \ \wedge \ -1 \leqslant j \leqslant i - 1\}$

```
    while 0<=j and T[j]>p {
```
$\{1 \leqslant i \leqslant T.length - 1 \ \wedge \ 0 \leqslant j \leqslant i - 1\}$

```
        T[j]=T[j+1]; j = j-1;
```
$\{1 \leqslant i \leqslant T.length - 1 \ \wedge \ -1 \leqslant j \leqslant i - 2\}$

```
    };
```
$\{1 \leqslant i \leqslant T.length - 1 \ \wedge \ -1 \leqslant j \leqslant i - 1\}$

```
    T[j+1]=p; i = i+1;
```
$\{2 \leqslant i \leqslant T.length + 1 \ \wedge \ -1 \leqslant j \leqslant i - 2\}$

```
};
```
$\{i = T.length\}$

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Case study : While

Syntax

$$
\begin{array}{rcll}
Exp & ::= & n & n \in \mathbb{Z} \\
 & | & ? & \\
 & | & x & x \in \mathbb{V} \\
 & | & Exp\ o\ Exp & o \in \{+, -, \times\} \\
 & & & \\
test & ::= & Exp\ c\ Exp & c \in \{<, \leqslant, >, \geqslant, =, \neq\} \\
 & | & \textbf{not}\ test & \\
 & | & test\ \textbf{and}\ test & \\
 & & & \\
Stm & ::= & {}^{l}[x\ :=\ Exp] & l \in \mathbb{P} \\
 & | & {}^{l}[\textbf{skip}] & \\
 & | & \textbf{if}\ {}^{l}[test]\ \{\ Stm\ \}\ \{\ Stm\ \} & \\
 & | & \textbf{while}\ {}^{l}[test]\ \{\ Stm\ \} & \\
 & | & Stm\ ;\ Stm &
\end{array}
$$

Case study : While

Syntax : example

$$
\begin{array}{l}
{}^{0}[x :=?]; \\
\textbf{if } {}^{1}[x < 0] \{ \\
\qquad \textbf{while } {}^{2}[x < 0] \{ \\
\qquad\qquad {}^{3}[x := x + 1]; \\
\qquad \}; \\
\qquad {}^{4}[y := x]; \\
\} \textbf{ else } \{ \\
\qquad {}^{5}[y := 0]; \\
\};
\end{array}
$$

While semantics

Semantic domains

$$\begin{array}{rcl} \text{Env} & \stackrel{\text{def}}{=} & \mathbb{V} \to \mathbb{Z} \\ \textbf{State} & \stackrel{\text{def}}{=} & \mathbb{P} \times \text{Env} \end{array}$$

Semantics of expressions

$$\mathcal{A}[\![e]\!]\rho \in \mathcal{P}(\mathbb{Z}), \quad e \in \mathit{Exp},\ \rho \in \text{Env}$$

$$\begin{array}{lcl} \mathcal{A}[\![n]\!]\rho & = & \{\, n \,\} \\ \mathcal{A}[\![?]\!]\rho & = & \mathbb{Z} \\ \mathcal{A}[\![x]\!]\rho & = & \{\, \rho(x) \,\},\ x \in \mathbb{V} \\ \mathcal{A}[\![e_1\ o\ e_2]\!]\rho & = & \{\, v_1 \bar{o}\, v_2 \mid v_1 \in \mathcal{A}[\![e_1]\!]\rho,\ v_2 \in \mathcal{A}[\![e_2]\!]\rho \,\} \\ & & o \in \{+, -, \times\} \end{array}$$

Remark : $\mathcal{A}[\![\cdot]\!]\rho$ is non-deterministe because of the expression ?.

Semantics of tests

$$\mathcal{B}[\![t]\!]\rho \in \mathcal{P}(\mathbb{B}), \quad t \in \mathit{test},\ \rho \in \mathrm{Env}$$

$$\frac{v_1 \in \mathcal{A}[\![e_1]\!]\rho \quad v_2 \in \mathcal{A}[\![e_2]\!]\rho \quad v_1\ \bar{c}\ v_2}{\mathbf{tt} \in \mathcal{B}[\![e_1\ c\ e_2]\!]\rho}$$

$$\frac{v_1 \in \mathcal{A}[\![e_1]\!]\rho \quad v_2 \in \mathcal{A}[\![e_2]\!]\rho \quad \neg\,(v_1\ \bar{c}\ v_2)}{\mathbf{ff} \in \mathcal{B}[\![e_1\ c\ e_2]\!]\rho}$$

$$\frac{b_1 \in \mathcal{B}[\![t_1]\!]\rho \quad b_2 \in \mathcal{B}[\![t_2]\!]\rho}{b_1 \wedge b_2 \in \mathcal{B}[\![t_1\ \mathbf{and}\ t_2]\!]\rho}$$

Structural Operational Semantics

Small-step semantics

$$(x := a, s) \Rightarrow s[x \mapsto v] \quad \text{if } \exists v \in \mathcal{A}[\![a]\!]s$$

$$(\texttt{skip}, s) \Rightarrow s$$

$$\frac{(S_1, s) \Rightarrow s'}{(S_1 \,;\, S_2, s) \Rightarrow (S_2, s')} \qquad \frac{(S_1, s) \Rightarrow (S_1', s')}{(S_1 \,;\, S_2, s) \Rightarrow (S_1' \,;\, S_2, s')}$$

$$(\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s) \Rightarrow (S_1, s) \quad \text{if } \mathbf{tt} \in \mathcal{B}[\![b]\!]s$$

$$(\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s) \Rightarrow (S_2, s) \quad \text{if } \mathbf{ff} \in \mathcal{B}[\![b]\!]s$$

$$(\texttt{while } b \texttt{ do } S, s) \Rightarrow ((S \,;\, \texttt{while } b \texttt{ do } S), s, tr) \quad \text{if } \mathbf{tt} \in \mathcal{B}[\![b]\!]s$$

$$(\texttt{while } b \texttt{ do } S, s) \Rightarrow s \quad \text{if } \mathbf{ff} \in \mathcal{B}[\![b]\!]s$$

A flowchart representation of program

The standard model of program in static analysis is *control flow graph*.
The graph model used here :

- the nodes are program point $k \in \mathbb{P}$,
- the edges are labeled with *instructions*

$$\begin{array}{rcl} \mathit{Instr} ::= & x := \mathit{Exp} \\ | & \textbf{skip} \\ | & \textbf{assert } \mathit{test} \end{array}$$

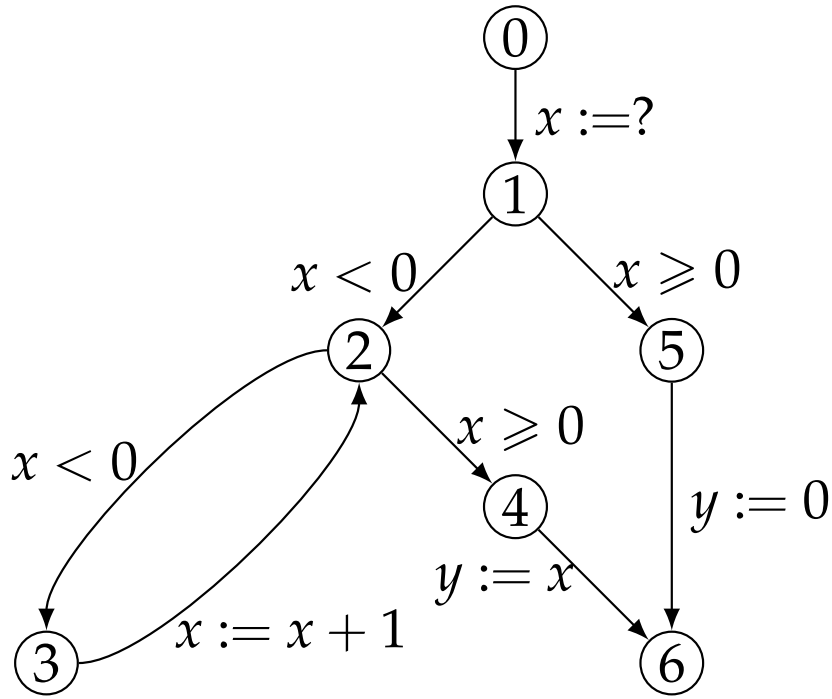
- formally a cfg is a triplet $(k_{\text{init}}, S, k_{\text{end}})$ with
  - $k_{\text{init}} \in \mathbb{P}$: the entry point,
  - $k_{\text{end}} \in \mathbb{P}$: the exit point,
  - $S \subseteq \mathbb{P} \times \mathit{Instr} \times \mathbb{P}$ the set of edges.

Remark : data-flow analyses are generally based on other versions of control flow graph (nodes are put in instructions).

Case study : While

Syntax : example

$^0[x :=?];$
if $^1[x < 0]$ {
  while $^2[x < 0]$ {
    $^3[x := x + 1];$
  };
  $^4[y := x];$
} **else** {
  $^5[y := 0];$
};



assert is left implicit

Small-step semantics of cfg

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

$$s \xrightarrow{x := a} s[x \mapsto v] \qquad \text{if } \exists v \in \mathcal{A}[\![a]\!]s$$

$$s \xrightarrow{\texttt{skip}} s$$

$$s \xrightarrow{\textbf{assert } t} s \qquad \text{if } \textbf{tt} \in \mathcal{B}[\![t]\!]s$$

Then a small-step relation $\rightarrow_p \subseteq \textbf{State} \times \textbf{State}$ for a cfg $p = (k_{\text{init}}, S, k_{\text{end}})$

$$\frac{(k_1, i, k_2) \in S \qquad s_1 \xrightarrow{i} s_2}{(k_1, s_1) \rightarrow_p (k_2, s_2)}$$

The set of initial states

$$\mathcal{S}_0 = \{k_{\text{init}}\} \times \text{Env}$$

Reachable states

$$[\![P]\!] = \{\, s \mid \exists s0 \in \mathcal{S}_0,\ s_0 \rightarrow_p^* s \,\}$$

$[\![P]\!]$ seen by Abstract Interpretation

$$[\![P]\!] \in \mathcal{P}(\textbf{State})$$

Complete lattice : $(\mathcal{P}(\textbf{State}, \subseteq, \bigcup, \bigcap)$

- $Q \in \mathcal{P}(\mathcal{D})$ is a *property*
  - to be an even number : $\{\, 2n \mid n \in \mathbb{Z} \,\}$
- $\subseteq$: logical implication

$$P_1 \text{ is more precise than } P_2 \text{ iff } P_1 \subseteq P_2$$

- $\bigcup$: logical *or*
- $\bigcap$: logical *and*

Plan

Plan

Poset

Définition

Un *ensemble partiellement ordonné (poset)* est un doublet $(A, \sqsubseteq)$ avec A un ensemble, et $\sqsubseteq$ une relation d'ordre partielle, c'est à dire :

$$\begin{array}{ll} \forall x \in A,\ x \sqsubseteq x & \text{(réflexivité)} \\ \forall x, y \in A,\ x \sqsubseteq y \ \wedge\ y \sqsubseteq x \ \Rightarrow\ x = y & \text{(antisymétrie)} \\ \forall x, y, z \in A,\ x \sqsubseteq y \ \wedge\ y \sqsubseteq z \ \Rightarrow\ x \sqsubseteq z & \text{(transitivité)} \end{array}$$

Exemples

- $(\mathbb{N}, \leqslant)$ (ordre total : $\forall x, y,\ x \sqsubseteq y \ \vee\ y \sqsubseteq x$)
- $(\mathbb{N}, \text{« être un diviseur de »})$ noté $(\mathbb{N}, |)$
- $(\mathcal{P}(X), \subseteq)$ avec X un ensemble
- $(A^*, \text{« être un préfixe de »})$ avec A un alphabet

Exercice

Montrer que $(\mathbb{N}, \text{« être un diviseur de »})$ est un poset.

Diagramme de Hasse

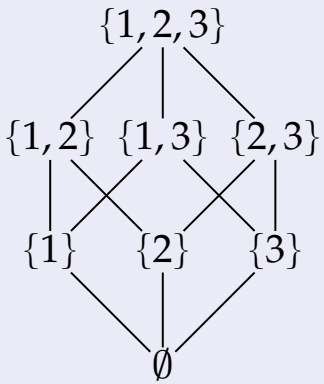
Représentation graphique d'un poset

Définition

Un dessin du plan (ensemble de points et de segments) est *un diagramme de Hasse* d'un poset $(A, \sqsubseteq)$ ssi

- chaque élément de A est associé à un point du plan
- si $x \sqsubseteq y$ avec $x \neq y$ et $\neg\exists z, x \sqsubseteq z \sqsubset y$ alors
  - un segment relie les points p_x et p_y associés respectivement à x et y
  - l'ordonnée de p_x est inférieure à celle de p_y

Exemple



est un diagramme de Hasse du poset $(\mathcal{P}(\{1,2,3\}), \subseteq)$

Exercice

Donner un diagramme de Hasse du poset $(\{1, 2, 3, 4, 6, 8, 12\}, |)$

Treillis

Définition

Un *treillis* est un quadruplet $(A, \sqsubseteq, \sqcup, \sqcap)$ avec

- $(A, \sqsubseteq)$ un poset,
- $\sqcup$ est une borne supérieure binaire :

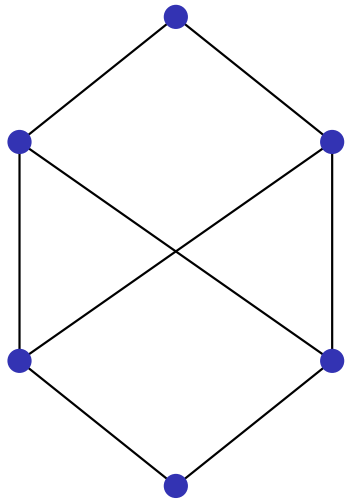
$$\forall x, y \in A,\ x \sqsubseteq x \sqcup y \ \wedge\ y \sqsubseteq x \sqcup y$$
$$\forall x, y, z \in A,\ x \sqsubseteq z \ \wedge\ y \sqsubseteq z \ \Rightarrow\ x \sqcup y \sqsubseteq z$$

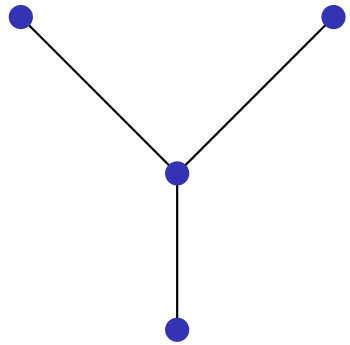
- $\sqcap$ est une borne inférieure binaire :

$$\forall x, y \in A,\ x \sqcap y \sqsubseteq x \ \wedge\ x \sqcap y \sqsubseteq y$$
$$\forall x, y, z \in A,\ z \sqsubseteq x \ \wedge\ z \sqsubseteq y \ \Rightarrow\ z \sqsubseteq x \sqcap y$$

Exercice

Parmi les diagrammes suivants, lesquels représentent des treillis ?





Exercice

Donner les structures de treillis de $(\mathbb{N}, \leqslant)$ et $(\mathbb{N}, |)$.

Treillis complets

Définition

Un treillis complet est un triplet $(A, \sqsubseteq, \bigsqcup)$ avec

- $(A, \sqsubseteq)$ un poset,
- $\bigsqcup$ est une borne supérieure : pour toute partie S de A,
  - $\forall a \in S,\ a \sqsubseteq \bigsqcup S$
  - $\forall b \in A, (\forall a \in S,\ a \sqsubseteq b) \Rightarrow \bigsqcup S \sqsubseteq b$

Un treillis complet possède nécessairement *une borne inférieure* $\bigsqcap$
i.e. : pour toute partie S de A,

- $\forall a \in S,\ \bigsqcap S \sqsubseteq a$
- $\forall b \in A, (\forall a \in S,\ b \sqsubseteq a) \Rightarrow b \sqsubseteq \bigsqcap S$

Il suffit de considérer

$$\bigsqcap S = \bigsqcup \{\, y \mid \forall x \in S, y \sqsubseteq x \,\}$$

Exercice

Montrer qu'un treillis complet admet

- *un plus grand élément* $\top$ $(\forall x, x \sqsubseteq \top)$
- *un plus petit élément* $\bot$ $(\forall x, \bot \sqsubseteq x)$

Points fixes, post-points et pré-point fixes

Définition

Soit $f \in A \to A$ avec $(A, \sqsubseteq)$ un poset, un élément $x \in A$

- est un *point fixe de* f ssi $f(x) = x$
- est un *plus grand point fixe de* f ssi $f(x) = x$ et $\forall y,\ f(y) = y \ \Rightarrow y \sqsubseteq x$
- est un *plus petit point fixe de* f ssi $f(x) = x$ et $\forall y,\ f(y) = y \ \Rightarrow x \sqsubseteq y$
- est un *post-point fixe de* f ssi $f(x) \sqsubseteq x$
- est un *pré-point fixe de* f ssi $x \sqsubseteq f(x)$

Définition

Soit $f \in A \to A$, f est *monotone* ssi

$$\forall x, y \in,\ x \sqsubseteq y \ \Rightarrow\ f(x) \sqsubseteq f(y)$$

Points fixes, post-points et pré-point fixes

Théorème (Knaster-Tarski)

Dans un treillis complet $(A, \sqsubseteq, \bigsqcup)$, *pour toute fonction monotone* $f \in A \to A$,

- *le plus petit point fixe* $\mathrm{lfp}(f)$ *de* f *existe et vaut* $\bigsqcap \{x \in A \mid f(x) \sqsubseteq x\}$,
- *le plus grand point fixe* $\mathrm{gfp}(f)$ *de* f *existe et vaut* $\bigsqcup \{x \in A \mid x \sqsubseteq f(x)\}$,

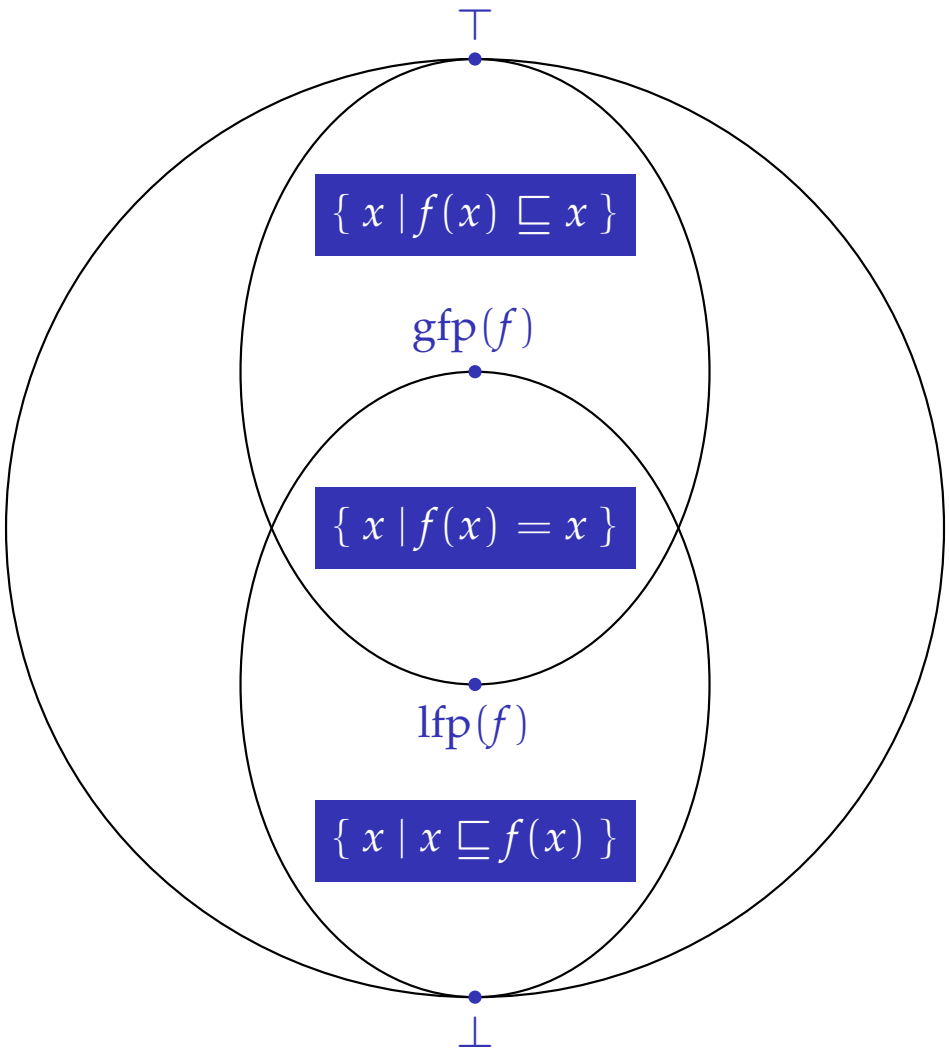
Preuve du théorème de Knaster-Tarski

- Posons $a = \bigsqcap P$ avec $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- Pour tout $x \in P$, on a
  - $a \sqsubseteq x$ — a borne inférieure de P
  - $f(a) \sqsubseteq f(x)$ — f croissante
  - $f(a) \sqsubseteq x$ — déf. P et transitivité

  donc $f(a)$ est un minorant de P
- $f(a) \sqsubseteq a$ — $f(a)$ minorant et a borne inférieure de P
  $\Rightarrow f(f(a)) \sqsubseteq f(a)$ — f croissante
  $\Rightarrow f(a) \in P$ — def. P
  $\Rightarrow a \sqsubseteq f(a)$ — a borne inférieure de P
  $\Rightarrow f(a) = a$ — antisymétrie

  donc $a \in Fix(f)$
- Si $x \in Fix(f)$ alors $x \in P$, donc $a \sqsubseteq x$ car a est la borne inférieure de P.
  Donc $a = \text{lfp}(f)$.
- Preuve de $\text{gfp}(f) = \bigsqcup PreFix(f)$ par dualité

Points fixes, post-points et pré-point fixes



$$\top = \bigsqcup \{ x \mid f(x) \sqsubseteq x \}$$
$$\mathrm{gfp}(f) = \bigsqcup \{ x \mid x \sqsubseteq f(x) \}$$
$$\mathrm{lfp}(f) = \bigsqcap \{ x \mid f(x) \sqsubseteq x \}$$
$$\bot = \bigsqcap \{ x \mid x \sqsubseteq f(x) \}$$

Calcul de point fixe

Théorème

Soit $(A, \sqsubseteq)$ un ensemble partiellement ordonné muni d'un plus petit élément $\bot$. Soit f une fonction monotone. Si la suite $\bot, f(\bot), \ldots, f^n(\bot), \ldots$ stabilise à partir d'un certain rang k (i.e. $f^k(a) = f^{k+1}(a)$, alors $f^k(\bot)$ est le plus petit point fixe de f.

Preuve : Puisque $\bot \sqsubseteq f(\bot)$ et que f est monotone, on peut montrer par récurrence que la suite $\bot, f(\bot), \ldots, f^n(\bot), \ldots$ est croissante.
Soit k tel que $f^k(a) = f^{k+1}(a)$.

- $f^k(a)$ est ainsi un point fixe de f.
- Si x est un point fixe de f, on montre par récurrence que $f^n(\bot) \sqsubseteq x$ pour tout entier n. Cela démontre en particulier que $f^k(a) \sqsubseteq x$.

□

Remarque : $\top, f(\top), \ldots, f^n(\top), \ldots$ permet de calculer le plus grand point fixe de f.

Calcul de point fixe

Définition

Un ensemble partiellement ordonné $(A, \sqsubseteq)$ vérifie la condition de chaîne ascendante si pour toute suite croissante $x_0 \sqsubseteq x_1 \sqsubseteq \cdots \sqsubseteq x_n \sqsubseteq \cdots$ il existe un indice k à partir duquel la suite est stationnaire ($\forall n \geqslant k,\ x_k = x_n$).

Corollaire

Soit $(A, \sqsubseteq)$ un ensemble partiellement ordonné vérifiant la condition de chaîne ascendante et f une fonction monotone. La suite $\bot, f(\bot), \ldots, f^n(\bot), \ldots$ est stationnaire à partir d'un certain rang. Sa limite est le plus petit point fixe de f.

Remarque : Un poset fini vérifie la condition de chaîne ascendante.

Calcul de point fixe

Définition

Soit $(A, \sqsubseteq, \bigsqcup)$ un treillis complet. Une fonction $f \in A \to A$ est continue ssi

$$\forall s \subseteq A,\ \sqcup f(S) = f(\sqcup S)$$

Théorème

Théorème de Kleene Dans un treillis complet $(A, \sqsubseteq, \bigsqcup)$, pour toute fonction continue $f \in A \to A$, le plus petit point fixe $\mathrm{lfp}(f)$ de f est égal à $\bigsqcup \{ f^n(\bot) \mid n \in \mathbb{N} \}$.

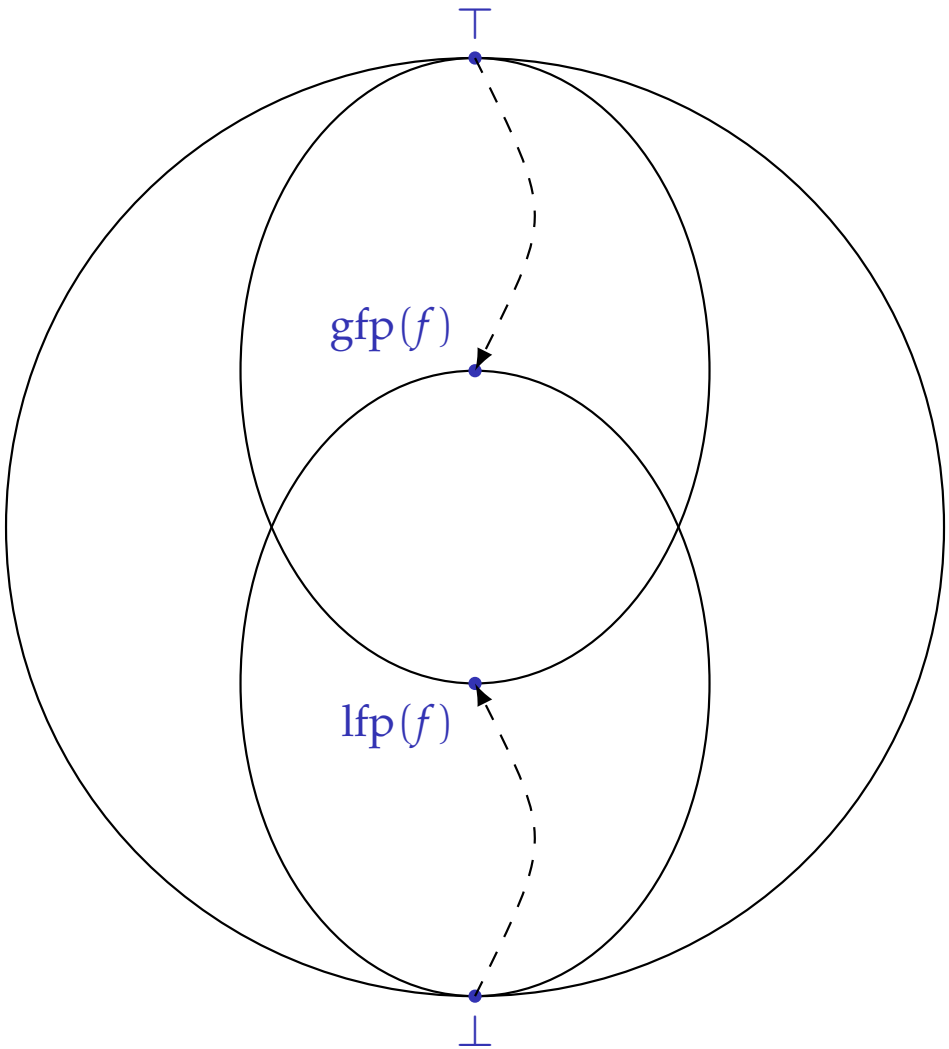
Preuve du théorème de Kleene

- On a vu que $f^n(\bot) \sqsubseteq f^{n+1}(\bot)$
- $\bigsqcup_{n \geqslant 0} f^n(\bot)$ est un point fixe de f :

$$
\begin{aligned}
& f(\bigsqcup_{n \geqslant 0} f^n(\bot)) \\
& = \bigsqcup_{n \geqslant 0} f(f^n(\bot)) && \text{continuité de } f \\
& = f^0(\bot) \sqcup \bigsqcup_{n \geqslant 0} f^{n+1}(\bot) && (\bot \sqcup x = x) \text{ et def. } f^n \text{ et} \\
& = \bigsqcup_{n \geqslant 0} f^n(\bot)
\end{aligned}
$$

- C'est le plus petit point fixe : Soit $x \in Fix(f)$
  - $f^0(\bot) = \bot \sqsubseteq x$
  - $\forall n \geqslant 0 : f^n(\bot) \sqsubseteq x$ induction sur n, car f croissante et $f(x) = x$
  - $\bigsqcup_{n \geqslant 0} f^n(\bot) \sqsubseteq x$ borne supérieure

Calcul de Point fixe



$\top, f(\top), \ldots, f^n(\top), \ldots, \mathrm{gfp}\, f$

$\bot, f(\bot), \ldots, f^n(\bot), \ldots, \mathrm{lfp}\, f$

Système d'inéquations

Les analyses sont parfois spécifiées avec un système d'inéquations :

$$\begin{cases} x_1 & \sqsupseteq & f_1(x_1, \ldots, x_n) \\ & \vdots & \\ x_n & \sqsupseteq & f_n(x_1, \ldots, x_n) \end{cases}$$

Exercice : Soit $(A, \sqsubseteq, \bigsqcup)$ un treillis complet et $f \in A \to A$. Montrer que le plus petit point fixe de f coïncide avec son plus petit post-point fixe.

Conclusion : les deux styles de spécification sont équivalents.

Plan

Sémantique collectrice

Nous allons utiliser une sémantique collectrice, qui définit l'ensemble des valeurs accessibles $[\![P]\!]_k^{\text{col}}$ à chaque point de programme k.

$$\forall k \in \mathbb{P},\ [\![P]\!]_k^{\text{col}} = \{\ \rho \mid (k, \rho) \in [\![P]\!]\ \}$$

$[\![P]\!]^{\text{col}}$ peut être caractérisée comme le plus petit point fixe du système d'équation suivant :

$$\forall k \in \mathbb{P},\ X_k = X_k^{\text{init}} \cup \bigcup_{(k',i,k) \in S} [\![i]\!](X_{k'})$$

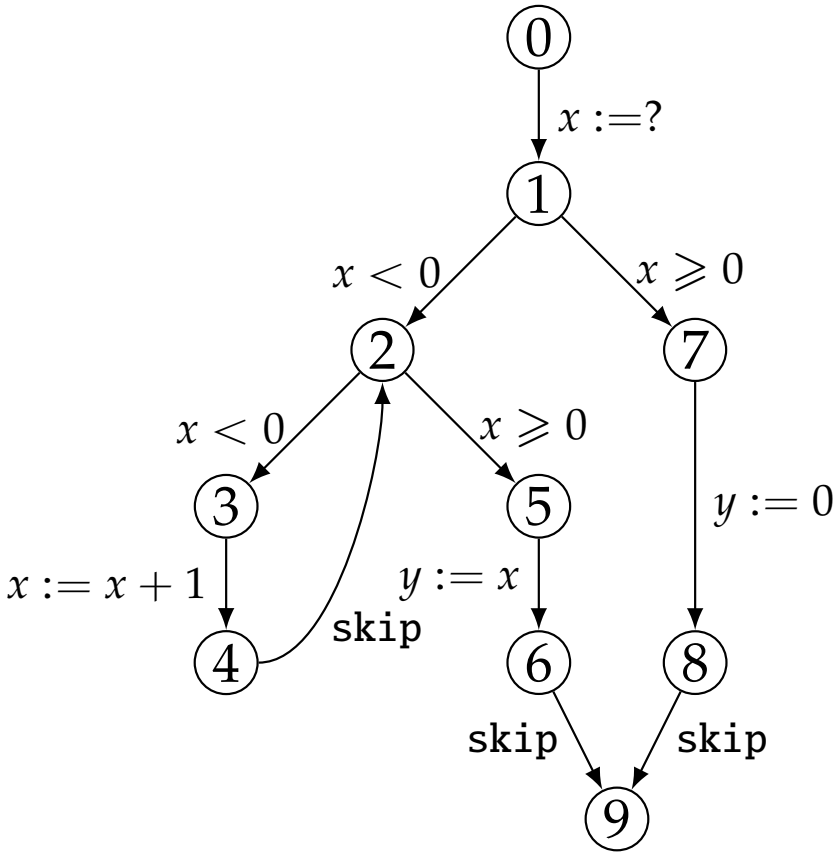
avec $X_k^{\text{init}} = \begin{cases} \text{Env} & \text{si } k = k_{\text{init}} \\ \emptyset & \text{sinon} \end{cases}$

et

$$\forall X \subseteq \text{Env},\ [\![i]\!] = \text{post}\left[\xrightarrow{i}\right](X) = \{\ s_2 \mid \exists s_1 \in X,\ s_1 \xrightarrow{i} s_2\ \}$$

Exercice

Pour le programme suivant, donner le système d'équation caractérisant $[\![P]\!]^{\text{col}}$:



$$
\begin{aligned}
X_0 &= \text{Env} \\
X_1 &= [\![x :=?]\!](X_0) \\
X_2 &= [\![x < 0]\!](X_1) \cup X_4 \\
X_3 &= [\![x < 0]\!](X_2) \\
X_4 &= [\![x := x + 1]\!](X_3) \\
X_5 &= [\![x \geqslant 0]\!](X_2) \\
X_6 &= [\![y := x]\!](X_5) \\
X_7 &= [\![x \geqslant 0]\!](X_1) \\
X_8 &= [\![y := 0]\!](X_7) \\
X_9 &= X_6 \cup X_8
\end{aligned}
$$

Sémantique collectrice et analyse exacte

Les X_k sont donc reliés par un système d'équations de point fixe :
$X_k = F_k(X_1, X_2, \ldots, X_N)$, $k \in \mathbb{P}$ ou encore $X = F(X)$

Analyse exacte : plus petite solution de ce système, ou limite de

$$X_k^0 = \emptyset \ , \ X_k^{n+1} = F_k(X_1^n, X_2^n, \ldots, X_{|K|}^n)$$

d'après le théorème de Kleene (les F_k sont continues)

Problème indécidable : intuitivement,

- Représentation des X_k (qui peuvent être infinis)
- Convergence en un nombre fini d'étapes

Analyse approchée

Analyse exacte :
Plus petite solution de $X = F(X)$ dans le treillis $(\mathcal{P}(\text{Env}), \subseteq, \cup, \cap)$
ou limite de $X^0 = \bot, X^{n+1} = F(X^n)$

Analyse approchée :

- Approximation statique : on remplace le treillis concret $L = \mathcal{P}(\text{Env})$ par un treillis abstrait $L^\sharp$
  - dont on sait représenter (efficacement) les éléments
  - dans lequel on sait calculer ($\sqcup^\sharp, \sqcap^\sharp, a^\sharp, \ldots$)

  et on "transpose" l'équation $X = F(X)$ de L dans $L^\sharp$.
- Approximation dynamique : lorsque $L^\sharp$ ne vérifie pas la condition de chaîne ascendante (ex : intervalles), la convergence du calcul itératif n'est toujours pas garantie (ou parfois trop lente). Dans ce cas, on utilise des méthodes d'extrapolation de la limite.

Analyse Approchée : les questions qui se posent

- Que représente une valeur abstraite (ex : pour les signes, > 0) ?
- Comment modéliser la notion d'approximation ?
- Comment abstraire les opérations concrète $op : L \to L$ dans le treillis abstrait ?

Plan

Qu'est qu'une abstraction ?

La sémantique collectrice manipule des propriétés du programme.

$$[\![P]\!]_k^{\text{col}} \in \mathcal{P}(Env) = \mathcal{A}$$

Abstraire la sémantique d'un programme, c'est restreindre l'ensemble des propriétés utilisés pour exprimer le comportement d'un programme.

$$\overline{\mathcal{A}} \subseteq \mathcal{A}$$

$p \in \mathcal{A}$ est *correctement approchée* par $\overline{p} \in \overline{\mathcal{A}}$ si $p \subseteq \overline{p}$ (sur-approximation)

Exemple : les signes

Pour un programme avec une seule variable

$$\mathcal{A} = \mathcal{P}(\text{Env}) = \mathcal{P}(\mathbb{Z}) \quad \text{et} \quad \overline{\mathcal{A}} = \{ \emptyset, \mathbb{Z}^+, \mathbb{Z}^-, \mathbb{Z} \}$$

La propriété « être plus grand que 1 » (*i.e.* $p_1 = \{ z \in \mathbb{Z} \mid z > 1 \}$) est correctement approchée par $\mathbb{Z}^+$ et $\mathbb{Z}$.

$$p_1 \subseteq \mathbb{Z}^+ \qquad p_1 \subseteq \mathbb{Z}$$

Qu'est qu'une bonne abstraction ? (1)

Cherchons l'outil mathématique qui caractérise la « bonne » abstraction.

Pour toute propriété p,

$$\mathcal{Q}_p = \{\, \overline{p} \in \overline{\mathcal{A}} \mid p \subseteq \overline{p} \}$$

- $\mathcal{Q}_p$ représente l'ensemble des approximations correctes pour p

Bonne abstraction :

- $\forall p, \mathcal{Q}_p \neq \emptyset$: il doit toujours y avoir au moins une approximation correcte
- si $p \subseteq \overline{p}_1$ et $p \subseteq \overline{p}_2$ laquelle choisir ?
  - si $\overline{p}_1$ et $\overline{p}_2$, sont comparables pour $\subseteq$: OK, on prend la plus fine (la plus petite)
  - sinon ? choix non déterministe...

$\overline{\mathcal{A}} \subseteq \mathcal{A}$ est « une bonne abstraction » si pour toute propriété $p \in \mathcal{A}$, $\mathcal{Q}_p = \{\, \overline{p} \in \overline{\mathcal{A}} \mid p \subseteq \overline{p} \}$ admet un plus petit élément.

Qu'est qu'une bonne abstraction ? (2)

Définition

Soit $(A, \sqsubseteq, \sqcup, \sqcap)$ un treillis complet et $Q \subseteq A$ une partie de A, Q est une *famille de Moore* si elle close par plus grande borne inférieure : pour toute partie $S \subseteq Q$, $\bigsqcap S \in Q$.

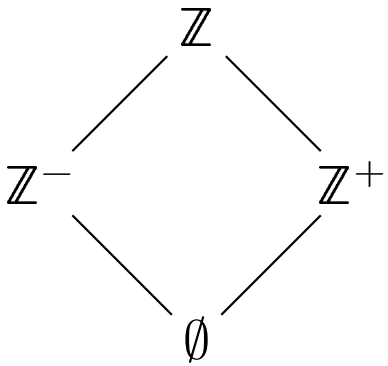
Théorème

Soit $(A, \sqsubseteq, \sqcup, \sqcap)$ un treillis complet et $\overline{A} \subseteq A$. Si pour tout $p \in A$, $\{ \overline{p} \in \overline{A} \mid p \subseteq \overline{p} \}$ admet un plus petit élément alors $\overline{A}$ est une famille de Moore, et réciproquement.

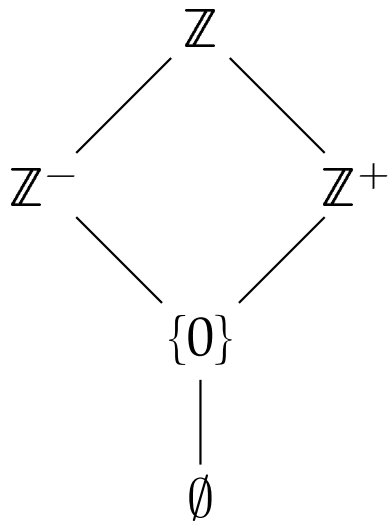
$\overline{\mathcal{A}} \subseteq \mathcal{A}$ est « une bonne abstraction » si c'est une famille de Moore.

Exemple : analyse de signe

$$\mathcal{A} = \mathcal{P}(\mathbb{Z})$$



$$\overline{\mathcal{A}}_1 = \{\emptyset, \mathbb{Z}, \mathbb{Z}^+, \mathbb{Z}^-\}$$



$$\overline{\mathcal{A}}_2 = \{\emptyset, \mathbb{Z}, \mathbb{Z}^+, \mathbb{Z}^-, \{0\}\}$$

Problème : $\{0\}$ n'a pas de meilleure approximation dans $\overline{\mathcal{A}}_1$.

Qu'est qu'une bonne abstraction ? (3)

$$\begin{array}{rccc} \rho : & \mathcal{A} & \longrightarrow & \overline{\mathcal{A}} \\ & P & \longmapsto & \bigsqcap \Omega_P \end{array}$$

Définition

Soit $(A, \sqsubseteq, \sqcup, \sqcap)$ un treillis complet. Une fonction $\rho \in A \longrightarrow A$ est une *fermeture supérieure* si elle vérifie les trois conditions suivantes :

- ρ est monotone ($\forall x, y \in A,\ x \sqsubseteq y \ \Rightarrow\ \rho(x) \sqsubseteq \rho(y)$)
- ρ est extensive ($\forall x \in A,\ x \sqsubseteq \rho(x)$)
- ρ est idempotente ($\forall x \in A,\ \rho(\rho(x)) = \rho(x)$)

Théorème

Soit $(A, \sqsubseteq, \sqcup, \sqcap)$ un treillis complet et $\overline{A} \subset A$. $\overline{\mathcal{A}}$ est une famille de Moore si et seulement si il existe une fermeture supérieure $\rho \in A \longrightarrow A$ telle que $\overline{A} = \rho(A)$.

$\overline{\mathcal{A}} \subset \mathcal{A}$ est « une bonne abstraction » si elle est de la forme $\overline{\mathcal{A}} = \rho(\mathcal{A}) = \{\ \rho(p) \mid p \in \mathcal{A}\ \}$ avec ρ une fermeture supérieure.

Qu'est qu'une bonne abstraction ? (4)

Théorème

Soit $(A, \sqsubseteq, \sqcup, \sqcap)$ *un treillis complet et* $\rho \in A \to A$ *une fermeture supérieure. Le quadruplé* $(\rho(A), \sqsubseteq, \lambda S.\rho(\sqcup S), \sqcap)$ *est un treillis complet.*

Ce treillis n'est pas forcément celui dans lequel on a envie de réaliser des calculs.

Changement de monde ?

- nouveau treillis complet $\left(A^\sharp, \sqsubseteq^\sharp, \sqcup^\sharp, \sqcap^\sharp\right)$
- isomorphisme d'ordre $\iota : A^\sharp \to \rho(A)$
  - $\iota \in A^\sharp \to A$ exprime les propriétés représentés par les éléments de $A^\sharp$ (fonction d'abstraction),
  - $\iota^{-1} \circ \rho \in A \to A^\sharp$ donne la meilleure abstraction d'une propriété, élément de $\mathcal{A}$ fonction de concrétisation,
  - un tel couple $\left(\iota^{-1} \circ \rho \in A \to A^\sharp, \iota \in A^\sharp \to A\right)$ de fonction forme une *insertion de Galois*.

Qu'est qu'une bonne abstraction ? (5)

Définition

Soit $(L_1, \sqsubseteq_1, \sqcup_1, \sqcap_1)$ et $(L_2, \sqsubseteq_2, \sqcup_2, \sqcap_2)$ deux treillis complets. Une paire de fonctions $\alpha \in L_1 \to L_2$ et $\gamma \in L_2 \to L_1$ est une *insertion de Galois* si elle vérifie les conditions suivantes :

- $\forall x_1 \in L_1, \forall x_2 \in L_2,\ \alpha(x_1) \sqsubseteq_1 x_2 \iff x_1 \sqsubseteq_2 \gamma(x_2)$
- γ est injective

Un treillis complet $\left(A^\sharp, \sqsubseteq^\sharp, \sqcup^\sharp, \sqcap^\sharp\right)$ est « une bonne abstraction » si il existe une insertion de Galois entre $(\mathcal{A}, \subseteq, \cup, \cap)$ et $\left(A^\sharp, \sqsubseteq^\sharp, \sqcup^\sharp, \sqcap^\sharp\right)$.

Qu'est qu'une bonne abstraction ? (6)

La nécessité d'avoir une fonction γ injective est parfois trop contraignante : elle oblige a avoir un unique représentant abstrait $p^\sharp \in A^\sharp$ pour chaque propriété abstraite $\overline{p} \in \overline{\mathcal{A}}$. On peut relâcher cette contrainte, en demandant seulement une connexion de Galois.

Définition

Soit $(L_1, \sqsubseteq_1, \sqcup_1, \sqcap_1)$ et $(L_2, \sqsubseteq_2, \sqcup_2, \sqcap_2)$ deux treillis complets. Une paire de fonctions $\alpha \in L_1 \to L_2$ et $\gamma \in L_2 \to L_1$ est une *connexion de Galois* si elle vérifie la condition

$$\forall x_1 \in L_1, \forall x_2 \in L_2,\ \alpha(x_1) \sqsubseteq_1 x_2 \iff x_1 \sqsubseteq_2 \gamma(x_2)$$

Un treillis complet $\left(A^\sharp, \sqsubseteq^\sharp, \sqcup^\sharp, \sqcap^\sharp\right)$ est « une bonne abstraction » si il existe une connexion de Galois entre $(\mathcal{A}, \subseteq, \cup, \cap)$ et $\left(A^\sharp, \sqsubseteq^\sharp, \sqcup^\sharp, \sqcap^\sharp\right)$.

Bilan : qu'est qu'une bonne abstraction ?

1. monde concret : treillis complet, généralement de type $(\mathcal{P}(\mathcal{D}), \subseteq, \bigcup, \bigcap)$
2. monde abstrait : treilli complet $\left(A^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \bigsqcap^\sharp\right)$
3. lien entre les deux : connexion de Galois.

$$\left(\mathcal{P}(\mathcal{D}), \subseteq, \bigcup, \bigcap\right) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \left(A^\sharp, \sqsubseteq^\sharp, \sqcup^\sharp, \sqcap^\sharp\right)$$

« $a^\sharp \in A^\sharp$ est une approximation correcte de $a \in \mathcal{P}(\mathcal{D})$ »

$$\iff \quad \alpha(a) \sqsubseteq^\sharp a^\sharp$$

$$\iff \quad a \subseteq \gamma(a^\sharp)$$

α :fonction d'abstraction $\qquad$ γ :fonction de concrétisation

Remarque : dans la pratique γ suffit pour prouver la correction d'une analyse, mais on perd alors de « beaux » théorèmes...

Plan

Connexion de Galois, propriétés (1)

Si $\alpha : L \to L^\sharp$, $\gamma : L^\sharp \to L$ connexion de Galois

- $\gamma \circ \alpha$ est extensive : $\forall x \in L,\ x \sqsubseteq \gamma(\alpha(x))$:
  l'abstraction est une approximation correcte
- $\alpha \circ \gamma$ est rétractive : $\forall y \in L^\sharp,\ \alpha(\gamma(y) \sqsubseteq^\sharp y$:
  si $L^\sharp$ est « bien concu », $\alpha \circ \gamma = \mathrm{id}$ et (donc) γ est injective (il n'y a pas deux valeurs abstraites qui représentent la même chose)
- α et γ sont monotones (croissantes)
  « plus on a d'information sur l'object concret, plus on en a sur son abstraction, et vice-versa »

Remarques :

- $x \sqsubseteq x'$ signifie que x contient plus d'information que x', ou donne une information plus précise,
- $\top$ signifie : tout est possible.

Connexion de Galois, propriétés (2)

Si $\alpha : L \to L^\sharp, \gamma : L^\sharp \to L$ connexion de Galois

- $\alpha \circ \gamma \circ \alpha = \alpha$ et $\gamma \circ \alpha \circ \gamma = \gamma$
- chaque fonction définit l'autre

$$\alpha(x) = \bigsqcap^\sharp \{y \in L^\sharp | x \sqsubseteq \gamma(y)\}$$

l'abstraction d'un élément concret x est égal au plus petit élément abstrait y qui le surapproxime

$$\gamma(y) = \bigsqcup \{x \in L | \alpha(x) \sqsubseteq^\sharp y\}$$

la concrétisation d'un élément abstrait y est égal au plus grand élément concret x qui le sous-approxime

- α préserve les bornes supérieures, γ les bornes inférieures

$$\alpha(\bigsqcup_{x \in X} x) = \bigsqcup^\sharp_{x \in X} \alpha(x) \qquad \gamma(\bigsqcap^\sharp_{y \in Y} y) = \bigsqcap_{y \in Y} \gamma(y)$$

Approximation des fonctions (1)

Lorsque certains calculs dans le monde concret sont indécidables ou trop coûteux, le monde abstrait peut être utilisé pour effectuer une version simplifiée de ces calculs.

- le résultat de ces calculs devra néanmoins toujours donner une réponse conservatrice vis à vis du calcul concret

Soit $f \in \mathcal{A} \to \mathcal{A}$ dans le monde concret et $f^\sharp \in \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ qui approche correctement chaque calcul concret.

$$\begin{array}{ccc} \mathcal{A} & \xrightarrow{f} & \mathcal{A} \\ \downarrow & & \downarrow \\ \mathcal{A}^\sharp & \xrightarrow{f^\sharp} & \mathcal{A}^\sharp \end{array}$$

Approximation des fonctions (1)

Si $\alpha : L \to L^\sharp, \gamma : L^\sharp \to L$ connexion de Galois

Définition

Une fonction $f^\sharp \in \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ est appelée *approximation correcte* de $f \in \mathcal{A} \to \mathcal{A}$ si

$$\forall a \in \mathcal{A}, a^\sharp \in \mathcal{A}^\sharp, \alpha(a) \sqsubseteq^\sharp a^\sharp \Rightarrow \alpha(f(a)) \sqsubseteq^\sharp f^\sharp(a^\sharp)$$

Pour les fonctions abstraites monotones, on peut énoncer plusieurs critères de correction équivalents.

Théorème

Une fonction $f^\sharp \in \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ monotone et une fonction $f \in \mathcal{A} \to \mathcal{A}$, les quatre assertions suivantes sont équivalentes :

(i) *$f^\sharp$ est une approximation correcte de f,*

(ii) $\alpha \circ f \mathrel{\dot{\sqsubseteq}^\sharp} f^\sharp \circ \alpha$

(ii) $\alpha \circ f \circ \gamma \mathrel{\dot{\sqsubseteq}^\sharp} f^\sharp$

(iv) $f \circ \gamma \mathrel{\dot{\sqsubseteq}^\sharp} \gamma \circ f^\sharp$

Transfert de point fixe

Théorème

Étant donné une connexion de Galois $(\mathcal{A}, \sqsubseteq, \bigsqcup, \bigsqcap) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \left(\mathcal{A}^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \bigsqcap^\sharp\right)$*, une fonction* $f^\sharp \in \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ *monotone et une fonction* $f \in \mathcal{A} \to \mathcal{A}$ *monotone qui vérifient* $\alpha \circ f = f^\sharp \circ \alpha$*. On a*

$$\alpha(\mathrm{lfp}(f)) = \mathrm{lfp}(f^\sharp)$$

Théorème

Étant donné une connexion de Galois $(\mathcal{A}, \sqsubseteq, \bigsqcup, \bigsqcap) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \left(\mathcal{A}^\sharp, \sqsubseteq^\sharp, \bigsqcup^\sharp, \bigsqcap^\sharp\right)$*, une fonction* $f^\sharp \in \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ *monotone et une fonction* $f \in \mathcal{A} \to \mathcal{A}$ *monotone qui vérifient* $\alpha \circ f \sqsubseteq f^\sharp \circ \alpha$*. On a*

$$\alpha(\mathrm{lfp}(f)) \sqsubseteq \mathrm{lfp}(f^\sharp)$$

Remarques

$$\alpha \circ f \mathrel{\dot{\sqsubseteq}^\sharp} f^\sharp \circ \alpha \iff \alpha \circ f \circ \gamma \mathrel{\dot{\sqsubseteq}^\sharp} f^\sharp \iff f \circ \gamma \mathrel{\dot{\sqsubseteq}^\sharp} \gamma \circ f^\sharp$$

donc $\alpha \circ f \circ \gamma$ est la meilleur fonction abstraite $f^\sharp$ vérifiant $\alpha(\mathrm{lfp}(f)) \sqsubseteq \mathrm{lfp}(f^\sharp)$.

$\alpha \circ f \circ \gamma$ ne doit cependant pas être confondu avec une implémentation de $f^\sharp$.

En pratique,

- $\alpha \circ f \circ \gamma$ représente seulement la spécification de la meilleure abstraction,
- et non un algorithme
  - la fonction α est rarement calculable

Plan

Plan