Principles of Abstract Interpretation

Program Analysis

A technique to check if a program satisfies a semantic property

Useful for optimisation and verification

What to Analyse:

Target Programs

- Domain-specific vs Non-domain-specific analyses
- Program-level vs Model-level analyses

Target Properties

- Safety properties: some behavior observable in finite time will never occur.
- Liveness properties: some behavior observable after infinite time will never occur.
- Information flow properties

When to Analyse:

Dynamic vs Static techniques

What to Analyse: Safety Properties

Some behaviors observable in finite time will never occur.

Examples:

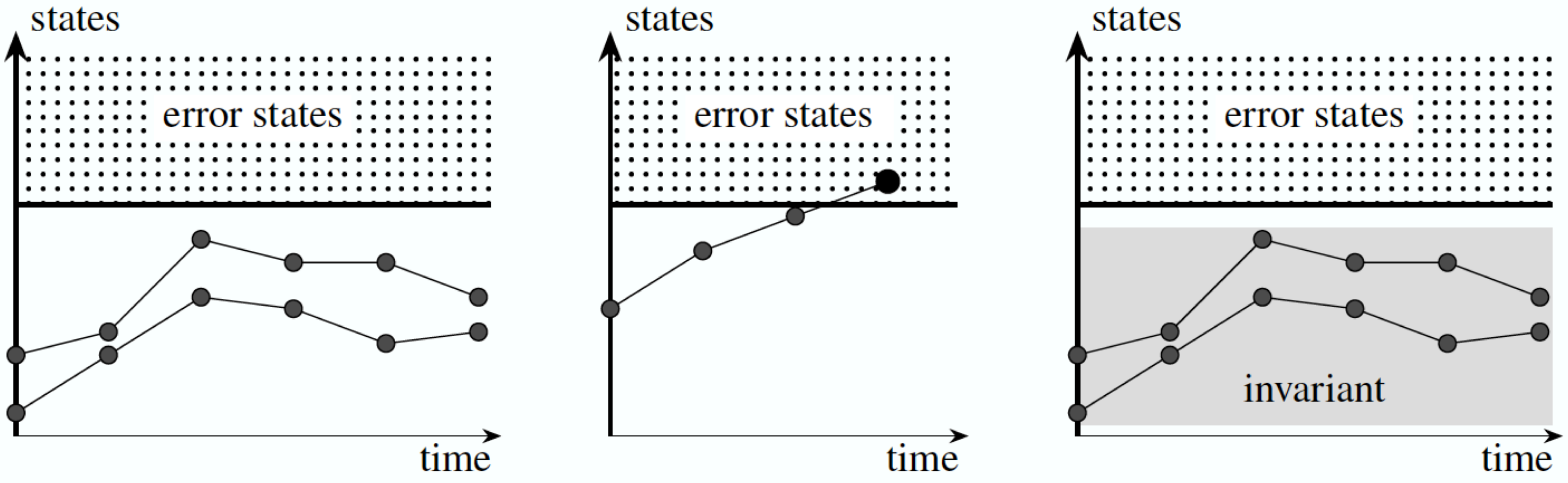
- No crashing error — e.g., no divide by zero, no uncaught exceptions, etc
- No invariant violation
  - Loop invariant: assertion that holds at the beginning of every loop iteration

What to Analyse: Safety Properties

```
x = 0;
while (x < 10)
   { x = x + 1;}
```

“x is an integer”

“0 <= x < 10”



(a) Correct executions (b) An incorrect execution (c) Proof by invariance

What to Analyse: Liveness Properties

Some behaviors observable after infinite time will never occur

Examples:

- No unbounded repetition of a given behavior
- No non-termination

What to Analyse: Liveness Properties

```
x = read_int ();
while ( x > 0 )
   { x = x - 1; }
```

- If x is initially a negative integer ⇒ the program terminates
- If x is initially a positive integer ⇒ x strictly decreases every iteration

⇒ the program terminates

Undecidability in the way

non trivial property:

- there exists a program c such that $\mathscr{P}(c)$ holds true
- and there exists also some program c such that $\mathscr{P}(c)$ is false

Rice theorem.

Let $\mathscr{P}(c)$ be a non trivial semantic property of programs c.

There exists no algorithm such that, for every program c,

it returns true if and only if $\mathscr{P}(c)$ holds true

no analysis method that is automatic, universal, exact !

For some program…

$\mathscr{P}(c) \equiv$ per ogni insieme di input lo stato finale assegna a x un valore diverso da 0

$c \triangleq$

```
x := 1;
```

and for some other program…

$\mathscr{P}(c) \equiv$ per ogni insieme di input lo stato finale assegna a x un valore diverso da 0

$c \triangleq$

```
while (n>1) {
    n := n+1;
    x := 0;
}
x := 1;
```

but for Collatz's conjecture?

$\mathscr{P}(c) \equiv$ per ogni insieme di input lo stato finale assegna a x un valore diverso da 0

$c \triangleq$

```
while (n>1) {
    if (even(n)) { n := n/2; }
    else { n:= 3n+1; }
} % does it terminate for any value of n?
x := 1;
```

As of 2020, the conjecture has been checked by computer for all starting values up to $2^{68} \approx 10^{20}$.

Limitations of the analysis

We need to give something up:

automation: machine-assisted techniques

the **universality** "for all programs": targeting only a restricted class of programs

claim to find **exact** answers: introduce approximations

Approximation: Soundness and Completeness

Given a semantic property P and a program $p \in L$. An analysis is perfectly accurate iff

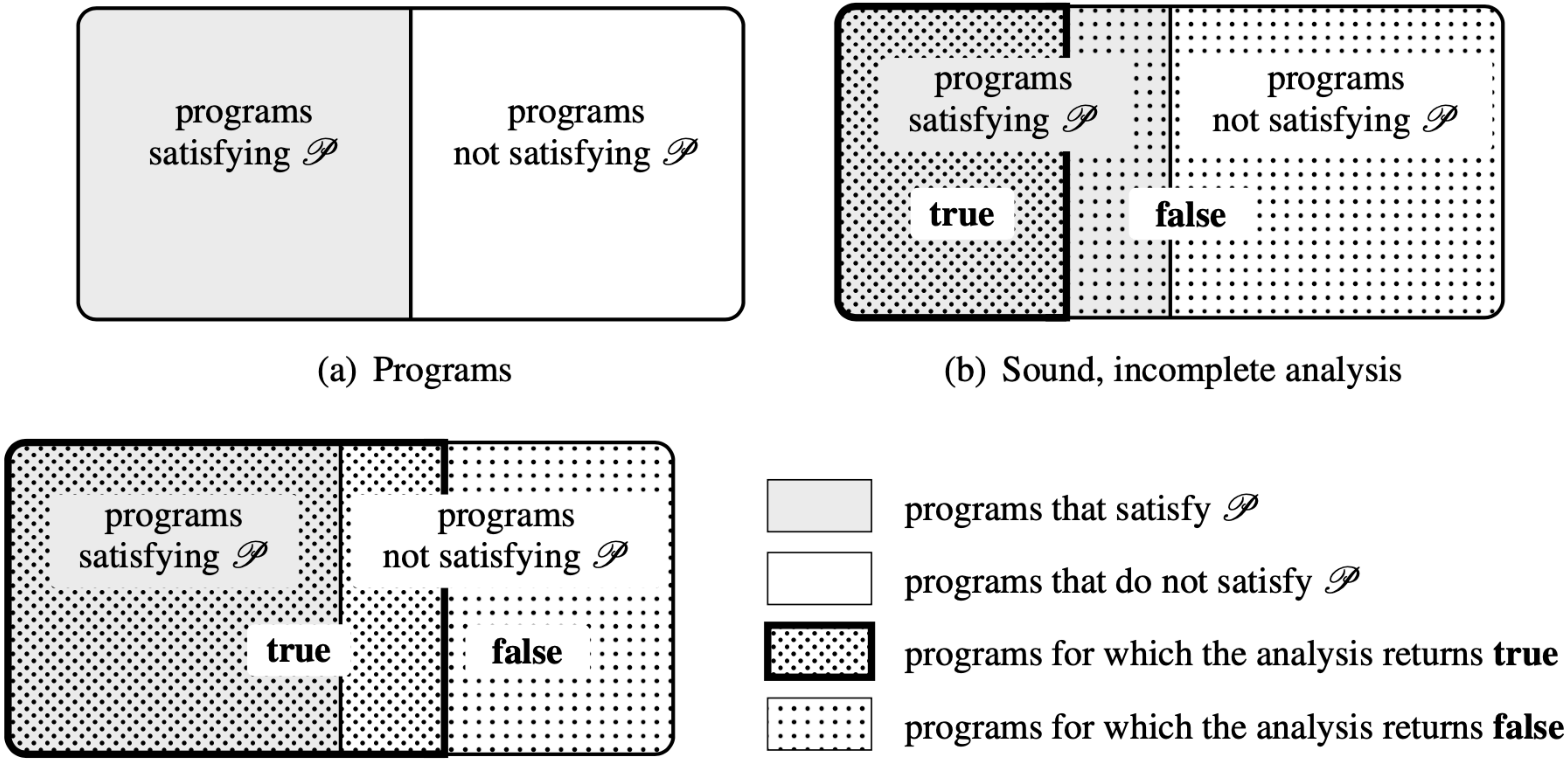
for all program p, analysis(p) = true $\Leftrightarrow$ p satisfies the property P

which consists of

1) for all program $p \in L$, analysis (p) = true $\Rightarrow$ p satisfies P (soundness)

2) for all program $p \in L$, analysis(p) = true $\Leftarrow$ p satisfies P (completeness)

Approximation: Soundness and Completeness



(a) Programs

(b) Sound, incomplete analysis

(c) Unsound, complete analysis

(d) Legend

Spectrum of Program Analysis Techniques

Testing

Machine-assisted proving

Finite-state model checking

Conservative static analysis

Bug-finding

Comparison

	automatic	sound	complete
testing	yes	no	yes
machine-assisted proving	no	yes	yes/no
finite-state model checking	yes	yes/no	yes/no
conservative static analysis	yes	yes	no
bug-finding	yes	no	no

Abstract Interpretation

A general technique, for any programming language L and safety property S, that checks, for input program P in L, if the semantics of a program P is contained in S

automatic (software)
finite (terminating)
sound (guarantee)
malleable for arbitrary precision

Denotational Semantics

Semantics

What is the meaning of a program "1 + 2" ?

Meaning = what it "denotes":

"3" (**Denotational semantics**)

Meaning = how to compute the result:

"add 1 into 2 and get 3" (**Operational semantics**)

Different approaches for different purposes and languages

Denotational Semantics

Mathematical meaning of a program (no machine states or transitions)

Program semantics is a function from input to output

The semantics of a program is determined by that of each component (compositionality principle)

Semantics of a Simple Language (WHILE)

$$
\begin{array}{lcll}
C & \rightarrow & \texttt{skip} & \\
 & | & x := E & \\
 & | & \texttt{if}\ E\ C\ C & \\
 & | & C; C & \\
 & | & \texttt{while}\ E\ C & \\
E & \rightarrow & n & (n \in \mathbb{Z}) \\
 & | & x & \\
 & | & E\ \texttt{+}\ E & \\
 & | & \texttt{-}\ E &
\end{array}
$$

The semantics of C is a function from memories to memories

Memory = Function from memory locations to values

Semantic Domain

A set of objects used to define program semantics (i.e., semantic objects)

$$x \in \mathbb{X} = ProgramVariables$$

$$\mathbb{V} = \mathbb{Z}$$

$$m \in \mathbb{M} = \mathbb{X} \to \mathbb{V}$$

Memories

Meaning of commands

$$[\![C]\!] : \mathbb{M} \to (\mathbb{M} \cup \bot)$$

may diverge

Meaning of expressions

$$[\![E]\!] : \mathbb{M} \to \mathbb{V}$$

Denotational Semantics of Expressions

$$
\begin{aligned}
[\![x]\!] \;\; m &= m\ (x) \\
[\![n]\!] \;\; m &= n \\
[\![E_1 + E_2]\!] \;\; m &= ([\![E_1]\!]\, m) + ([\![E_2]\!]\, m) \\
[\![- E]\!] \;\; m &= -([\![E]\!]\, m)
\end{aligned}
$$

$$
\begin{aligned}
[\![3 + x]\!]\ \{x \mapsto 2, y \mapsto 1\} &= [\![3]\!]\ \{x \mapsto 2, y \mapsto 1\} + [\![x]\!]\ \{x \mapsto 2, y \mapsto 1\} \\
&= 3 + 2 = 5
\end{aligned}
$$

Compositional!

(i.e., the semantics of an expression is determined

by that of its sub-expressions)

Denotational Semantics of Commands

$$
\begin{aligned}
[\![\texttt{skip}]\!]\ m &= m \\
[\![x \texttt{ := } E]\!]\ m &= m\{x \mapsto [\![E]\!]m\} \\
[\![\texttt{if}\ E\ C_1\ C_2]\!]\ m &= \mathit{if}\ [\![E]\!]m \neq 0\ \mathit{then}\ [\![C]\!]m\ \mathit{else}\ [\![C_2]\!]m \\
[\![C_1\ ;\ C_2]\!]\ m &= [\![C_2]\!]([\![C_1]\!]m)
\end{aligned}
$$

memory update

may diverge

$[\![C]\!]\ \bot = \bot$

E.g., $[\![\texttt{x:=7;y:=3}]\!]\{\} = \{\texttt{x} \mapsto 7, \texttt{y} \mapsto 3\}$

Compositional!

(i.e., the semantics of a program is determined by that of its sub-components)

Semantics of Loops

The semantics of $\texttt{while}\ E\ C$

$$\llbracket \texttt{while}\ E\ C \rrbracket\, m$$
$$= \mathit{if}\ \llbracket E \rrbracket\, m \neq 0\ \mathit{then}\ \llbracket \texttt{while}\ E\ C \rrbracket (\llbracket C \rrbracket M)\ \mathit{else}\ \ m$$

is not compositional!

Not a definition, but a **recursive equation**!

Semantics of Loops

$$[\![\texttt{while}\ E\ C]\!]\ m$$
$$= \mathit{if}\ [\![E]\!]\ m \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]M)\ \mathit{else}\ \ m$$

how to denote functions:
λx . function body
where x is the parameter
e.g. $inc(x) = x + 1$ vs
$inc = \lambda x\,.\,x + 1$

$$[\![\texttt{while}\ E\ C]\!] =$$
$$\lambda m.\mathit{if}\ [\![E]\!]\ m \neq 0\ \mathit{then}\ [\![\texttt{while}\ E\ C]\!]([\![C]\!]m)\ \mathit{else}\ m$$

$$F_{E,C}(X) = \lambda m. \begin{cases} X([\![C]\!]\ m) & \text{if } [\![E]\!]\ m \neq 0 \\ m & \text{otherwise} \end{cases}$$

$$[\![\texttt{while}\ E\ C]\!] = F_{E,C}([\![\texttt{while}\ E\ C]\!])$$

Semantics of Loops

Semantics of a loop: a solution of this equation

$$[\![\texttt{while}\ E\ C]\!] = F_{E,C}([\![\texttt{while}\ E\ C]\!])$$

Semantics of Loops

Semantics of a loop: a solution of this equation

$$[\![\texttt{while}\ E\ C]\!] = F_{E,C}([\![\texttt{while}\ E\ C]\!])$$

Solution: a fixed point of $F_{E,C}$

$$F_{E,C}(X) = \lambda m.\ \begin{cases} X([\![C]\!]\,m) & \text{if } [\![E]\!]\,m) \neq 0 \\ m & \text{otherwise} \end{cases}$$

Domain for Commands

$[\![C]\!] : \mathbb{M} \to \mathbb{M}_\perp$ where $\forall m \in \mathbb{M},\ \perp \sqsubseteq m$

$$[\![\mathtt{while}\ E\ C]\!] = F_{E,C}([\![\mathtt{while}\ E\ C]\!])$$

$(\mathbb{M} \to \mathbb{M}_\perp)$

A partial function,

We can represent it as sets of pairs (m,m')

$$F_{E,C}(X) = \lambda m. \begin{cases} X([\![C]\!]\, m) & \text{if } [\![E]\!]\, m) \neq 0 \\ m & \text{otherwise} \end{cases}$$

$F_{E,C}$ $(\mathbb{M} \to \mathbb{M}_\perp) \to (\mathbb{M} \to \mathbb{M}_\perp)$

It is monotone and continuous on the domain of partial functions

Semantics of while

By applying Klene's theorem

$$[\![\texttt{while}\ E\ C]\!] = \text{fix}\, F_{E,C} = \bigsqcup_n F^n_{E,C}(\ \lambda\sigma.\ \bot\ )$$

$$\texttt{while}\ \underbrace{x > 1}_{E}\ \texttt{do}\ \underbrace{x := x - 1}_{C}$$

$$F_{E,C}(X) = \lambda m. \begin{cases} (m, X(m[m(x) - 1/x]) & m(x) > 1 \\ (m, m) & m(x) \leq 1 \end{cases}$$

$$F_{E,C}(X) = \lambda m. \begin{cases} (m, m') & m(x) > 1, (m[m(x) - 1/x], m') \in X \\ (m, m) & m(x) \leq 1 \end{cases}$$

$$F^0_{E,C}(\varnothing) = \varnothing$$

$$F^1_{E,C}(\varnothing) = \{(m, m) \mid m(x) \leq 1\} \quad \supseteq \{(m[1/x], m[1/x])\}$$

$$F^2_{E,C}(\varnothing) = F^1_{E,C}(\varnothing) \cup \{(m, m[1/x]) \mid m(x) = 2\}$$

$$F^3_{E,C}(\varnothing) = F^2_{E,C}(\varnothing) \cup \{(m, m[1/x]) \mid m(x) = 3\}$$

$$\cdots$$

$$F^n_{E,C}(\varnothing) = \{(m, m) \mid m(x) \leq 1\} \cup \{(m, m[1/x]) \mid 1 < m(x) \leq n\}$$

$$\cdots$$

$$\text{fix}\, F_{E,C} = \{(m, m) \mid m(x) \leq 1\} \cup \{(m, m[1/x]) \mid 1 < m(x)\}$$