Lecture 5 – Abstract Interpretation

SWS128: Software Analysis

Jihyeok Park

PLRG

2026 Fall

Recall

- A **dataflow analysis**: data and a join, a transfer per edge, a direction, an initial value. One engine, `solve`, runs them all.
- We said its answer is **sound** and the **least** one. We never said why.
- Today: the theory behind those two words. Dataflow analysis turns out to be one instance of it.

Contents

Contents

A Program and All Its Runs

```
def f(c):
    x = 1
    if c < 0:
        x = -1
    y = x + 2
    return y
```

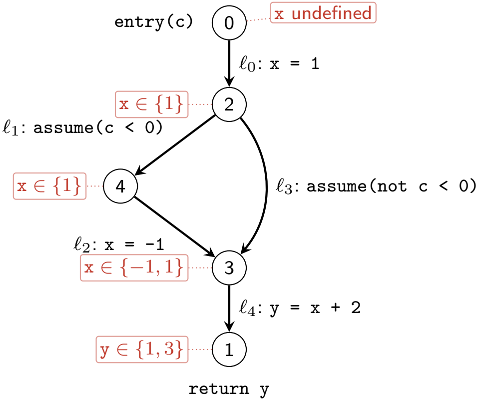
One run per input:

c	returns
-3	1
0	3
5	3
$\vdots$	$\vdots$

The interpreter answers for **one** c. An analysis must answer for **every** c at once: *is y always positive?*

Collecting Semantics

At each point, the values `x` and `y` can have, over **all** runs:



The exact answer: `y` is always positive. The equations are the dataflow ones, $\mathsf{C}(q) = \bigcup_{p \to q} f(\mathsf{C}(p))$, over sets of concrete values.

With a Loop, It Never Stops

```
def g(n):
    x = 0
    while x < n:
        x = x + 1
    return x
```

The values of `x` at the loop head, one iteration of the equations at a time:

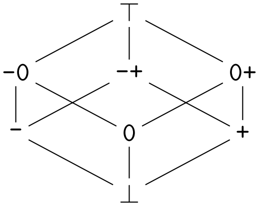
$$\{0\} \subset \{0,1\} \subset \{0,1,2\} \subset \cdots$$

- The chain never stabilizes: `n` can be any number.
- So the collecting semantics is exact but **uncomputable**. That is Rice's theorem again.

Abstract Domains

Replace each set of values by a **finite description** of it.

Sign: which of −, 0, + can occur



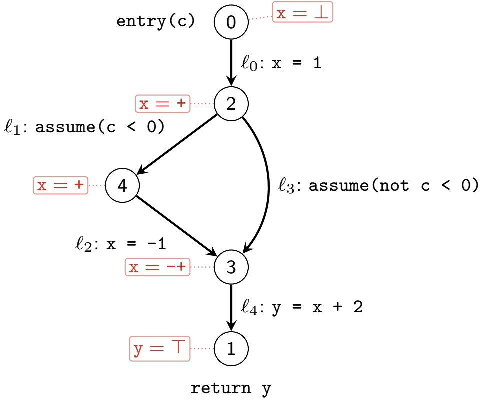
Interval: $[l, u]$, ordered by inclusion

$$[1,1] \sqsubseteq [-1,1] \sqsubseteq [-\infty,+\infty] = \top$$

S	$\alpha(S)$ in sign	$\alpha(S)$ in interval
$\{1\}$	+	$[1,1]$
$\{-1,1\}$	−+	$[-1,1]$
$\{0,1,2,\ldots\}$	0+	$[0,+\infty]$

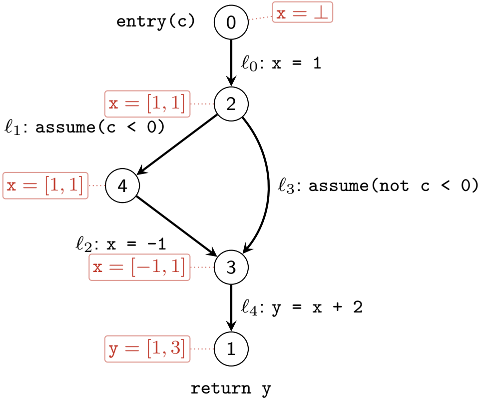
Contents

Sign, Step by Step



ℓ_4: y $= \textbf{-+} +^\sharp \textbf{+} = \top$, since $-1+2>0$ but $-5+2<0$. *Is y positive?* Sign **cannot tell**.

Interval, Step by Step



ℓ_4: $\mathrm{y} = [-1, 1] +^\sharp [2, 2] = [1, 3]$. *Is y positive?* Interval says **yes**.

Sign versus Interval

	concrete	sign	interval
x at 3	$\{-1, 1\}$	-+	$[-1, 1]$
y at 1	$\{1, 3\}$	$\top$	$[1, 3]$
y positive?	yes	*don't know*	yes

- Both are **correct**: every real value is inside what they say.
- Sign kept "not zero" but not *how big*: a negative plus 2 can be anything. Interval kept $[-1, 1]$, enough to add 2 and stay positive.
- Choosing a domain is choosing what you can prove.

Contents

Abstraction and Concretization

Two functions connect sets of values and the abstract domain:

- $\gamma(a)$: the set of values a describes.

$$\gamma(\texttt{+}) = \{1, 2, 3, \ldots\} \qquad \gamma([1, 3]) = \{1, 2, 3\}$$

- $\alpha(S)$: the best description of S.

$$\alpha(\{-1, 1\}) = \texttt{-+} \text{ (sign)} \qquad \alpha(\{-1, 1\}) = [-1, 1] \text{ (interval)}$$

They form a **Galois connection** when

$$\alpha(S) \sqsubseteq a \iff S \subseteq \gamma(a)$$

that is, a describes S correctly exactly when a is above $\alpha(S)$.

Abstraction Loses Information

$$S = \{1, 100\} \qquad \alpha(S) = [1, 100] \qquad \gamma(\alpha(S)) = \{1, 2, \ldots, 100\}$$

- Intervals cannot say "1 or 100". They say "between 1 and 100", and 98 values that never occur come with it.
- Always $S \subseteq \gamma(\alpha(S))$: abstraction adds values, never removes them. That is the price of computability.

Abstract Transformers

Each operation needs an abstract version. For sign, define it on the three signs:

$+^\sharp$	-	0	+
-	-	-	$\top$
0	-	0	+
+	$\top$	+	+

$\times^\sharp$	-	0	+
-	+	0	-
0	0	0	0
+	-	0	+

- Any other element is a set of signs: take every pair, then join.

$$\text{-+} +^\sharp \text{+} \;=\; (\text{-} +^\sharp \text{+}) \sqcup (\text{+} +^\sharp \text{+}) \;=\; \top \sqcup \text{+} \;=\; \top$$

- **Sound**: $f(\gamma(a)) \subseteq \gamma(f^\sharp(a))$. A negative times a negative is always positive, so $\text{-} \times^\sharp \text{-} = \text{+}$.
- Interval: $[a, b] +^\sharp [c, d] = [a + c, b + d]$.

Soundness

The analysis is **sound** when, at every point,

$$\mathsf{C}(n) \subseteq \gamma(\mathsf{A}(n))$$

	C	γ(sign)	γ(interval)
x at 2	$\{1\}$	$\{1, 2, \ldots\}$ ✓	$\{1\}$ ✓
x at 3	$\{-1, 1\}$	$\mathbb{Z} \setminus \{0\}$ ✓	$\{-1, 0, 1\}$ ✓
y at 1	$\{1, 3\}$	$\mathbb{Z}$ ✓	$\{1, 2, 3\}$ ✓

- Each transfer is sound and each join is sound, so the result is.
- This is the contract of every analysis in this course.

Contents

Knaster–Tarski

All the equations together are one function F on the table of abstract values. The answer is a table A with $F(A) = A$.

Theorem (Knaster–Tarski)

If $\mathbb{D}$ is a complete lattice and F is monotone, then F has a **least fixpoint**, $\mathbf{lfp}\, F$.

- Every fixpoint is sound. The least one is the most precise.
- So the answer we want exists, and there is exactly one.

Kleene Iteration

Start from $\bot$ and apply F until nothing changes:

$$\bot \sqsubseteq F(\bot) \sqsubseteq F^2(\bot) \sqsubseteq \cdots \qquad \mathbf{lfp} F = \bigsqcup_i F^i(\bot)$$

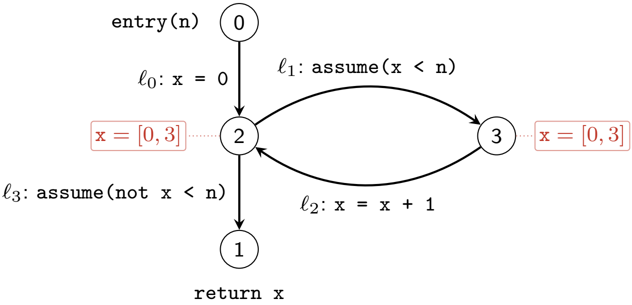
`x` at the loop head of g, in sign:

	F^0	F^1	F^2	F^3
`x`	$\bot$	0	$0 \sqcup (0 +^\sharp +) = 0+$	$0 \sqcup (0+ +^\sharp +) = 0+$

- Two steps and it is stable at 0+. The concrete chain $\{0\}, \{0, 1\}, \ldots$ never was.
- The chain **is** the worklist algorithm, minus the bookkeeping.
- Sign has finite height, so the chain must stop. Intervals do not.

Intervals Never Stop

The same loop g, in interval:



Each lap adds one: $[0,0] \sqsubseteq [0,1] \sqsubseteq [0,2] \sqsubseteq \cdots$ **never stops**.

- Sign stopped at `0+` after two steps. Interval has **infinite height**: the chain has no end.
- The engine gives up (`RuntimeError: no fixpoint`). The fix, **widening**, is the next lecture.

What Dataflow Analysis Was

Abstract interpretation	**Dataflow analysis**
abstract domain $\mathbb{D}$	sets of facts, ordered by $\subseteq$
abstract transformer $f^\sharp$	gen $\cup$ $(X \setminus$ kill$)$
join $\sqcup$	$\cup$ or $\cap$
$\mathbf{lfp}\, F$	what `solve` returned
soundness C $\subseteq \gamma$(A)	assumed, never stated

- A recipe for a new analysis: pick $\mathbb{D}$, define γ, write a sound $f^\sharp$ per operation, run the engine.

Contents

The Domain Interface

```
class Domain:
    bot: object
    top: object

    def le(self, a, b) -> bool: raise NotImplementedError
    def join(self, a, b): raise NotImplementedError
    def meet(self, a, b): raise NotImplementedError
    def alpha(self, ns: set[int]): raise NotImplementedError
    def gamma(self, a) -> str: raise NotImplementedError
    def of_num(self, n: int): raise NotImplementedError
    def binop(self, op: str, a, b): raise NotImplementedError
    def unop(self, op: str, a): raise NotImplementedError

    def widen(self, a, b):  return self.join(a, b)
    def narrow(self, a, b): return self.meet(a, b)
```

- The lattice (`le`, `join`, `meet`), the Galois connection (`alpha`, `gamma`), and the transformers (`of_num`, `binop`, `unop`).
- `widen` and `narrow` are for the next lecture.

The Sign Domain

```
# an element is a set of the three signs; its name is the key
ATOMS = {BOT: "", NEG: "-", ZERO: "0", POS: "+",
         NONPOS: "-0", NONNEG: "0+", NONZERO: "-+", TOP: "-0+"}
NAME = {frozenset(v): k for k, v in ATOMS.items()}

ADD = {("-", "-"): "-", ("-", "0"): "-", ("-", "+"): "-0+",
       ("0", "0"): "0", ("0", "+"): "+", ("+", "+"): "+"}

class Sign(Domain):
    def join(self, a, b):
        return NAME[frozenset(ATOMS[a] + ATOMS[b])]
    def binop(self, op, a, b):
        ...
        out = ""
        for x in ATOMS[a]:                          # every pair of signs
            for y in ATOMS[b]:
                out += table.get((x, y)) or table[(y, x)]
        return NAME[frozenset(out)]
```

- Join is union of signs. An operation is the three-sign table, applied to every pair.

Running It

```
>>> from syntax import parse
>>> from cfg import build
>>> from domain import Sign, analyze, returned
>>> src = """
... def f(c):
...     x = 1
...     if c < 0:
...         x = -1
...     y = x + 2
...     return y
... """
>>> g = build(parse(src))
>>> d = Sign()
>>> d.join("-", "+"), d.binop("+", "-+", "+")
('-+', 'top')
>>> res = analyze(g, d)
>>> res[3]["x"], returned(g, d, res)
('-+', 'top')
```

- The same answers as the step-by-step: x is $-+$ at the merge, y is $\top$. The interval domain is the next homework.

Summary

1. From Concrete to Abstract
   - Collecting Semantics
   - Abstract Domains
2. Two Analyses
   - Step by Step
3. Galois Connections
   - The Connection
   - Transformers
4. Fixpoints
   - Existence
   - Computation
5. In Python
   - Code
   - Running It

Next Lecture

- Widening and Narrowing

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr