Fundamentals of Programming Languages V

Abstract Interpretation

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Assignment

Assignment 2 is announced! *Deadline Nov. 15*

Reference

Cousot P., Cousot R., Abstract Interpretation: A Unified Lattice Mode for Static Analysis of Programs by Construction or Approximation of Fixpoints, POPL'77

Steffen, B., Data Flow Analysis as Model Checking, TACS'91

Schmidt, D.A., Data Flow Analysis is Model Checking of Abstract Interpretations POPL'98

Lacey,D., Proving Correctness of Compiler Optimizations by Temporal Logic POPL'02

Turing Award

1986, Tony Hoare
For his fundamental contributions to the definition and design of programming languages...

1991, Robin Milner
For three distinct and complete achievements: 1) LCF, the mechanization of Scott's Logic of Computable Functions, probably the first theoretically based yet practical tool for machine assisted proof construction; 2) ML, the first language to include polymorphic type inference together with a type-safe exception-handling mechanism; 3) CCS, a general theory of concurrency. In addition, he formulated and strongly advanced full abstraction, the study of the relationship between operational and denotational semantics...

1996, Amir Pnueli
For seminal work introducing temporal logic into computing science and for outstanding contributions to program and systems verification...

2007, Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis
For their roles in developing model checking into a highly effective verification technology, widely adopted in the hardware and software industries...

Abstract Interpretation in Practice

Hardware Model Checking

- Classical Model Checking (in Clarke's book)(hardware model checking): Implementer and verifier are same.
  - During designing a system, each model is checked and refined.
  - A correct system implementation is finally obtained.
- e.g. a modulo 8 counter
  - $v_0' = \neg v_0$
  - $v_1' = v_0 \oplus v_1$
  - $v_2' = (v_0 \wedge v_1) \oplus v_2$

Software Model Checking

- Modern Model Checking(software model checking): Implementer and verifier may be different.
  - Given a system implementation (e.g. a source code), construct a model and verify it.
  - Construction of models: manually/automatically.
- e.g.
  - security protocols
  - real-time embedded systems
  - program analysis

Classical Program Analysis

- From Dragon book
  - dead code detection
  - constant propagation
  - common expression detection
  - code motion

Classical Program Analysis

- From Dragon book
  - dead code detection
  - constant propagation
  - common expression detection
  - code motion
- From POPA
  - available expression analysis
  - reaching definition analysis
  - very busy expression analysis
  - live variable analysis

Classical Program Analysis

- From Dragon book
  - dead code detection
  - constant propagation
  - common expression detection
  - code motion
- From POPA
  - available expression analysis
  - reaching definition analysis
  - very busy expression analysis
  - live variable analysis
- Object-oriented program analysis
  - point-to analysis
  - escape analysis
  - inference of class invariants
  - shape analysis

Classical Program Analysis

- From Dragon book
  - dead code detection
  - constant propagation
  - common expression detection
  - code motion
- From POPA
  - available expression analysis
  - reaching definition analysis
  - very busy expression analysis
  - live variable analysis
- Object-oriented program analysis
  - point-to analysis
  - escape analysis
  - inference of class invariants
  - shape analysis
- Concurrent program analysis
  - deadlock, livelock
  - data race
- Aspect-oriented program analysis
- ?????

Brief Idea of Abstract Interpretation

- In order to know properties of a program, execute it for all possible inputs.
- It will not terminate since domains of variables are often infinite.
- Abstract input data to a suitable finite domain. Then finiteness guarantees termination.
- Functions needs to be reinterpreted.

Brief Idea of Abstract Interpretation

- In order to know properties of a program, execute it for all possible inputs.
- It will not terminate since domains of variables are often infinite.
- Abstract input data to a suitable finite domain. Then finiteness guarantees termination.
- Functions needs to be reinterpreted.
- Examples:
  - Security protocols: a principal may receive infinitely many messages from environments.
    (some affects the principal, some are not)
  - Embedded systems: a scheduler may have to schedule infinitely many released tasks.
    (when time exceeds, all tasks have the same results)

Abstract Interpretation on Program Analysis: Examples

- Even/Odd analysis: $x \times (x + 1)$ is even.

Abstract Interpretation on Program Analysis: Examples

- Even/Odd analysis: $x \times (x + 1)$ is even.
  - Abstraction:
    - *Even*
    - *Odd*

Abstract Interpretation on Program Analysis: Examples

- Even/Odd analysis: $x \times (x + 1)$ is even.
  - Abstraction:
    - $Even$
    - Odd
  - Interpretation:
    - $Even + Even = Even$, $Even + Odd = Odd$,
    - $Even \times Odd = Even$, $\ldots$

Abstract Interpretation on Program Analysis: Examples

- Even/Odd analysis: $x \times (x + 1)$ is even.
  - Abstraction:
    - $Even$
    - Odd
  - Interpretation:
    - $Even + Even = Even$, $Even + Odd = Odd$,
    - $Even \times Odd = Even$, $\ldots$
- Positive/nagative analysis

Abstract Interpretation on Program Analysis: Examples

- Even/Odd analysis: $x \times (x + 1)$ is even.
  - Abstraction:
    - $Even$
    - Odd
  - Interpretation:
    - $Even + Even = Even$, $Even + Odd = Odd$,
    - $Even \times Odd = Even$, $\ldots$
- Positive/nagative analysis
  - Abstraction:
    - Pos
    - Neg
    - $Zero$

Abstract Interpretation on Program Analysis: Examples

- Even/Odd analysis: $x \times (x+1)$ is even.
  - Abstraction:
    - $Even$
    - Odd
  - Interpretation:
    - $Even + Even = Even$, $Even + Odd = Odd$,
    - $Even \times Odd = Even$, ...
- Positive/nagative analysis
  - Abstraction:
    - Pos
    - Neg
    - $Zero$
  - Interpretation:
    - $Pos + Pos = Pos$, $Pos + Neg = \{Pos, Zero, Neg\}$, ...
    - $Neg \times Neg = Pos$, $Pos \times Zero = Zero$, ...

Abstract Interpretation on Program Analysis: Examples

- Even/Odd analysis: $x \times (x + 1)$ is even.
  - Abstraction:
    - $Even$
    - Odd
  - Interpretation:
    - $Even + Even = Even$, $Even + Odd = Odd$,
    - $Even \times Odd = Even$, ...
- Positive/nagative analysis
  - Abstraction:
    - Pos
    - Neg
    - $Zero$
  - Interpretation:
    - $Pos + Pos = Pos$, $Pos + Neg = \{Pos, Zero, Neg\}$, ...
    - $Neg \times Neg = Pos$, $Pos \times Zero = Zero$, ...
  - The result will be a (sound) approximation.

Sound Abstraction

The result is an approximation

Sound Abstraction

The result is an approximation

Completeness: sufficient condition

- if no errors detected there are really no errors.

Soundness: necessary condition

- if errors detected they are really errors.

Sound Abstraction

The result is an approximation

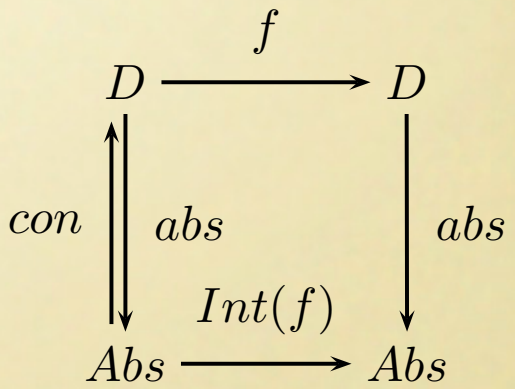
Completeness: sufficient condition

- if no errors detected there are really no errors.

Soundness: necessary condition

- if errors detected they are really errors.

- Data Abstraction : $abs : D \to Abs$
- Typical concretization : $con = abs^{-1}$
- $(abs \cdot con = id_{Abs}) \wedge (con \cdot abs \supseteq id_D)$ guarantee soundness.



Abstract Interpretation

Define abstraction and finite abstract domain

$$abs : D \to Abs$$

Abstract Interpretation

Define abstraction and finite abstract domain

$$abs : D \to Abs$$

Interpret primitive functions

$$+, \times, if, =, \geq, \leq, \ldots$$

Abstract Interpretation

Define abstraction and finite abstract domain

$$abs : D \to Abs$$

Interpret primitive functions

$$+, \times, if, =, \geq, \leq, \ldots$$

Execute looping structures via program semantics

while, function call

Program Semantics

Semantics: interpret a program (syntax) as an input-output relation (semantics).

Intuitively, the input-output relation is a relation between input environment and output environment of variables in a scopes.

Looping structure is solved as least fixed point.

Program Semantics

Semantics: interpret a program (syntax) as an input-output relation (semantics).

Intuitively, the input-output relation is a relation between input environment and output environment of variables in a scopes.

Looping structure is solved as least fixed point.

Theorem

Tarski's least fixed point theorem:
Assume $\tau : \mathscr{P}(D) \to \mathscr{P}(D)$ *is a monotonic, then there exists the unique least fixed point, which is computed as* $\bigcup_i \tau^i(\emptyset)$.

Program Semantics

Semantics: interpret a program (syntax) as an input-output relation (semantics).

Intuitively, the input-output relation is a relation between input environment and output environment of variables in a scopes.

Looping structure is solved as least fixed point.

Theorem

Tarski's least fixed point theorem:
Assume $\tau : \mathscr{P}(D) \to \mathscr{P}(D)$ *is a monotonic, then there exists the unique least fixed point, which is computed as* $\bigcup_i \tau^i(\emptyset)$.

We will come back to this theorem again!

Example: Computing Sum

```
/* sum */
1: read n;
2: i:=0;
3: S:=0;
4: c:=True;
5: while c
   do
6: S=S+i;
7: i:=i+1;
8: c:=i<=n;
   od;
9: write S
```

```
if ¬c
then skip
else
 S:=S+i;
 i:=i+1;
 c:=i<=n;
```

τ

```
if ¬c
then skip
else
 S:=S+i;
 i:=i+1;
 c:=i<=n;
```

τ^2

$env = [n, i, s, c]$

$\tau : \mathscr{P}(env \times env) \to \mathscr{P}(env \times env)$

step 0: ϕ

step 1:

```
{([0,0,0,True],
  [0,1,0,False])}
```

step 2:

```
{([0,0,0,True],
  [0,1,0,False]),
 ([1,0,0,True],
  [1,2,1,False])}
```

Example: Positive/Negative Analysis

Abstraction: $Nat \to \{Pos, Zero\}$

```
/* sum */
1: read n;
2: i:=0;
3: S:=0;
4: c:=True;
5: while c
   do
6: S=S+i;
7: i:=i+1;
8: c:=i<=n;
   od;
9: write S
```

```
if ¬c
then skip
else
 S:=S+i;
 i:=i+1;
 c:=i<=n;
```

τ

```
if ¬c
then skip
else
 S:=S+i;
 i:=i+1;
 c:=i<=n;
```

τ^{12}

$env = [n, i, s, c]$

$\tau : \mathcal{P}(env \times env) \to \mathcal{P}(env \times env)$

step 0: ϕ

step 1:

```
{([Zero,Zero,Zero,True],
  [Zero,Pos,Zero,False])}
```

step 2:

```
{([Zero,Zero,Zero,True],
  [Zero,Pos,Zero,False]),
  [Pos,Zero,Zero,True],
  [Pos,Pos,Pos,
            True/False])}
```

Least Fixed Point!

Example: Variation

Abstraction: $Nat \rightarrow \{Many, One, Zero\}$

```
/* sum */
1: read n;
2: i:=0;
3: S:=0;
4: c:=True;
5: while c
   do
6: S=S+i;
7: i:=i+1;
8: c:=i<=n;
   od;
9: write S
```

```
if ¬c
then skip
else
 S:=S+i;
 i:=i+1;
 c:=i<=n;
```

τ

```
if ¬c
then skip
else
 S:=S+i;
 i:=i+1;
 c:=i<=n;
```

$env = [n, i, s, c]$

$\tau : \mathscr{P}(env \times env) \rightarrow \mathscr{P}(env \times env)$

step 0: ϕ

step 1:

$\tau^{2}_{'2}$ {([Zero,Zero,Zero,True], [Zero,One,Zero,False])}

step 2:

```
{([Zero,Zero,Zero,True],
  [Zero,One,Zero,False]),
  [One,Zero,Zero,True],
  [One,One,One,True])}
```

Program Analysis = Abstract Interpretation + Model Checking

Model Checking Based Program Analysis

- Each line of a program is regarded as nondeterministic transition.
- After abstraction, states becomes finite.
- Least fixed point computation over finite domains can be done by model checking.

Model Checking Based Program Analysis

- Each line of a program is regarded as nondeterministic transition.
- After abstraction, states becomes finite.
- Least fixed point computation over finite domains can be done by model checking.
- Advantage:
  - Obtain more precise results.
  - Need not develop each program for each analysis, just reduce the analysis to a model checking problem.
  - Save implementing time.

Model Checking Based Program Analysis

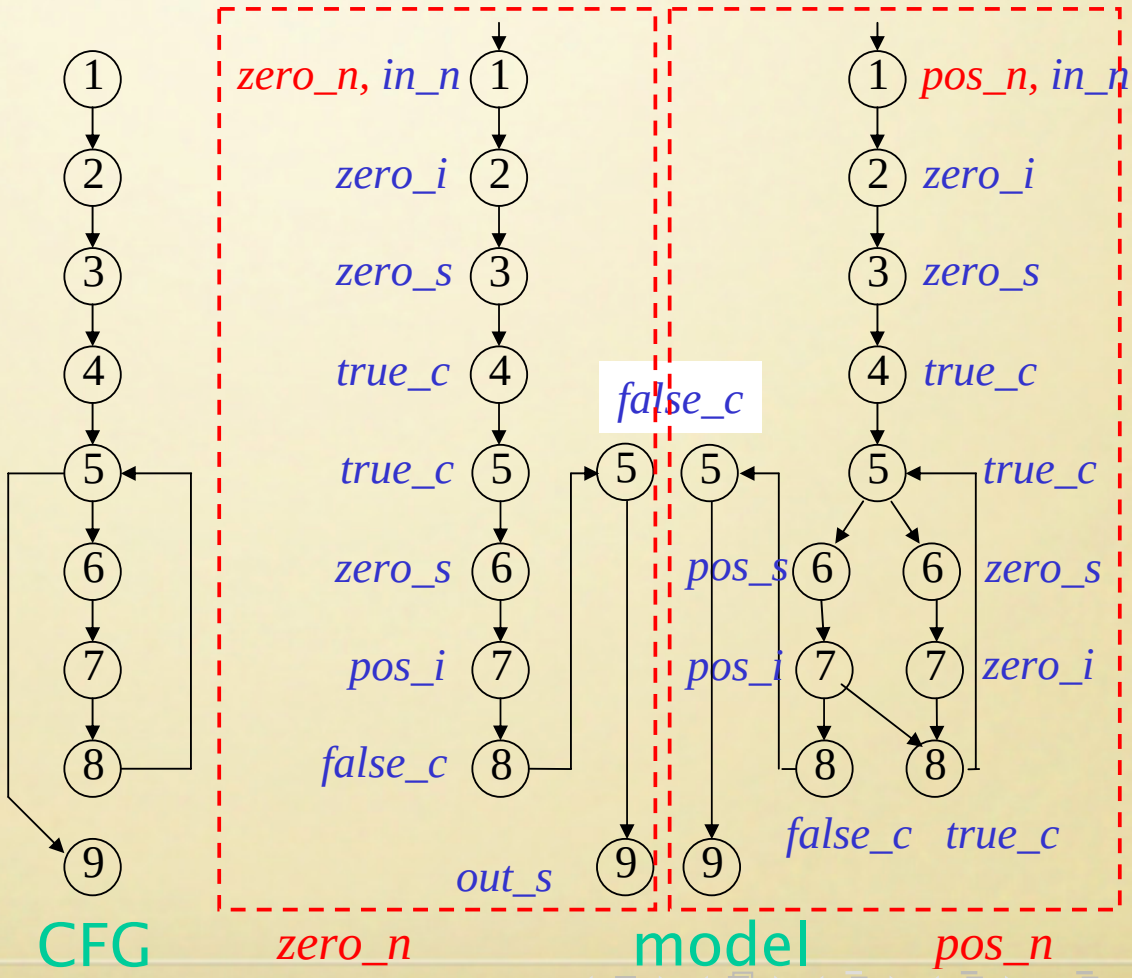
- Each line of a program is regarded as nondeterministic transition.
- After abstraction, states becomes finite.
- Least fixed point computation over finite domains can be done by model checking.
- Advantage:
  - Obtain more precise results.
  - Need not develop each program for each analysis, just reduce the analysis to a model checking problem.
  - Save implementing time.
- Disadvantage:
  - not so efficient. (do we really need preciseness?)

Example: Positive/Negative Analysis

Abstraction: $Nat \rightarrow \{Pos, Zero\}$

```
/* sum */
1: read n;
2: i:=0;
3: S:=0;
4: c:=True;
5: while c
   do
6: S=S+i;
7: i:=i+1;
8: c:=i<=n;
   od;
9: write S
```

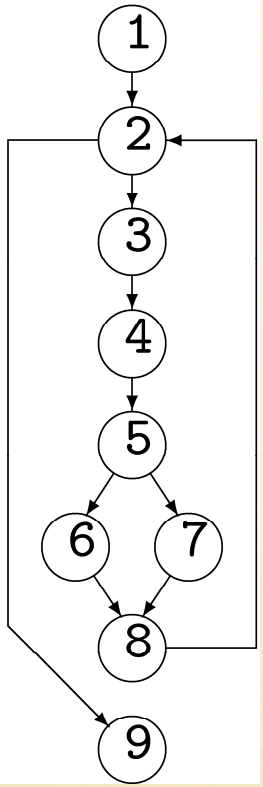
state = [n,i,s,c],
in_n, out_s
n,i,s,c: Boolean



Example: Even/Odd Analysis

```
1 :   read X;
2 :   while X > 1
      do
3 :   Z := X * 3 + 1;
4 :   C := X%2;
5 :   if C = 0
      then
6 :   X := X/2;
      else
7 :   X := Z;
      fi;
8 :   Z := X * 2;
      od;
9 :   write X;
```

This is expected to return 1



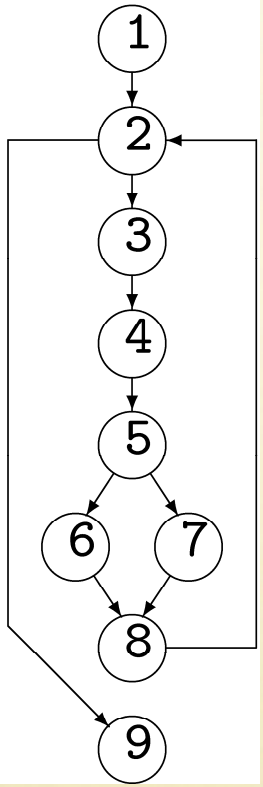
CFG

$$state = line\,x, [x, z, c]$$
$$read_X, write_X)$$
$$x, z, c : Bool(even/odd)$$

Example: Even/Odd Analysis

```
1 :   read X;
2 :   while X > 1
      do
3 :   Z := X * 3 + 1;
4 :   C := X%2;
5 :   if C = 0
      then
6 :   X := X/2;
      else
7 :   X := Z;
      fi;
8 :   Z := X * 2;
      od;
9 :   write X;
```

This is expected to return 1



CFG

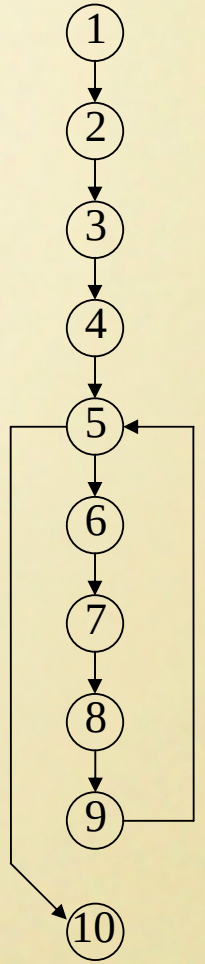
$state = line\,x, [x, z, c]$
$read_X, write_X)$
$x, z, c : Bool(even/odd)$

$AG(!EF(write_x)$
$\wedge x = even))$

Dead Code Detection

- Detect variables that are defined, but never used or redefined before used.
- Abstraction:
  - def_x when x is defined (e.g. $x := 2$)
  - use_x when x is referred (e.g. $y := x + 1$)

```
/* sum */
1:  read n;
2:  i:=0;
3:  S:=0;
4:  c:=True;
5:  while c
    do
6:  S:=S+i;
7:  c:=False;
8:  i:=i+1;
9:  c:=i<=n;
    od;
10: write S
```



CFG

Dead Code Detection

- Useless code: x is defined but never used.

Dead Code Detection

- Useless code: x is defined but never used.

$$AG\,\neg(def_x \wedge AX\,\neg EF\,use_x)$$

x is never defined if it will not used

Dead Code Detection

- Useless code: x is defined but never used.

$$AG\,\neg(def_x \wedge AX\,\neg EF\,use_x)$$

x is never defined if it will not used

- x is defined but not used before re-defined.

Dead Code Detection

- Useless code: x is defined but never used.

$$AG\,\neg(def_x \wedge AX\,\neg EF\,use_x)$$

x is never defined if it will not used

- x is defined but not used before re-defined.

$$AG\,\neg(def_x \wedge AX\,\neg A(\neg use_x\ U\ def_x)$$

Abstract Interpretation in Theory

Abstract Interpretation

Mathematical framework that helps the highlevel design of a program analysis.

Based on lattice theory.

Suggests to design an analysis by choosing appropriate lattices and functions between them.

PA in a Nutshell

Two main components of a program analysis:

- Abstract domain D s.t. $\bot \in D$
- Transfer function $F : D \to D$

Aim: Compute a fixpoint of F.

Algorithm: Start from $\bot$. Apply F repeatedly until fixpoint.

$$\bot, F(\bot), F^2(\bot), \ldots, F^n(\bot), F^{n+1}(\bot)$$

where n is the first s.t. $F^n(\bot) = F^{n+1}(\bot)$

A Running Example

assume($x == 2$);
while ($nondet()$) { $x = x + 2$; }
assert($x > 0$);

A Running Example

```
assume(x == 2);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

- $D = Sign = \{\bot, pos, neg, Z\}$ where $\bot = \emptyset$
- $F(d) = pos \cup (\text{if } (d = neg) \text{ then } Z \text{ else } d)$.
- Analysis run:

$$\bot = \emptyset, \quad F(\bot) = pos, \quad F^2(\bot) = pos$$

Main Concerns

Main Concerns

Q1: Does a fixpoint of F exist?

Q2: Is a fixpoint of F a correct answer?

Q3: Does our algorithm always terminate? If not, what should we do?

Main Concerns

Q1: Does a fixpoint of F exist?

Q2: Is a fixpoint of F a correct answer?

Q3: Does our algorithm always terminate? If not, what should we do?

Abstract interpretation answers these questions. It provides guidance about how to choose D and F.

Main Concerns

Q1: Does a fixpoint of F exist?

Q2: Is a fixpoint of F a correct answer?

Q3: Does our algorithm always terminate? If not, what should we do?

Abstract interpretation answers these questions. It provides guidance about how to choose D and F.

Tarski's Fixpoint Theorem

Answer of Q1

Recall the components of a abstract interpretation:

$$D, \quad F : D \to D \text{ such that } \bot \in D$$

.

Q1: Does F have a fixpoint?

Answer of Q1

Recall the components of a abstract interpretation:

$$D, \quad F : D \to D \text{ such that } \bot \in D$$

.

Q1: Does F have a fixpoint?

A1: If D is a complete lattice and F is monotone, F has the least and the greatest fixpoints.

Preorder and Partial Order

Preorder and Partial Order

A binary relation $\sqsubseteq$ on a set D is a preorder if

- reflexivity: $d \sqsubseteq d$ for all d in D
- transitivity: $(d \sqsubseteq e \wedge e \sqsubseteq f) \Rightarrow d \sqsubseteq f$

Preorder and Partial Order

A binary relation $\sqsubseteq$ on a set D is a preorder if

- reflexivity: $d \sqsubseteq d$ for all d in D
- transitivity: $(d \sqsubseteq e \wedge e \sqsubseteq f) \Rightarrow d \sqsubseteq f$

A preorder $\sqsubseteq$ is a partial order if

- antisymmetry: $(d \sqsubseteq e \wedge e \sqsubseteq d) \Rightarrow e = d$

Preorder and Partial Order

A binary relation $\sqsubseteq$ on a set D is a preorder if

- reflexivity: $d \sqsubseteq d$ for all d in D
- transitivity: $(d \sqsubseteq e \wedge e \sqsubseteq f) \Rightarrow d \sqsubseteq f$

A preorder $\sqsubseteq$ is a partial order if

- antisymmetry: $(d \sqsubseteq e \wedge e \sqsubseteq d) \Rightarrow e = d$

A preset means a set D with a preorder $\sqsubseteq$. A poset means a set D with a partial order $\sqsubseteq$. Usually written in $(D, \sqsubseteq)$

Preorder and Partial Order

A binary relation $\sqsubseteq$ on a set D is a preorder if

- reflexivity: $d \sqsubseteq d$ for all d in D
- transitivity: $(d \sqsubseteq e \wedge e \sqsubseteq f) \Rightarrow d \sqsubseteq f$

A preorder $\sqsubseteq$ is a partial order if

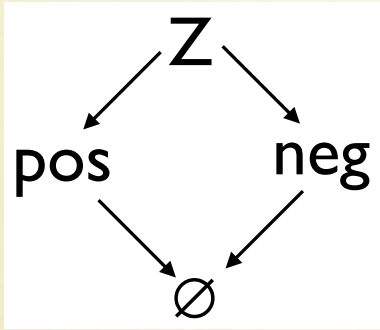
- antisymmetry: $(d \sqsubseteq e \wedge e \sqsubseteq d) \Rightarrow e = d$

A preset means a set D with a preorder $\sqsubseteq$. A poset means a set D with a partial order $\sqsubseteq$. Usually written in $(D, \sqsubseteq)$

Idea: $\sqsubseteq$ is an approximate subset/implication relation

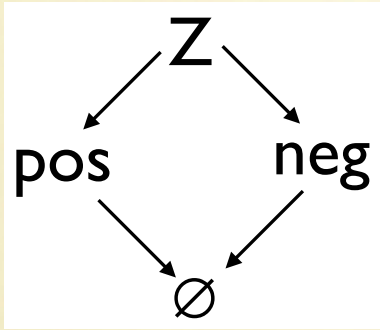
Quiz: Poset

Q1: Sign:



Quiz: Poset

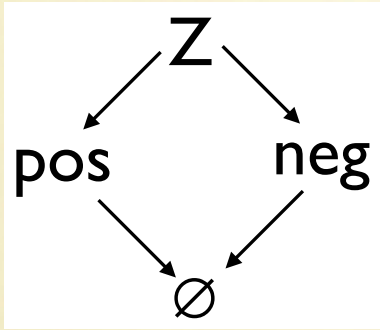
Q1: Sign:



Q2: Powerset of $\mathbb{Z}$: $(P(\mathbb{Z}), \subseteq)$

Quiz: Poset

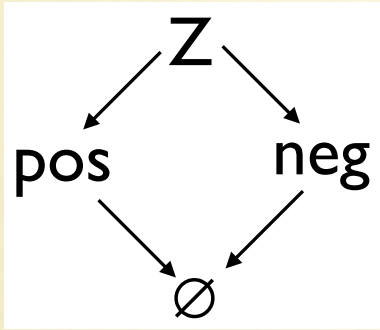
Q1: Sign:



Q2: Powerset of $\mathbb{Z}$: $(P(\mathbb{Z}), \subseteq)$

Q3: Interval: $(\{[n,m] \mid n,m \in \mathbb{Z} \wedge n \leq m\}, \subseteq)$

Quiz: Poset

Q1: Sign: 

Q2: Powerset of $\mathbb{Z}$: $(P(\mathbb{Z}), \subseteq)$

Q3: Interval: $(\{[n,m] \mid n,m \in \mathbb{Z} \wedge n \leq m\}, \subseteq)$

Q4: $LinCon_A : (\{Ax \leq b \mid b \in \mathbb{Z}_n\}, \Rightarrow)$

Least Upper Bound (lub)

Let $(D, \sqsubseteq)$ be a poset, and E a subset of D

- d is an upper bound of E if $e \sqsubseteq d$ for all $e \in E$
- d is a least upper bound (lub) of E if d is an upper bound and $d \sqsubseteq d'$ for all upper bound d' of E

Least Upper Bound (lub)

Let $(D, \sqsubseteq)$ be a poset, and E a subset of D

- d is an upper bound of E if $e \sqsubseteq d$ for all $e \in E$
- d is a least upper bound (lub) of E if d is an upper bound and $d \sqsubseteq d'$ for all upper bound d' of E
- Notations: $lub(E)$ and $\sqcup E$

Least Upper Bound (lub)

Let $(D, \sqsubseteq)$ be a poset, and E a subset of D

- d is an upper bound of E if $e \sqsubseteq d$ for all $e \in E$
- d is a least upper bound (lub) of E if d is an upper bound and $d \sqsubseteq d'$ for all upper bound d' of E
- Notations: $lub(E)$ and $\sqcup E$

Quiz: $lub(E)$ is unique?

Greatest Lower Bound (glb)

Dual to the definition of lub.

- d is an lower bound of E if $e \sqsupseteq d$ for all $e \in E$
- d is a greatest lower bound (glb) of E if d is a lower bound and $d \sqsupseteq d'$ for all lower bound d' of E.

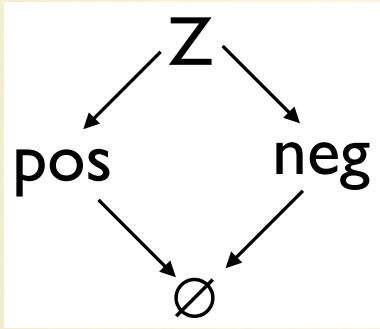
Greatest Lower Bound (glb)

Dual to the definition of lub.

- d is an lower bound of E if $e \sqsupseteq d$ for all $e \in E$
- d is a greatest lower bound (glb) of E if d is a lower bound and $d \sqsupseteq d'$ for all lower bound d' of E.
- Notations: $glb(E)$ and $\sqcap E$

Quiz: Compute lub

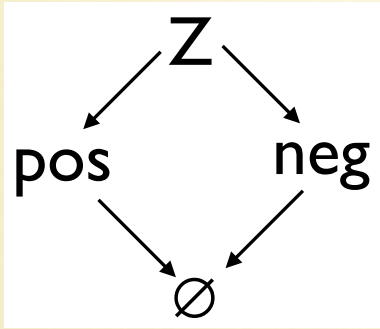
Q1: Sign:



- $pos \sqcup neg = ??$

Quiz: Compute lub

Q1: Sign:



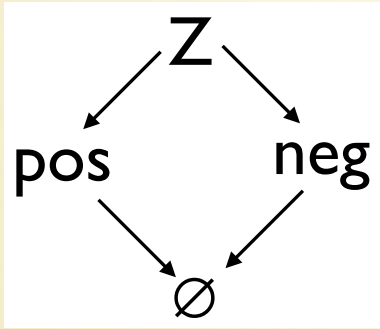
- $pos \sqcup neg = ??$

Q2: Powerset of $\mathbb{Z}$: $P(Z)$

- $\{2\} \sqcup \{4\} = ??$

Quiz: Compute lub

Q1: Sign:



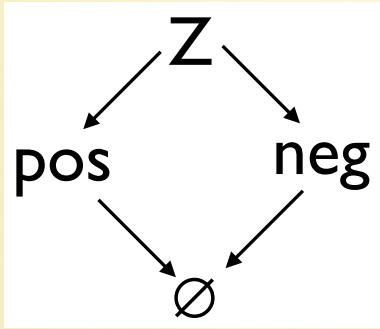
- $pos \sqcup neg = ??$

Q2: Powerset of $\mathbb{Z}$: $P(Z)$

- $\{2\} \sqcup \{4\} = ??$
- $\{2\} \sqcup \{4\} \sqcup \{6\} \sqcup \ldots = ??$

Quiz: Compute lub

Q1: Sign:



- $pos \sqcup neg = ??$

Q2: Powerset of $\mathbb{Z}$: $P(Z)$

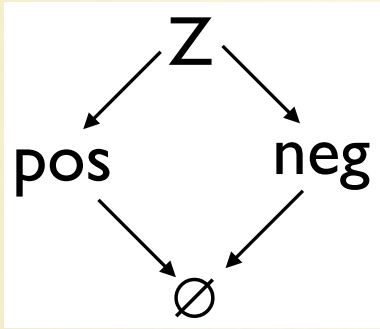
- $\{2\} \sqcup \{4\} = ??$
- $\{2\} \sqcup \{4\} \sqcup \{6\} \sqcup \ldots = ??$

Q3: Interval: $\{[n,m] \mid n,m \in \mathbb{Z} \wedge n \leq m\}$

- $[2,2] \sqcup [4,4] = ??$

Quiz: Compute lub

Q1: Sign:



- $pos \sqcup neg = ??$

Q2: Powerset of $\mathbb{Z}$: $P(Z)$

- $\{2\} \sqcup \{4\} = ??$
- $\{2\} \sqcup \{4\} \sqcup \{6\} \sqcup \ldots = ??$

Q3: Interval: $\{[n,m] \mid n,m \in \mathbb{Z} \wedge n \leq m\}$

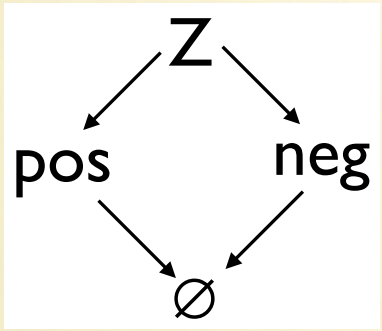
- $[2,2] \sqcup [4,4] = ??$
- $[2,2] \sqcup [4,4] \sqcup [6,6] \sqcup \ldots = ??$

Complete Lattice

A poset $(D, \sqsubseteq)$ is a complete lattice if every subset of D has the lub and glb.

Quiz: Complete Lattice

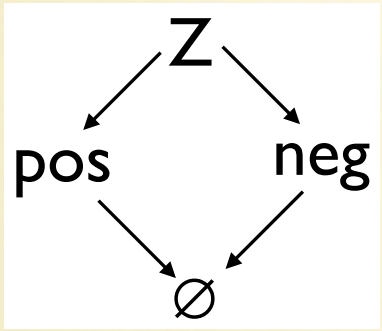
Q1: Sign:



Q2: Powerset of $\mathbb{Z}$: $(P(\mathbb{Z}), \subseteq)$

Q3: Interval: $([n,m] \mid n,m \in \mathbb{Z} \wedge n \leq m, \subseteq)$

Quiz: Complete Lattice

Q1: Sign: 

Q2: Powerset of $\mathbb{Z}$: $(P(\mathbb{Z}), \subseteq)$

Q3: Interval: $([n,m] \mid n,m \in \mathbb{Z} \wedge n \leq m, \subseteq)$

- $(\emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}, \subseteq)$

Tarski's fixpoint theorem

Let $(C, \subseteq)$ and $(D, \sqsubseteq)$ be presets. A function $F : C \to D$ is monotone if

$$c \subseteq c' \Rightarrow F(c) \sqsubseteq F(c')$$

.

Tarski's fixpoint theorem

Let $(C, \subseteq)$ and $(D, \sqsubseteq)$ be presets. A function $F : C \to D$ is monotone if

$$c \subseteq c' \Rightarrow F(c) \sqsubseteq F(c')$$

.

[Tarski's Theorem]
Every monotone function F on a complete lattice D has both the least and the greatest fixpoints.

Tarski's fixpoint theorem

Let $(C, \subseteq)$ and $(D, \sqsubseteq)$ be presets. A function $F : C \to D$ is monotone if

$$c \subseteq c' \Rightarrow F(c) \sqsubseteq F(c')$$

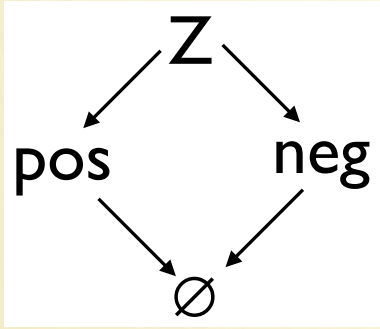
.

[Tarski's Theorem]
Every monotone function F on a complete lattice D has both the least and the greatest fixpoints.

Notations: $lfp(F)$ and $gfp(F)$.

Quiz: Compute Fixpoints

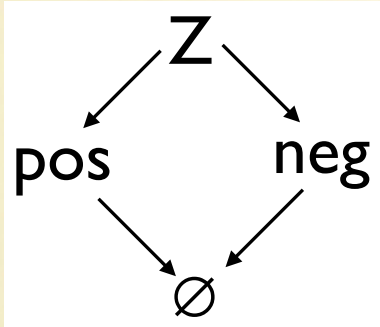
Q1: Sign:



- $F(d) = pos \sqcup (if\ (d = neg)\ then\ Z\ else\ d)$.

Quiz: Compute Fixpoints

Q1: Sign:



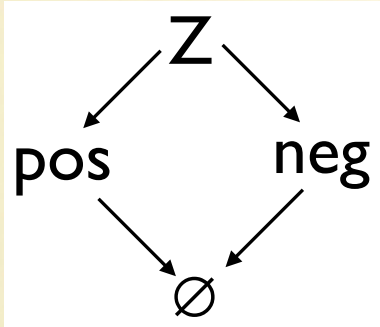
- $F(d) = pos \sqcup (if\ (d = neg)\ then\ Z\ else\ d)$.

Q2: Powerset of $\mathbb{Z}$: $P(Z)$

- $F(d) = \{2\} \sqcup \{i + 2 \mid i \in d\}$

Quiz: Compute Fixpoints

Q1: Sign:



- $F(d) = pos \sqcup (if\ (d = neg)\ then\ Z\ else\ d)$.

Q2: Powerset of $\mathbb{Z}$: $P(Z)$

- $F(d) = \{2\} \sqcup \{i + 2 \mid i \in d\}$

Q3: Interval: $(\emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}, \subseteq)$

- $F(d) = [2,2] \sqcup (\text{if } (d = [n,m]) \text{ then } [n+2, m+2] \text{ else } \emptyset)$

Proof of the Theorem

Tarski's Theorem

Every monotone function F on a complete lattice D has both the least and the greatest fixpoints.

Proof of the Theorem

Tarski's Theorem

Every monotone function F on a complete lattice D has both the least and the greatest fixpoints.

- Let $Pre = \{d \mid F(d) \sqsubseteq d\}$ and $Post = \{d \mid d \sqsubseteq F(d)\}$.

Proof of the Theorem

Tarski's Theorem

Every monotone function F on a complete lattice D has both the least and the greatest fixpoints.

- Let $Pre = \{d \mid F(d) \sqsubseteq d\}$ and $Post = \{d \mid d \sqsubseteq F(d)\}$.
- $glb(Pre)$ is the least fixpoint of F.

Proof of the Theorem

Tarski's Theorem

Every monotone function F on a complete lattice D has both the least and the greatest fixpoints.

- Let $Pre = \{d \mid F(d) \sqsubseteq d\}$ and $Post = \{d \mid d \sqsubseteq F(d)\}$.
- $glb(Pre)$ is the least fixpoint of F.
- $lub(Post)$ is the greatest fixpoint of F.

Finiteness Gains Termination

Lemma

If D is a finite complete lattice and $F : D \to D$ is monotonic, then the sequence

$$\bot, F(\bot), F^2(\bot), \ldots, F^n(\bot), \ldots$$

converges to $lfp(F)$ in finite steps.

Finiteness Gains Termination

Lemma

If D is a finite complete lattice and $F : D \to D$ is monotonic, then the sequence

$$\bot, F(\bot), F^2(\bot), \ldots, F^n(\bot), \ldots$$

converges to $lfp(F)$ in finite steps.

The lemma implies that our analysis terminates.

Finiteness Gains Termination

Lemma

If D is a finite complete lattice and $F : D \to D$ is monotonic, then the sequence

$$\bot, F(\bot), F^2(\bot), \ldots, F^n(\bot), \ldots$$

converges to $lfp(F)$ in finite steps.

The lemma implies that our analysis terminates.

Quiz:

- What if we drop the finiteness requirement

Finiteness Gains Termination

Lemma

If D is a finite complete lattice and $F : D \to D$ is monotonic, then the sequence

$$\bot, F(\bot), F^2(\bot), \ldots, F^n(\bot), \ldots$$

converges to $lfp(F)$ in finite steps.

The lemma implies that our analysis terminates.

Quiz:

- What if we drop the finiteness requirement
- Modify the lemma for greatest fixpoint

Main Concerns

Q1: Does a fixpoint of F exist?

Q2: Is a fixpoint of F a correct answer?

Q3: Does our algorithm always terminate? If not, what should we do?

Main Concerns

Q1: Does a fixpoint of F exist?

- Yes, if D is a complete lattice and F is monotonic.

Q2: Is a fixpoint of F a correct answer?

Q3: Does our algorithm always terminate? If not, what should we do?

Main Concerns

Q1: Does a fixpoint of F exist?

- Yes, if D is a complete lattice and F is monotonic.

Q2: Is a fixpoint of F a correct answer?

Q3: Does our algorithm always terminate? If not, what should we do?

Main Concerns

Q1: Does a fixpoint of F exist?

- Yes, if D is a complete lattice and F is monotonic.

Q2: Is a fixpoint of F a correct answer?

Q3: Does our algorithm always terminate? If not, what should we do?

- Use finite D.

Main Concerns

Q1: Does a fixpoint of F exist?

- Yes, if D is a complete lattice and F is monotonic.

Q2: Is a fixpoint of F a correct answer?

Q3: Does our algorithm always terminate? If not, what should we do?

- Use finite D.

Galois Connection

The Sign Example

```
assume(x == 2);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

- $D = Sign = \{\bot, pos, neg, Z\}$ where $\bot = \emptyset$
- $F(d) = pos \cup (\text{if } (d = neg) \text{ then } Z \text{ else } d)$.
- $lfp(F) = Pos$.

The Sign Example

```
assume(x == 2);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

- $D = Sign = \{\bot, pos, neg, Z\}$ where $\bot = \emptyset$
- $F(d) = pos \cup (\text{if } (d = neg) \text{ then } Z \text{ else } d)$.
- $lfp(F) = Pos$. Correct. It overapproximates all possible values of x at the loop entry.

The Sign Example

```
assume(x < 0);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

- $D = Sign = \{\bot, pos, neg, Z\}$ where $\bot = \emptyset$
- $F(d) = pos \cup (\text{if } (d = neg) \text{ then } Z \text{ else } d)$.
- $lfp(F) = Pos$. Correct?

The Guarantee of Correctness

```
assume(x == 2);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

$D = Sign$, $F(d) = pos \cup (\text{if } (d = neg) \text{ then } Z \text{ else } d)$

The Guarantee of Correctness

```
assume(x == 2);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

$D = Sign, F(d) = pos \cup (\text{if } (d = neg) \text{ then } Z \text{ else } d)$

- Define the concrete semantics (called collecting semantics):

$$C = P(Z), E(c) = \{2\} \cup \{i + 2 \mid i \in c\}$$

- Connect C and D via Galois connection
- Show that F is a sound abstraction of E
- Then, correct by fixpoint transfer theorem

Galois Connection

Let $(C, \subseteq)$ and $(D, \sqsubseteq)$ be presets. A Galois connection from C to D is a pair of monotone functions:

$$\alpha : C \to D \text{ (abstraction)}, \quad \gamma : D \to C \text{ (concretization)}$$

such that for all c in C and d in D,

$$\alpha(c) \sqsubseteq d \Leftrightarrow c \subseteq \gamma(d)$$

Galois Connection

Let $(C, \subseteq)$ and $(D, \sqsubseteq)$ be presets. A Galois connection from C to D is a pair of monotone functions:

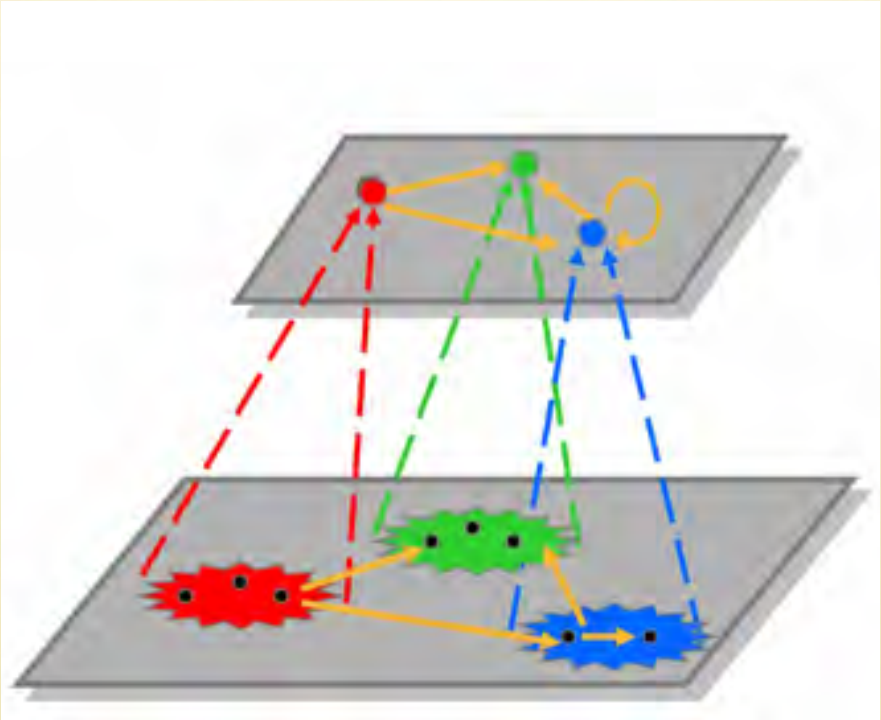
$$\alpha : C \to D \text{ (abstraction)}, \quad \gamma : D \to C \text{ (concretization)}$$

such that for all c in C and d in D,

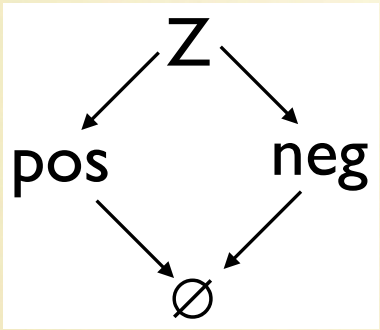
$$\alpha(c) \sqsubseteq d \Leftrightarrow c \subseteq \gamma(d)$$

Intuition: elements in D abstract those in C. $\alpha(c)$ is the best abstraction of c. $\gamma(d)$ is the meaning of d.

Galois Connection



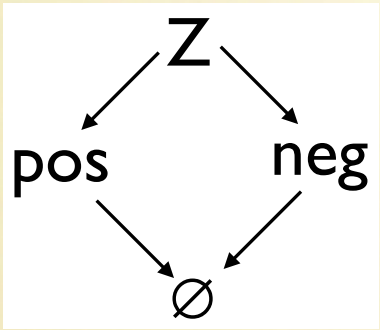
Quiz: Galois Connection

Sign: 

Powerset of $\mathbb{Z}$: $P(Z)$

Interval: $(\emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}, \subseteq)$

Quiz: Galois Connection

Sign: 

Powerset of $\mathbb{Z}$: $P(Z)$

Interval: $(\emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}, \subseteq)$

Find Galois connections

- From Powersets to Intervals
- From Intervals to Signs
- From Powersets to Signs

Soundness of Transfer Functions

Assume C, D: Complete lattices, and Galois connection from C to D: $\alpha : C \to D, \gamma : D \to C$

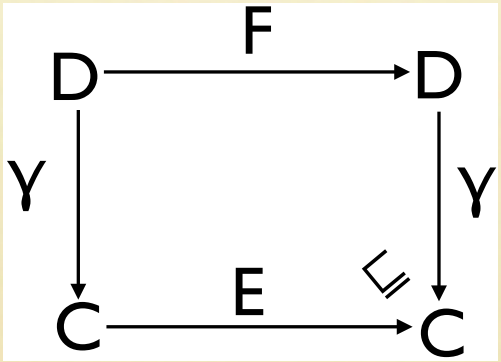
Soundness of Transfer Functions

Assume C, D: Complete lattices, and Galois connection from C to D: $\alpha : C \to D, \gamma : D \to C$

Let $E : C \to C$ and $F : D \to D$ be monotonic functions. F is a sound abstraction of E if

$$(E \circ \gamma)(d) \subseteq (\gamma \circ F)(d)$$

for all $d \in D$, diagrammatically,



Quiz: Which One is Sound

$C = P(Z), E(c) = \{2\} \cup \{i + 2 \mid i \in c\}\ D = \{\bot, pos, neg, Z\}$

Quiz: Which One is Sound

$C = P(Z), E(c) = \{2\} \cup \{i + 2 \mid i \in c\}\ D = \{\bot, pos, neg, Z\}$

- $F_1(d) = \text{if } (d = neg) \text{ then } Z \text{ else } (d \sqcup pos)$
- $F_2(d) = pos$
- $F_3(d) = Z$

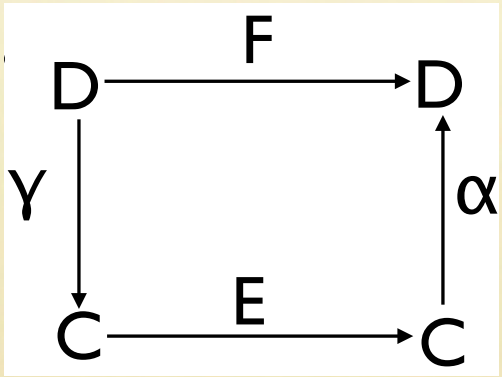
Best Abstraction

Let $E : C \to C$ and $F : D \to D$ be monotonic functions.

F is the best abstraction of E if

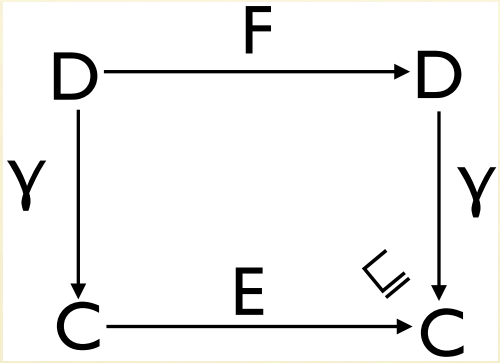
$$\alpha \circ E \circ \gamma = F$$

diagrammatically,



Fixpoint Transfer

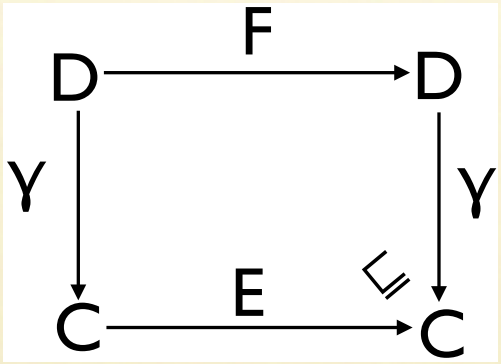
Let $E : C \to C$ and $F : D \to D$ be monotonic functions. If



then $lfp(E) \sqsubseteq \gamma(lfp(F))$

Fixpoint Transfer

Let $E : C \to C$ and $F : D \to D$ be monotonic functions. If



then $lfp(E) \sqsubseteq \gamma(lfp(F))$

Same condition. Preset D. For all d in D, if $F(d) \sqsubseteq d$, then $lfp(E) \subseteq \gamma(d)$.

Guarantee for Correctness

Given data:

- Program analysis $(D, F : D \to D)$
- Concrete semantics $(C, E : C \to C)$

1. Find a Galois connection from C to D
2. Show that F is a sound abstraction of E
3. Then, correct by fixpoint transfer theorem

Main Concerns

Q1: Does a fixpoint of F exist?

- Yes, if D is a complete lattice and F is monotonic.

Q2: Is a fixpoint of F a correct answer?

- Yes, if Galois connection, and sound transfer.

Q3: Does our algorithm always terminate? If not, what should we do?

- Use finite D.

Main Concerns

Q1: Does a fixpoint of F exist?

- Yes, if D is a complete lattice and F is monotonic.

Q2: Is a fixpoint of F a correct answer?

- Yes, if Galois connection, and sound transfer.

Q3: Does our algorithm always terminate? If not, what should we do?

- Use finite D.

Widening and Narrowing

Analysis with Intervals

```
assume(x == 2);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

- $D = \emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}$
- $F(d) = [2,2] \cup (\text{if } (d = [i,j]) \text{ then } [i+2, j+2] \text{ else } \emptyset)$.
- $lfp(F) = [2,+\infty]$.

Analysis with Intervals

```
assume(x == 2);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

- $D = \emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}$
- $F(d) = [2,2] \cup (\text{if } (d = [i,j]) \text{ then } [i+2, j+2] \text{ else } \emptyset)$.
- $lfp(F) = [2,+\infty]$. But it cannot be reached in finite steps. So, the analysis will not terminate.

Analysis with Intervals

```
assume(x == 2);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

- $D = \emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}$
- $F(d) = [2,2] \cup (\text{if } (d = [i,j]) \text{ then } [i+2, j+2] \text{ else } \emptyset)$.
- $lfp(F) = [2,+\infty]$. But it cannot be reached in finite steps. So, the analysis will not terminate.
- $\emptyset, F(\emptyset) = [2,2], F^2(\emptyset) = [2,4], F^3(\emptyset) = [2,6], \ldots$

Widening Operator ∇

binary operator $\nabla : D \times D \to D$

Widening Operator ∇

binary operator $\nabla : D \times D \to D$

such that

- $d_i \sqsubseteq d_1 \nabla d_2$ for $i \in \{1, 2\}$; and
- for every sequence $\{d_n\}_n$ in D, the following sequence $\{w_n\}_n$ has w_k with $w_k = w_{k+1}$:

$$w_0 = d_0, w_{n+1} = w_n \nabla d_{n+1}$$

Widening Operator ∇

binary operator $\nabla : D \times D \to D$

such that

- $d_i \sqsubseteq d_1 \nabla d_2$ for $i \in \{1, 2\}$; and
- for every sequence $\{d_n\}_n$ in D, the following sequence $\{w_n\}_n$ has w_k with $w_k = w_{k+1}$:

$$w_0 = d_0, w_{n+1} = w_n \nabla d_{n+1}$$

Intuition: $d_1 \nabla d_2$ extrapolates the change from d_1 to d_2.

Quiz: Which One is Widening

$$Interval = \emptyset \cup \{[i,j] \mid i,j \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge i \leq j\}$$

I. $\emptyset \nabla d = d \nabla \emptyset = d, [i,j] \nabla [i',j'] = [i'',j'']$ where

- $i'' = (i \leq i')?i : -\infty$
- $j'' = (j' \leq j)?j : +\infty$

II. $d \nabla d' = d \sqcup d'$

III. $\emptyset \nabla d = d \nabla \emptyset = d, [i,j] \nabla [i',j'] = [i'',j'']$ where

- $i'' = (min(i,i') < L)? - \infty : min(i,i')$
- $j'' = (max(j,j') > U)? + \infty : max(j,j')$

Program Analysis with Widening

Analysis: $(D, \sqsubseteq, \bot, F, \nabla)$

Algorithm: Generate w_k according to the following rule until $w_{k+1} = w_k$.

$$w_0 = \bot, w_{k+1} = w_k \nabla F(w_k)$$

Program Analysis with Widening

assume($x == 2$);
while ($nondet()$) { $x = x + 2$; }
assert($x > 0$);

- $D = \emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}$
- $F(d) = [2,2] \sqcup (\text{if } (d = [i,j]) \text{ then } [i+2,j+2] \text{ else } \emptyset)$.
- $\emptyset \nabla d = d \nabla \emptyset = d$,
- $[i,j] \nabla [i',j'] = [i'',j'']$ where
  - $i'' = (i \leq i')?i : -\infty$
  - $j'' = (j' \leq j)?j : +\infty$

Program Analysis with Widening

```
assume(x == 2);
while (nondet()) { x = x + 2; }
assert(x > 0);
```

- $D = \emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}$
- $F(d) = [2,2] \sqcup (\text{if } (d = [i,j]) \text{ then } [i+2,j+2] \text{ else } \emptyset)$.
- $\emptyset \nabla d = d \nabla \emptyset = d$,
- $[i,j] \nabla [i',j'] = [i'',j'']$ where
  - $i'' = (i \leq i')?i : -\infty$
  - $j'' = (j' \leq j)?j : +\infty$

Q: Simulate the analysis algorithm.

Program Analysis with Widening

Analysis: $(D, \sqsubseteq, \bot, F, \nabla)$

Algorithm: Generate w_k according to the following rule until $w_{k+1} = w_k$.

$$w_0 = \bot, w_{k+1} = w_k \nabla F(w_k)$$

Theorem The algorithm terminates.

Theorem On termination, $F(w_k) \sqsubseteq w_k$. Thus, correct if $\exists$-Galois connection and F is a sound abstraction.

Modification

Analysis: $(D, \sqsubseteq, \bot, F, \nabla)$

Algorithm: Generate w_k according to the following rule until $F(w_k) \sqsubseteq w_k$.

$$w_0 = \bot, w_{k+1} = w_k \nabla F(w_k)$$

Theorem The algorithm terminates.

Theorem On termination, $F(w_k) \sqsubseteq w_k$. Thus, correct if $\exists$-Galois connection and F is a sound abstraction.

Quiz

Design a widening operator for $P(\mathbb{Z})$.

Narrowing Operation Δ

Binary operator $\Delta : D \times D \to D$
such that

- $d_1 \sqcap d_2 \sqsubseteq d_1 \Delta d_2 \sqsubseteq d_1$ and
- for every decreasing sequence $\{d_n\}_n$ in D, the following sequence $\{v_n\}_n$ has v_k with $v_k = v_{k+1}$:

$$v_0 = d_0, \qquad v_{n+1} = v_n \Delta d_{n+1}$$

Intuition: $d_1 \Delta d_2$ interpolates the change from d_1 to $d_1 \sqcap d_2$.

Example

$$\mathit{Interval} = \emptyset \cup \{[i,j] \mid i,j \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge i \leq j\}$$

$$\emptyset \Delta d = d \Delta \emptyset = \emptyset$$

$$[i,j]\Delta[i',j'] = [(i = -\infty)?i' : i, (j = +\infty)?j' : j]$$

Program Analysis with Widening and Narrowing

Analysis: $(D, \sqsubseteq, \bot, F, \nabla, \Delta)$

Algorithm:

1. Generate w_k according to the following rule until $w_{k+1} = w_k$.

$$w_0 = \bot, w_{k+1} = w_k \nabla F(w_k)$$

2. Refine w_k by narrowing, Generate v_m according to the following rule until $v_{m+1} = v_m$

$$v_0 = w_k, v_{m+1} = v_m \Delta (v_m \sqcap F(v_m))$$

Example

```
assume(x == 2);
while (x < 9) { x = x + 2; }
```

- $D = \emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}$
- $F(d) = [2,2] \sqcup ((d \sqcap [-\infty,9] = [i,j])?[i+2,j+2] : \emptyset)$.
- $[i,j] \nabla [i',j'] = [(i \leq i')?i : -\infty, (j' \leq j)?j : +\infty]$
- $[i,j] \Delta [i',j'] = [(i = -\infty)?i' : i, (j = +\infty)?j' : j]$

Example

assume($x == 2$);
while ($x < 9$) { $x = x + 2$; }

- $D = \emptyset \cup \{[n,m] \mid n,m \in \mathbb{Z} \cup \{-\infty,+\infty\} \wedge n \leq m\}$
- $F(d) = [2,2] \sqcup ((d \sqcap [-\infty,9] = [i,j])?[i+2,j+2] : \emptyset)$.
- $[i,j]\nabla[i',j'] = [(i \leq i')?i : -\infty, (j' \leq j)?j : +\infty]$
- $[i,j]\Delta[i',j'] = [(i = -\infty)?i' : i, (j = +\infty)?j' : j]$

Q: Simulate the analysis algorithm.

Correctness

Why is the narrowing step correct?

Main Concerns

Q1: Does a fixpoint of F exist?

- Yes, if D is a complete lattice and F is monotonic.

Q2: Is a fixpoint of F a correct answer?

- Yes, if Galois connection, and sound transfer.

Q3: Does our algorithm always terminate? If not, what should we do?

- Use finite D. Or use widening.

Roadmap

Specify an analysis in terms of the following data:

1. Preset D (abstract domain).
2. Monotone function F (abstract transfer function).
3. Widening operator ∇ (unless D is finite).
4. Narrowing operator Δ (optional).
5. Galois connection from C (concrete domain) to D.
6. Soundness of F wrt. E (concrete transfer function).

Then, our analysis terminates and is correct (sound).

Report

Rep6. Algorithms for widening and narrowing (Maximal 3 students)