Lecture 10

Invariant Generation

Zvonimir Rakamarić
University of Utah

Announcements

- Project proposals – well done everybody!!!!!
- Z3 is open source now
- I will most likely be away Oct 22 & 24
  - Invited to Z3 Special Interest Group meeting
  - Ganesh on proving termination

Announcements: Student Presentations

- One presentation per student – 21 presentations
- One related work paper (or more)
- Oct 15, 17 and Oct 29, 31 in class
  - 5 per class, 6 on Oct 31 – we'll have to stay 15 mins after class
- 15 mins total – 13 for presentation + 2 for questions
  - Practice your presentation and don't go over your time budget

Announcements: Final Presentation

- 15 teams
- Two options:
  - In class: last 3 classes, 15 minutes per team
  - Conference-style: we cancel last 3 classes and everybody presents on the same day
    - I would try to find a day that works for everybody and book a room for the whole day (Dec 3-14)
    - Start in the morning
    - We could switch to 20 mins per team and make up for the Monday class that I unfortunately cancelled ☺
    - Order pizza for lunch???

Last Time

- Solving examples in PiVC

What is Going on Here?

- Thanks Mohammed for a weird (i.e., nice ☺) example

```
@pre true
@post rv <-> exists x.(x > 0 -> a[x] = e)
bool LinearSearch(int[] a, int e)
{
  return true;
// return false;
}
```

This Time

- Halting problem and Rice's theorem
- Abstraction
- Static analysis
- Abstract interpretation

Halting Problem

- Decision problem
  - Given a program and an input, decide whether the program will
    - eventually halt/terminate, or
    - run forever
- Alan Turing [1936]: halting problem is undecidable
  - A general algorithm for solving the halting problem for all possible program-input pairs cannot exist

    (Programs written in a Turing-complete language)

Proof by Contradiction

- Assume halting problem is decidable
  - We have a function `halts(f, i)` that returns true if function `f` halts on input `i`, and false otherwise
- Then, we define a function

```
test(f) {
  if (halts(f, f))
    while (true);
  else
    return;
}
```

- Invoke `test(test)`

Proof by Contradiction cont.

```
if (halts(test, test))
  while (true);
else
  return;
```

- Two cases:
  - `halts(test, test)` returns true but test loops forever – contradiction
  - `halts(test, test)` returns false but test returns – contradiction
- Hence halting problem is undecidable

Rice's Theorem

- Henry Gordon Rice, PhD thesis [1951]
- Essence
  - Any interesting question about the behavior of a program is undecidable
- Examples
  - Does this assertion always hold?
  - Is x always positive?
  - Which locations can pointer p point to?
  - What is a strong enough loop invariant?
  - …

Proof by Reduction from Halting Problem

- Assume `always_pos(f,x)` is decidable
- Reduce solving `halts` to solving `always_pos`

```
function solve_halts() {
   int x := 1;
   f(i);
   x := -1;
}
```

- Invoke `always_pos(solve_halts, x)`
  - If returns true then `f(i)` loops forever, otherwise `f(i)` halts
- Hence `always_pos(f, x)` is undecidable

Workarounds

- Interactive verification
  - Ask a user to provide required insight (e.g., loop invariants)
- Model checking
  - Ask a user to extract a finite-state model
- Static analysis
  - Automatically compute approximations of program behaviors using abstraction

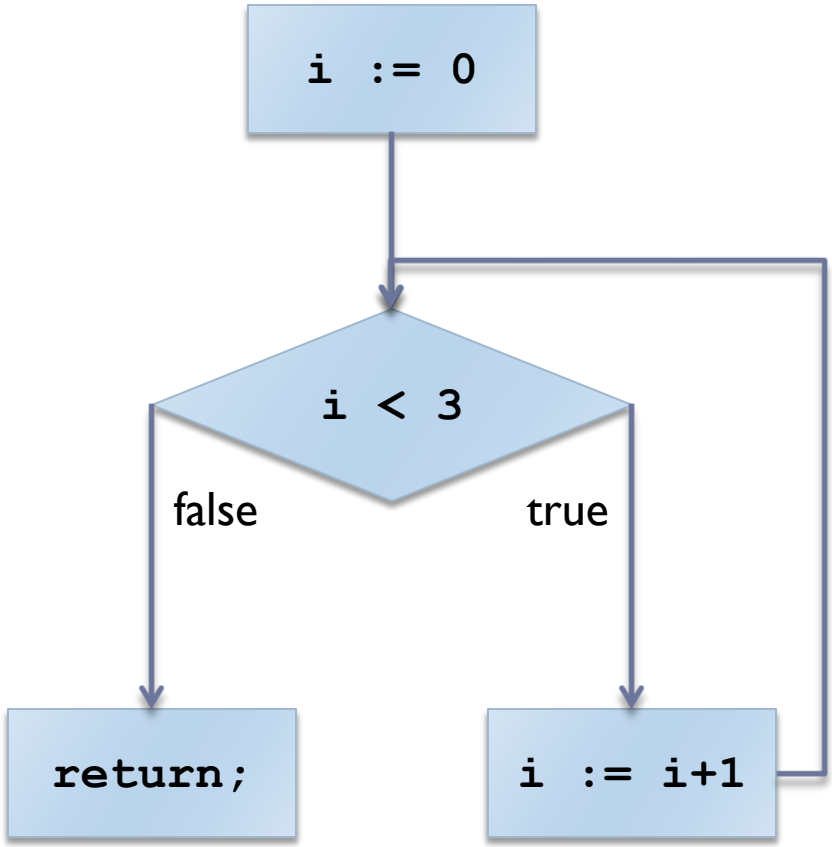
Abstraction

- Abstraction is a property from some domain
- Concrete value: 2
- Abstract value:
  - {2}
  - {2,5}
  - [2,10]
  - > 1
  - non-negative integer
  - even integer
  - integer
  - …

Example

```
1:   i := 0;
2:   while i < 3 do
3:     i = i + 1;
4:   end
5:   return;
```

Control Flow Graph

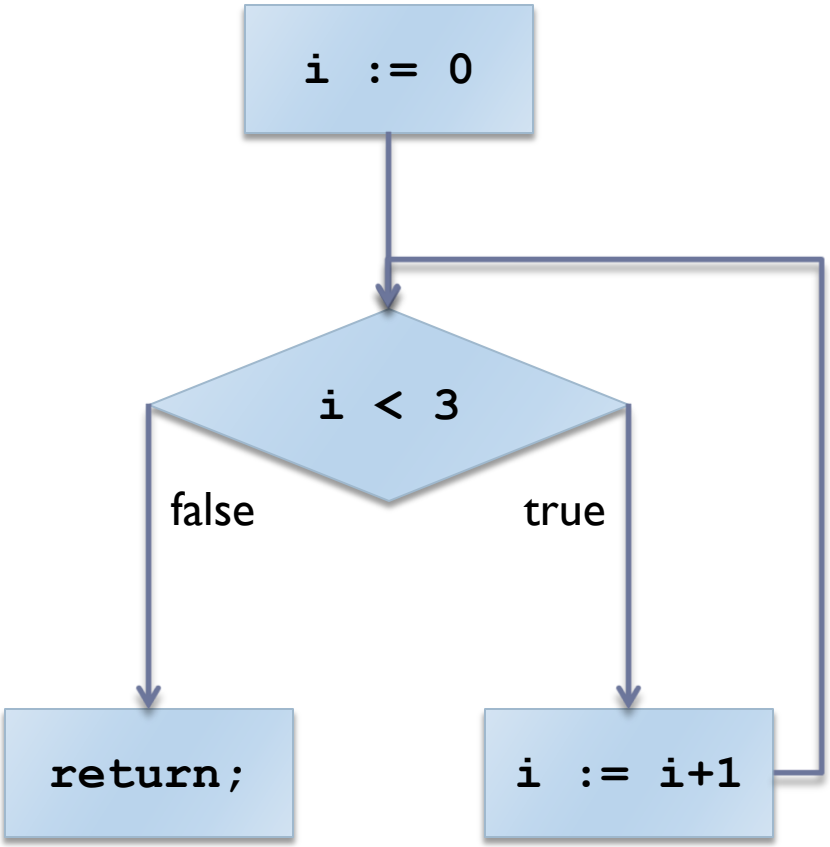


Abstract Domains

- Sign
  - $x \rightarrow +$, $y \rightarrow -$,...
- Intervals
  - $x \rightarrow [a, b]$, $y \rightarrow [a', b']$,...
- Polyhedra
  - $x + y - 2*z \leq 10$,...
- Difference-bound matrices
  - $y - x \leq 5$, $z - y \leq 10$,...
- …

Example with Intervals

- Concrete vs abstract semantics



Abstract Interpretation

- Patrick Cousot, Radhia Cousot [1977]
- Theoretical foundations and methodology for designing static analyses that are
  - Correct by construction
  - Generic
- Approximation is the core idea
- Needs (only) 3 things:
  - Control flow graph
  - Lattice
  - Transfer functions

Lattice

Partially-Ordered Set

- A partially-ordered set is a structure $(S, \sqsubseteq)$
  - S is a set
  - $\sqsubseteq$ is a partial order: a binary relation that for all x,y,z in S satisfies:
    - $x \sqsubseteq x$ [reflexivity]
    - $x \sqsubseteq y \wedge y \sqsubseteq x \rightarrow x=y$ [antisymmetry]
    - $x \sqsubseteq y \wedge y \sqsubseteq z \rightarrow x \sqsubseteq z$ [transitivity]
- There can be x and y in S such that they are incomparable

Least Upper Bound

- Upper bound of a set X is any b such that $x \sqsubseteq b$ for all $x \in X$
- Least upper bound of X is any lub such that:
  - lub is an upper bound of X
  - $lub \sqsubseteq b$ for any upper bound b of X
- If X has a least upper bound, then it is unique and written $\sqcup X$

Greatest Lower Bound

- Lower bound of a set X is any b such that $b \sqsubseteq x$ for all $x \in X$
- Greatest lower bound of X is any glb such that:
  - glb is a lower bound of X
  - $b \sqsubseteq glb$ for any lower bound b of X
- If X has a greatest lower bound, then it is unique and written $\sqcap X$

Lattice

- Lattice is a partially-ordered set $(L, \sqsubseteq)$ where every finite subset of L has a greatest lower and a least upper bound
- Complete lattice is a partially-ordered set $(L, \sqsubseteq)$ where every subset of L has a greatest lower and a least upper bound
- Complete lattice contains
  - Least element or “bottom” or $\bot$: $\bot = \sqcap L$
  - Greatest element or “top” or $\top$: $\top = \sqcup L$

Example: Interval Lattice (Hasse Diagram)

Fixpoint (Fixed Point)

- Fixpoint for f: $L \rightarrow L$ is any $x \in L$ such that $f(x)=x$

Least Fixpoint

- Least fixpoint of f is any lfp$\in$L such that:
  - lfp is a fixpoint of f
  - lfp$\sqsubseteq$x for any fixpoint x of f
- If f has a least fixpoint, then it is unique and written lfp(f)

Greatest Fixpoint

- Greatest fixpoint of f is any gfp$\in$L such that:
  - gfp is a fixpoint of f
  - x$\sqsubseteq$gfp for any fixpoint x of f
- If f has a greatest fixpoint, then it is unique and written gfp(f)

Fixpoint Theorem

- f is monotonic if $x \sqsubseteq y \rightarrow f(x) \sqsubseteq f(y)$
- Theorem:
  Every monotonic function f on a complete lattice $(L, \sqsubseteq)$ has a least fixpoint lfp(f) and a greatest fixpoint gfp(f)

Transfer Function

Abstraction and Concretization

- Function α maps concrete values into abstract values that best describe them (abstraction)

  α(2) = [2,10]

- Function γ maps abstract values into concrete values they represent (concretization)

  γ([2,10]) = {2,3,…,9,10}

- Abstraction followed by concretization is (usually) an approximation

  γ(α(2)) = γ([2,10]) = {2,3,…,9,10}

Approximation

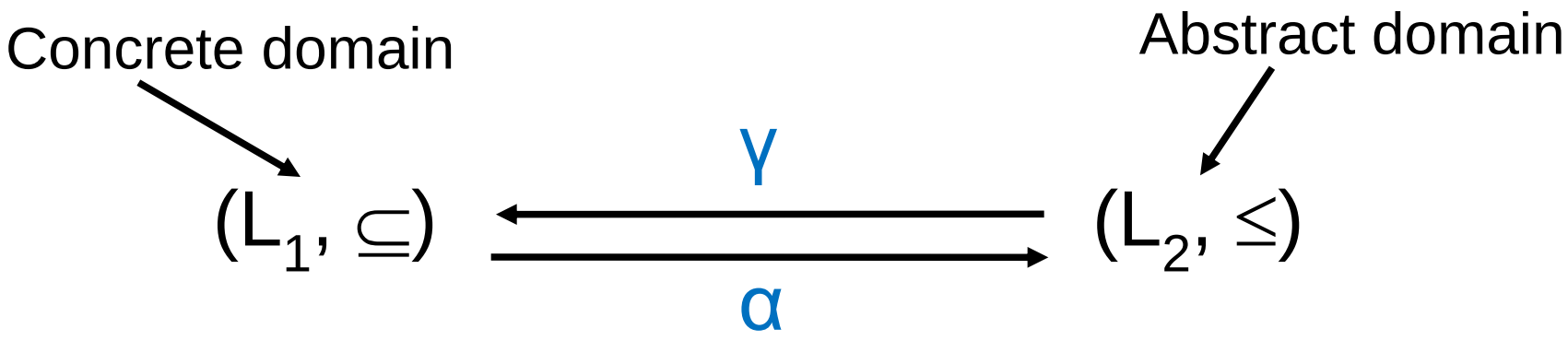
- Problem:
  Compute a sound approximation $S^{\#}$ of S:

$$S \subseteq S^{\#}$$

- Solution: Galois connection

Galois Connection

- L_1, L_2 are two lattices

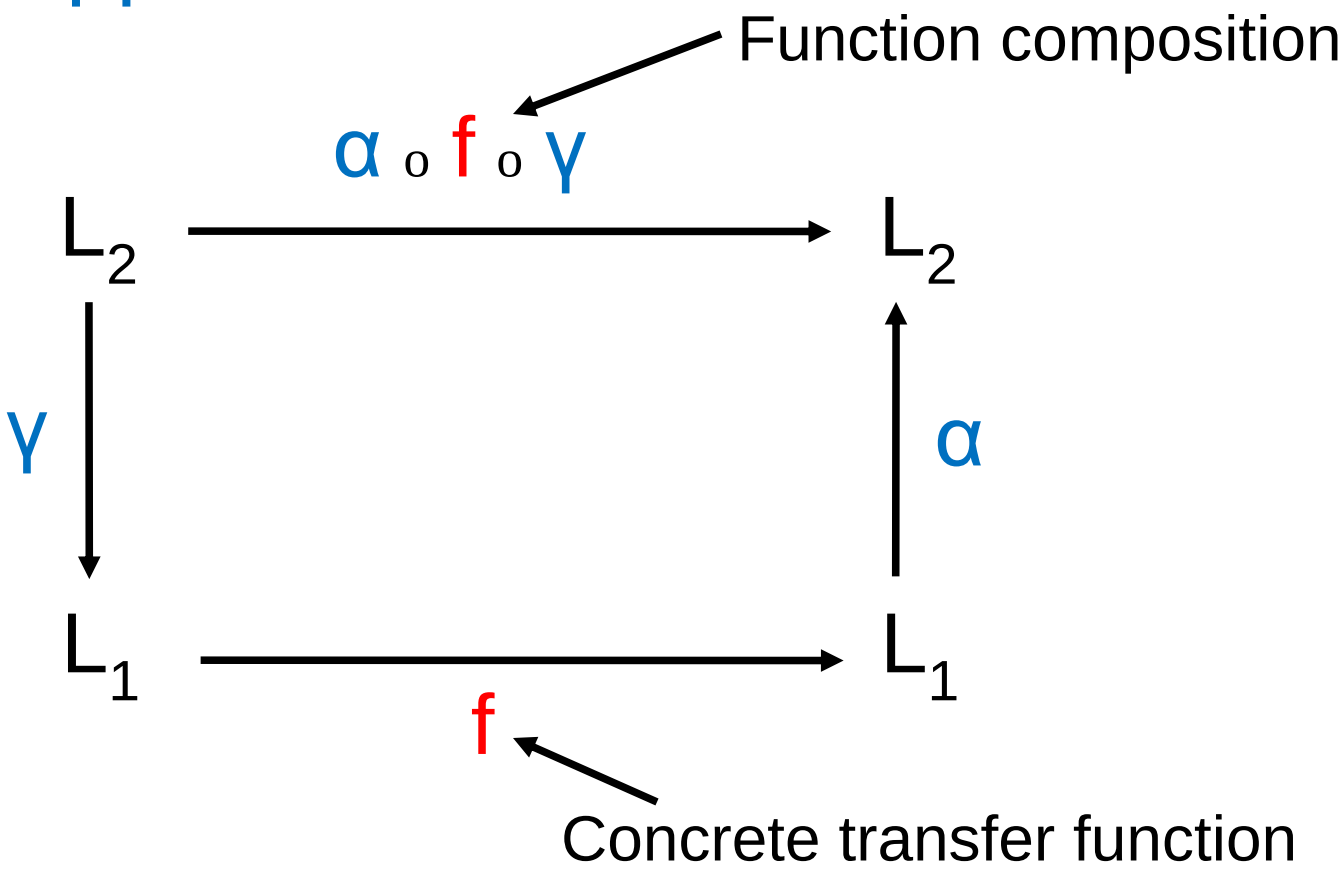


- Two conditions:

$$\forall x \forall y : \alpha(x) \leq y \leftrightarrow x \subseteq \gamma(y)$$

$$\forall x \forall y : x \subseteq \gamma(\alpha(x)) \wedge \alpha(\gamma(y)) \leq y$$

Fixpoint Approximation



- Theorem: $\text{lfp } f \subseteq \gamma(\text{lfp } \alpha \circ f \circ \gamma)$

Putting it all Together

- Define a lattice for your abstract domain
- Define α and γ that are monotonic and form a Galois connection
- Define an abstract transfer function $f^{\#}$:

$$\alpha \circ f \circ \gamma \leq f^{\#}$$