

- ◆ **New homework will be on the web soon**
- ◆ **Port your sign/parity domain into cXprop**
- ◆ **Analyze programs!**

# Last Time

- ◆ C semantics

# Today

- ◆ **Correctness of dataflow analysis**

- **Argument via abstract interpretation theory**
- **Goal: Relate analysis results to language semantics**



- ◆ **Material is difficult and I'm not an expert on it!**

- **Stop me if you get confused and we'll work things out in more detail**

# Motivation

- ◆ **So far, we've looked at how dataflow analysis works**
- ◆ **How do we know our analyses are correct?**
  - **We could reason about each individual analysis one a time**
  - **However, a unified framework would make proofs easier to develop and understand**
  - **Correctness criterion: Analysis result holds for all possible executions of a program**
- ◆ **Incorrect analyses typically lead to transformations that change a program's semantics**
  - **Clearly this is very bad**
  - **E.g. for safety critical or security critical systems**
  - **Compiler bugs particularly pernicious because no inspection of the application source will reveal the bug**

# Abstract Interpretation

- ◆ **Abstract interpretation provides a framework for reasoning about dataflow analyses**
- ◆ **Abstract interpretation is all about**
  - **Fixed points**
  - **Approximations**
- ◆ **In general, abstract interpretation is a theory of fixed point approximation**
  - **Abstract interpretation is very flexible**
  - **We won't look at it in its full generality**

# Reminder

- ◆ An analysis (ignore subscripts for now...):

$$\langle D_a, \sqcup_a, \sqcap_a, \sqsubseteq_a, \top_a, \perp_a, F_a \rangle$$

- ◆  $F_a$  is global flow function, which takes a map from edges to dataflow information

- ◆ Solution is  $\bigsqcup_{i=0}^{\infty} F_a^i(\perp_a)$

# Const Prop Example

```
x := 0;           ← {x=0}
y := 0;           ← {x=0, y=0}
while (...) {
  x := x + 1;     ← {x=1, y=0}
  print (x);
}
print (y);        ← {y=0}
```

- ◆ What do elements of the constant propagation lattice really mean?
  - So far they are just symbols with informal meaning

# Twist on Const Prop

- ◆ Stop merging maps at CF merge points
- ◆ Rather, take union of maps

```
x := 0;
y := 0;
while (...) {
    x := x + 1;
    print (x);
}
print (y);
```

← { {x=0, y=0}, {x=1, y=0}, ... }

# What's Going On Here?

- ◆ **This new analysis...**
  - **Produces much more information**
  - **No longer terminates**
  - **Is a lot like just running the program**

◆ **As we have seen...**

◆ **Semantics of a programming language**

- **Captures how instructions of a programming language operate**

◆ **Semantics of a program**

- **Instantiates the language semantics to capture how a given program runs**
- **This is what we're interested in today**

# Semantics of a Program

- ◆ Fixed points can be used to capture the semantics of a program:

$$\langle D_c, \sqcup_c, \sqcap_c, \sqsubseteq_c, \top_c, \perp_c, F_c \rangle$$

- ◆ Solution:  $\bigsqcup_{i=0}^{\infty} F_c^i(\perp_c)$

- ◆ Note: Here we're talking about the meaning of the actual program
  - This isn't about program analysis

# Semantics of a Program

- ◆ **Back to constant propagation example**
- ◆ **What were we computing?**
  - **Set of all possible program states at a given CFG edge**
  - **But notice that we took a limited view of “program state”**
    - **Our facts were only about constant values**
- ◆ **This fixed point is not computable**
  - **No problem: it’s a strawman**
  - **We never compute it, only reason about it**

# Abstract Interpretation

- ◆ An abstract interpretation  $I$  is a tuple:

$$I = \langle D, \sqcup, \sqcap, \sqsubseteq, \top, \perp, F \rangle$$

- ◆ Solution:  $\bigsqcup_{i=0}^{\infty} F^i(\perp)$

- ◆  $F$  is the global flow function

- ◆  $D$  is the global domain

- Most of the time  $D$  will contain maps from edges to dataflow information

# Concrete vs. Abstract

- ◆ “Concrete” interpretation
- ◆ “Running” program in concrete domain
- ◆ Generally incomputable
- ◆ “Abstract” interpretation
- ◆ “Running” program in abstract domain
- ◆ Generally computable
  - But not necessarily fast

$$\langle D_c, \sqcup_c, \sqcap_c, \sqsubseteq_c, \top_c, \perp_c, F_c \rangle$$

$$\bigsqcup_{i=0}^{\infty} F_c^i(\perp_c)$$

$$\supseteq \langle D_a, \sqcup_a, \sqcap_a, \sqsubseteq_a, \top_a, \perp_a, F_a \rangle$$

$$\bigsqcup_{i=0}^{\infty} F_a^i(\perp_a)$$

# Concrete vs. Abstract

- ◆ **Even our “concrete” constant propagation is abstract**
  - For example, it abstracts away information about how a program point was reached
- ◆ **So the terminology here is confusing**
  - Key is that “concrete” and “abstract” are relative, not absolute
- ◆ **All interpretations are in some sense abstract**
  - In other words there is no “most concrete” interpretation

# Collecting Semantics

- ◆ **Collecting semantics computes set of observable program behaviors in the operational semantics**
  - We'll assume behaviors are partitioned by program points
- ◆ **A collecting semantics is the starting point for designing a program analysis**
  - **Collecting semantics can be generated from other forms of language semantics without too much difficulty**
    - **At least in principle – as far as I know nobody has tried to do this for Norrish's semantics**
  - **There are many different collecting semantics**
- ◆ **We saw a collecting semantics in the constant propagation example**

# More Collecting Semantics

- ◆ **More concrete than collecting semantics:**
  - **Trace prefix semantics**
    - **Compute at edge  $e$  the set of all program traces that reach  $e$**
- ◆ **Less concrete than trace semantics:**
  - **Input-output semantics**
    - **Compute set of input-output pairs**
- ◆ **Even more concrete than trace semantics:**
  - **Full trace semantics**
    - **Collect set of all traces**
- ◆ **Even more concrete?**

# Summary

- ◆ **There are many ways to interpret a program**
  - **All are abstract to some degree**
  - **Can think of them as forming a lattice with partial order operator “is more abstract than”**

- ◆ **What is the top element of the lattice of abstract interpretations?**
  - **I.e., the interpretation that is more abstract than all others?**
  
- ◆ **Why is it desirable for the formal semantics of a language to be abstract?**
  
- ◆ **Why is a concrete interpretation inherently easier to think about than an abstract interpretation?**

- ◆ **Choosing the right concrete semantics is important**
  - **Affects what you can prove about the program**
- ◆ **Too concrete → Clumsy, complicated proofs**
  - **Example?**
- ◆ **Too abstract → Can't prove properties of interest**
  - **Example?**
  
- ◆ **But the key is that all can be expressed as fixed point computations**

# Back to Correctness

- ◆ We have two fixed point computations  $I_c$  and  $I_a$
- ◆  $I_c$  : the “concrete” semantics of our program
  - Easy to understand because it closely mirrors actual execution
  - But we can’t compute this fixpoint
- ◆  $I_a$  : the abstract semantics
  - $I_a$  is computable, but... is it meaningful?
  - In other words does  $I_a$  tell us something about  $I_c$ ?
- ◆ We want to show that the abstract fixed point approximates the concrete one

# Formally

$$I_c = \langle D_c, \sqcup_c, \sqcap_c, \sqsubseteq_c, \top_c, \perp_c, F_c \rangle \quad I_a = \langle D_a, \sqcup_a, \sqcap_a, \sqsubseteq_a, \top_a, \perp_a, F_a \rangle$$

$$\bigsqcup_{i=0}^{\infty} F_c^i(\perp_c) \quad \longleftrightarrow \quad \bigsqcup_{i=0}^{\infty} F_a^i(\perp_a)$$

- ◆ **Formalize relation between the two fixed points using two functions**
  - **Abstraction function  $\alpha$**
  - **Concretization function  $\gamma$**

# Concretization Function

- ◆  $\gamma: D_a \rightarrow D_c$
- ◆  $\gamma(d_a)$  returns concrete information that  $d_a$  approximates
- ◆ For constant propagation in a program with a single variable
  - $\gamma(C) = \{ \{x=C\} \}$
  - $\gamma(\perp) = \{ \}$
  - $\gamma(\top) = \{ \dots, \{x=-2\}, \{x=-1\}, \{x=0\}, \{x=1\}, \{x=2\}, \dots \}$
- ◆ This is key – must understand it before we proceed
- ◆ Note that typically there are many valid concretization functions
  - Want the most precise one

# Approximation

- ◆ **Formally:  $d_a$  approximates  $d_c$  iff:  $d_c \sqsubseteq_c \gamma(d_a)$**
- ◆ **Assume that at a given edge  $e$ , the dataflow info says that  $x$  is 4**
  - **I.e.  $d_a(e) = \{ \{x=4\} \}$**
- ◆ **Now assume that  $d_a$  approximates  $d_c$** 
  - **I.e.  $d_c \sqsubseteq_c \gamma(d_a)$**
- ◆ **From  $d_c \sqsubseteq_c \gamma(d_a)$ , using the definition of  $\sqsubseteq_c$ , we get:**
  - **$d_c(e) \subseteq \gamma(d_a)(e)$**
- ◆ **From  $d_a(e) = \{ \{x=4\} \}$  and defn of  $\gamma$ , we get:**
  - **$\gamma(d_a)(e)$  is the set of all program states where  $x$  is 4**
- ◆ **So what does  $d_c(e) \subseteq \gamma(d_a)(e)$  say?**
  - **$x$  evaluates to 4 in all program states at  $e$**

# Fixpoint Approximation

- ◆ Goal is to show that:

$$\bigsqcup_{i=0}^{\infty} F_c^i(\perp_c) \sqsubseteq_c \gamma \left( \bigsqcup_{i=0}^{\infty} F_a^i(\perp_a) \right)$$

- ◆ First, let's see  $\alpha$ , the abstraction function

# Abstraction Function

- ◆  $\alpha: D_c \rightarrow D_a$
- ◆  $\alpha(d_c)$  returns abstract information that approximates a concrete program state  $d_c$
- ◆ Formally:  $d_a$  approximates  $d_c$  iff:  $\alpha(d_c) \sqsubseteq_a d_a$
- ◆ For constant propagation in a single variable  $x$ 
  - $\alpha(\{\}) = \perp$
  - $\alpha(\{ \{x=C\}, \{x=C\}, \dots \}) = C$ 
    - I.e.,  $x$  has the same value in all program states at this CF edge
  - $\top$  otherwise
- ◆ Typically there are many valid abstraction functions
  - Want the most precise one

# Concretization and Abstraction

- ◆ **Note that abstraction followed by concretization is often lossy, e.g.**

- $\alpha(\{ \{x=4\}, \{x=5\} \}) = \top$
- $\gamma(\top) = \{ \dots, \{x=-1\}, \{x=0\}, \{x=1\}, \dots \}$

- ◆ **This information loss is necessary**

- **Without it the abstract interpretation would be just as concrete as the concrete interpretation**

- ◆ **Concretization followed by abstraction can also be lossy**

- **But it isn't for our version of constant propagation**
- **In general it doesn't need to be (shouldn't be?)**

# Concretization and Abstraction

- ◆ **What are  $\alpha$  and  $\gamma$  for**
  - **Parity domain?**
  - **Signedness domain?**
  - **Interval domain?**
  - **Bitwise domain?**
  
- ◆ **Note: Constants, intervals, bitwise, parity, signedness are all non-relational abstract domains**
  - **Variables are treated in isolation**
  - **Relational domains show how variables relate**
    - **E.g. polyhedra:  $Ax + By + Cz < W$**

# Galois Connection

◆  $\alpha$  and  $\gamma$  establish a Galois connection between two lattices if and only if

- $\gamma$  and  $\alpha$  are monotone functions
- $\forall x \in D_c : x \sqsubseteq_c \gamma \circ \alpha(x)$
- $\forall y \in D_a : \alpha \circ \gamma(y) \sqsubseteq_a y$

◆ This is the basic requirement

◆ Alternately:

- $\forall x \in D_c \forall y \in D_a : \alpha(x) \sqsubseteq_a y \text{ iff } x \sqsubseteq_c \gamma(y)$

◆ Many times it will be the case that

- $\forall y \in D_a : \alpha \circ \gamma(y) = y$

# Summary

- ◆ **Start with a concrete domain – the collecting semantics**
  - **We must already have confidence that this domain captures the semantics of the language we are analyzing**
    - **Ideally because it was derived mathematically from the language's operational semantics**
- ◆ **Design an abstract domain**
  - **Our goal: Gain confidence that a fixpoint in this domain always approximates a fixpoint in the concrete semantics**
- ◆ **Relate the domains using a Galois connection**
- ◆ **This is a good start**
  - **But there's plenty more... we haven't even talked about transfer functions yet**