

Last Time

◆ Abstract interpretation

- Dual of dataflow analysis
- Dataflow view emphasizes implementation issues
- Abstract interpretation view emphasizes correctness via connection with actual execution

◆ Abstract interpretation =

- One or more abstract domains +
- Transfer functions +
- Collecting semantics for a PL +
- Concretization and abstraction operators +
- Partitioning scheme +
- Iterative solution algorithm

Today

- ◆ **More abstract interpretation**
 - **Example domains**
 - **Widening**
 - **Combining domains**

Domains

- ◆ **We already looked at domains such as**
 - **Constants**
 - **Intervals**
 - **Bitwise**
- ◆ **These are all non-relational domains**
 - **Treat variables in isolation**
 - **These are fine for supporting optimizations**
 - **However: Surprisingly weak for computing stronger program properties such as safety invariants**
- ◆ **Relational domains are much more powerful**
 - **Unsurprisingly, these track relations among variables**

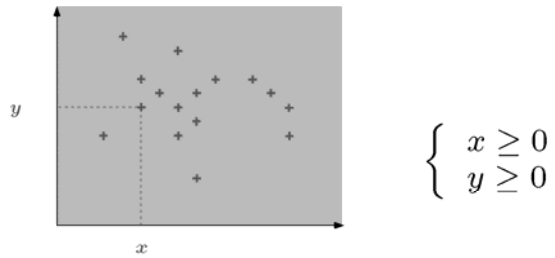


Fig. 2
SIGNS

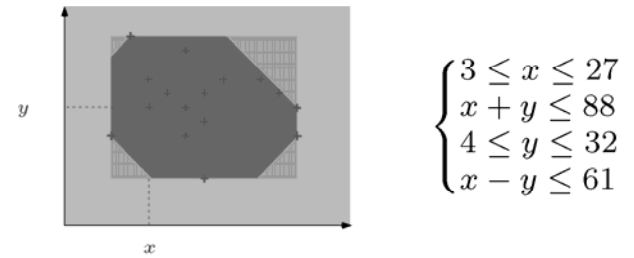


Fig. 4
OCTAGONS

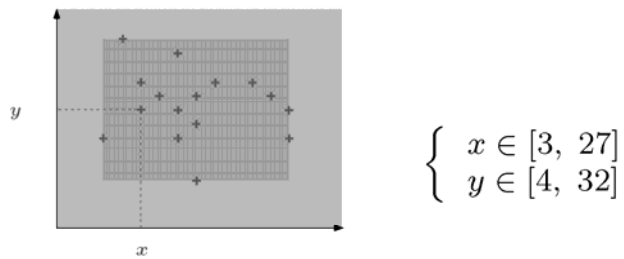


Fig. 3
INTERVALS

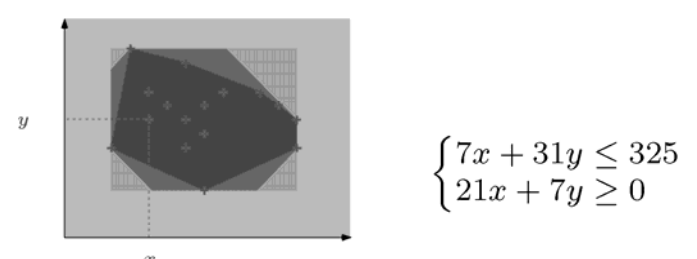


Fig. 5
POLYHEDRA

Figure from Cousot 2000

Relational Numerical Domains

- ◆ **Transfer functions are usually expensive**
 - **Polyhedra: Exponential**
 - **Octagons: Cubic**
 - **C.f. intervals: Linear**
 - **How to deal with this in practice?**
- ◆ **Are unsound in presence of overflow / underflow**
 - **When does $x > y$ imply that $(x+1) > y$?**
 - **How to deal with this in practice?**
- ◆ **Good transfer functions are available: APRON library**
 - **C and OCaml interfaces supported**
 - **FP and integer types supported**

Decision Tree Abstract Domain

- ◆ **Non-numerical relational domain developed for Astree analyzer**
 - **Astree verified the absence of runtime errors in Airbus A340 fly by wire software**
- ◆ **Abstract value relates Booleans with a numerical abstract domain**
 - **Doesn't matter which**
- ◆ **E.g.**
 - **$b \rightarrow a=[1..20], \sim b \rightarrow a=[0..19]$**
- ◆ **Goal: Add some path sensitivity**
 - **Increases analysis precision**
 - **Exponential cost**

Widening

- ◆ **Problem: Abstract domains with large or infinite height result in analysis that is slow or non-terminating**
- ◆ **I.e. for some domains we can't (efficiently) compute:**

$$fix = \bigsqcup F^i(\perp, \dots, \perp)$$

- ◆ **Solution: Introduce a widening operator w that composes with the transfer function like this:**

$$fixw = \bigsqcup (F \circ w)^i(\perp, \dots, \perp)$$

Widening

- ◆ **As long as w is monotonic, the analysis remains sound**
- ◆ **Example widening function for intervals:**
 - **For some fixed set of constants, find the smallest interval with bounds chosen from this set that contains starting interval**
 - **For example, for the set $\{-\infty, -10, -1, 1, 10, \infty\}$**
 - **$[0..1]$ goes to $[-1..1]$**
 - **$[-5..0]$ goes to $[-10..1]$**
 - **$[-15..11]$ goes to $[-\infty.. \infty]$**
 - **Set could be found by looking for constants in the source code**

Widening Example

- ◆ Let's say we want to analyze this code using the interval domain

```
y=0;  
while (opaque) {  
    x=7;  
    x++;  
    y++;  
}
```

- ◆ Interval analysis does not terminate (quickly)
 - Result is $x=[8..8]$, $y=[0..\infty]$

Widening Example

```
y=0;  
while (opaque) {  
    x=7;  
    x=x+1;  
    y=y+1;  
}
```

- ◆ **Widening with the set of constants $\{-\infty, 0, 1, 7, \infty\}$**
 - **Result is $x=[7..\infty]$, $y=[0..\infty]$**

Widening Discussion

- ◆ **Why use widening when we could just define a rapidly-terminating lattice ahead of time?**
 - **Widening effectively permits a new sub-lattice to be picked for each program analyzed**
 - **Avoids messing with the original, clean domain**
 - **Narrowing – can potentially recover some precision by running the original transfer functions after widening has overshot the fixpoint**

Combining Abstract Domains

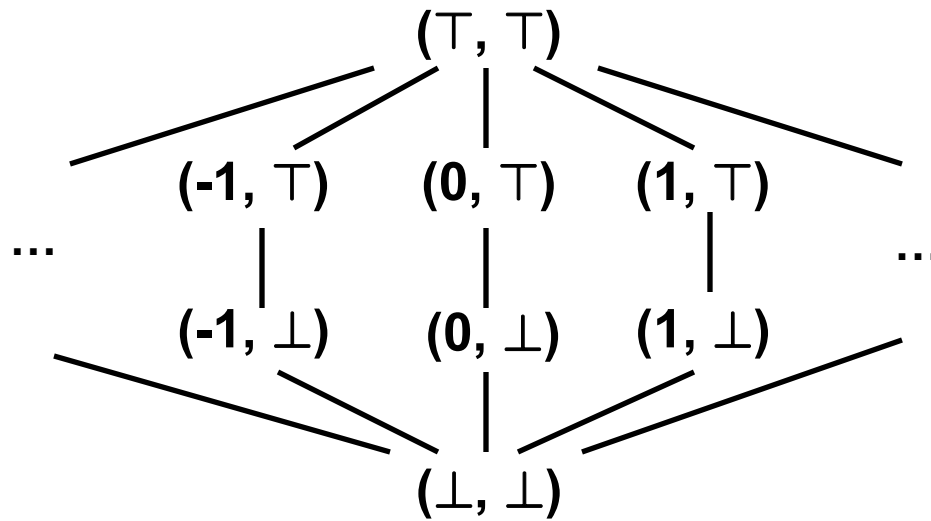
- ◆ **Motivating example program:**

```
x = 1;
while (opaque) {
    if (x != 1) {
        x = 2;
    }
}
```

- ◆ **First analyze using constant propagation**
- ◆ **Second perform liveness analysis**
- ◆ **Will repeating the analyses help?**
- ◆ **What is going on here?**

Domain Combination 1

- ◆ **Compute the direct product of the constant and liveness lattices**
 - Lattice is cross product
 - Transfer functions are created by applying original functions element-wise
- ◆ **For programs with one variable:**



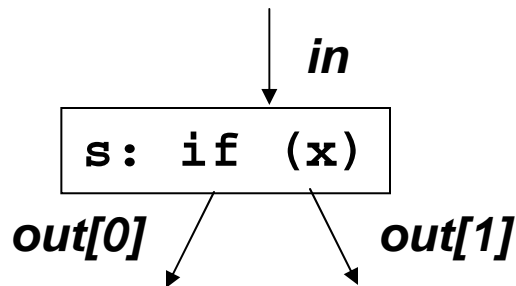
Domain Combination 1

- ◆ Now analyze using the combined domain

```
x = 1;
while (opaque) {
    if (x != 1) {
        x = 2;
    }
}
```

- ◆ What happens?
- ◆ What's going on here?

Domain Combination 1



For constants:

$$out[0] = in \wedge$$

$$out[1] = in$$

For liveness:

$$out[0] = \perp \text{ if } x \neq 0 \text{ or } in = \perp$$

\wedge

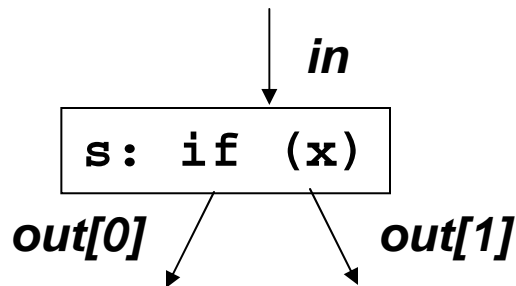
$$out[1] = \perp \text{ if } x = 0 \text{ or } in = \perp$$

Here 0 on the right side is literal zero

Domain Combination 2

- ◆ **Compute the reduced product of the constant and liveness lattices**
 - **Lattice is cross product**
 - **Transfer functions take optimal advantage of the interacting domains**
- ◆ **In this case we can improve one of the transfer functions**
- ◆ **Result is called “conditional constant propagation”**
 - **CCP paper is great**

Domain Combination 2



For constants:

$$out[0] = in \wedge$$

$$out[1] = in$$

For liveness:

$$out[0] = \perp \text{ if } x \neq 0 \text{ or } in = \perp$$

\wedge

$$out[1] = \perp \text{ if } x = 0 \text{ or } in = \perp$$

Here 0 on the right side is an element of the constant lattice

Back to Example

```
x = 1;
while (opaque) {
    if (x != 1) {
        x = 2;
    }
}
```

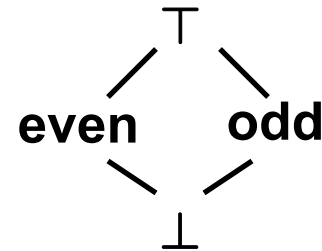
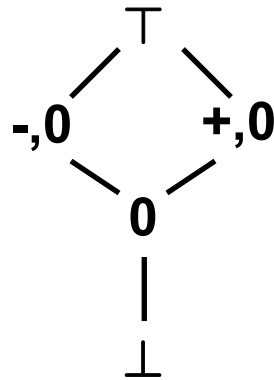
◆ Finally we get the desired result

Reduced Product

- ◆ **Recall the requirement for local transfer function F_A :**
 - $\alpha \circ F_C \circ \gamma \sqsubseteq_A F_A$
- ◆ **In other words, the optimal transfer function is obtained by:**
 - **Concretizing the abstract value(s)**
 - **Applying the concrete transfer function**
 - **Abstracting the resulting set of concrete values**
- ◆ **Similarly, the reduced product may be computed by:**
 - **Concretizing the abstract values in all domains**
 - **Applying the concrete transfer functions**
 - **Intersecting the resulting sets of concrete values**
 - **Abstracting the intersection**
- ◆ **Clearly not (efficiently) computable**

Another Example

- ◆ Consider lattices of signs, parity, and constants



- ◆ Exercise: Give a nontrivial transfer function from the reduced product domain
 - $(+,0, \text{odd}, *) - (*, *, 1) =$
 - $(+,0, \text{even}, \top)$

Another Example

```
a = 5;  
b = 10;  
p = &a;  
while (*p > 5) {  
    p = &b;  
    b = 20;  
}
```

- ◆ Does CCP do the trick here?
- ◆ What needs to be added?

Domain Combination

- ◆ **Different domains discover different information about a program**
- ◆ **Combining domains permits them to “learn” from each other**
- ◆ **In practice**
 - **Combining domains is hard**
 - **Only do it for a few important special cases like CCP**

Summary

- ◆ **Lots of abstract domains exist, trading off**
 - **Expressive power**
 - **Time and space efficiency**
 - **Transfer function complexity**
- ◆ **Widening is a way to reach the fixpoint faster**
- ◆ **Combining abstract domains provides a powerful way to design program analyses**
 - **Con: Domain combination isn't easy**
 - **Pro: May still be easier than designing the composite domain from scratch**