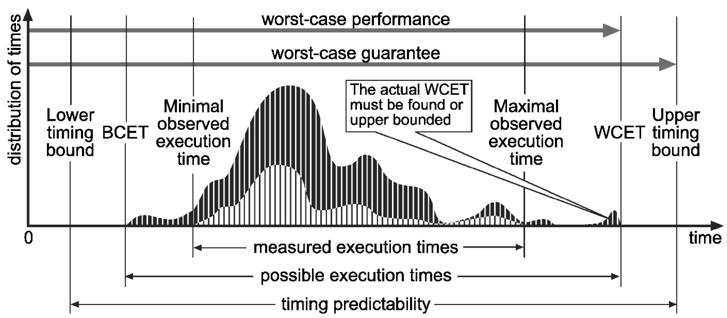
Physical Timing Constraints and SW Execution Time

Recall the loop behavior of most reactive programs:

```
init
while true loop
      -- point (a)
      get inputs
      compute
      emit outputs
end loop
```

Execution of the loop body = input sampling rate

Variations of Execution Time



From: The worst-case execution-time problem. Overview of methods and survey of tools.

ACM Transactions on Embedded Computing Systems (TECS). Volume 7 Issue 3, April 2008, Article No. 36

Static Evaluation of the Execution Time

WCET = Worst-Case-Execution-Time, BCET, ...

Real-time guarantees require a precise estimation of the software execution time, statically (before putting the software into operation).

Problems:

- Details on the execution platform (the hardware, the OS, the network, ...) that may influence the execution time
- Even for a single isolated machine with an old-style processor (no pipeline, no cache): the execution time of a program depends on its flow of control, that cannot be known statically.

Structural Analysis for a Basic Execution Platform

- WCET of each instruction (WCET=BCET=the time it takes for this instruction)
- Sequence : sum
- IF-THEN-ELSE : `max`
- FOR loops : multiplication
- General WHILE loops : need an estimation of the number of executions

Examples with approximations

```
for i in 1..100 loop
    if i mod 2 = 0 then
         // cost s
    else
         // cost B
end loop ;
```

Max cost (easy to find, pessimistic) = 100 * max (s, B) = 100 * B

More precise (but hard to compute) = 50 * (B + s)

Modern Architectures

Even for a single core:

- Pipeline :
  WCET (Inst1 ; Inst2) $\neq$ WCET (Inst1) + WCET (Inst2)
- Cache :
  WCET (Inst) depends on the state of the cache
  Secure but pessimistic hypothesis: each access to the cache is a cache-miss.
  Less pessimistic: try and find states in which some accesses to the cache are guaranteed to be cache-hits.

Even worse: timing anomalies (there exist architectures for which a cache hit may be longer than a cache miss, in some states); multicore systems with asynchronous networks-on-chip; ....

A Survey

The worst-case execution-time problem. Overview of methods and survey of tools.
ACM Transactions on Embedded Computing Systems (TECS) Volume 7 Issue 3, April 2008, Article No. 36
Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner Jan Staschulat, Per Stenström.

Complex Architectures and Timing

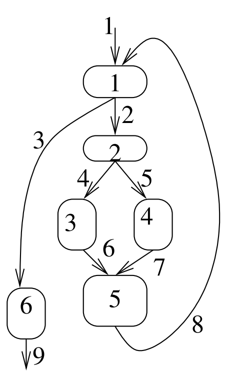
Hopeless?

For critical systems we need:
— Determinism (reproducibility)
— Predictability of timing

Open question: Is it possible to find a way to use complex architectures (e.g, multicore platforms) that makes timing predictable, even at the price of not exploiting the full power of these architectures?

CFG and Basic Blocks

A Basic Block (BB) is a piece of machine code with a single entry point and a single exit point, with no branching in-between. The Control-Flow Graph (CFG) is a graph where the nodes are the BBs, and the edges are the branches.



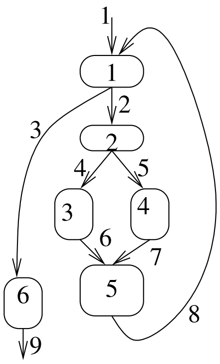
Kirchhoff's circuit laws for CFGs

Consider a BB with en_1, en_2, ..., en_k as incoming edges, and ex_1, ex_2, ..., ex_ℓ as ougoing edges.

Associate variables with: the edges, and the BB itself. Express a constraint on the number of times the edges are taken, or the BB executed:

$$bb = \sum_{i=1}^{i=k} en_i = \sum_{i=1}^{i=\ell} ex_i$$

Kirchhoff's circuit laws for CFGs - Example



$edge1 = 1$
$bb1 = edge1 + edge8 = edge2 + edge3$
$bb6 = edge3 = edge9$
$edge9 = 1$
$bb2 = edge2 = edge4 + edge5$
$bb3 = edge4 = edge6$
$bb4 = edge5 = edge7$
$bb5 = edge6 + edge7 = edge8$
Loop bound, let's say:
$edge8 \leq 100$

Maximize:

$$\sum_{i=1}^{i=6} \text{wcet}(BB_i) \times bb_i$$

Solving the Kirchhoff's circuit laws for CFGs

There are tools to find the maximum value of:

$$\sum_{i=1}^{i=6} \text{wcet}(BB_i) \times bb_i$$

(and the individual values of the bb_is)

Example:
http://lpsolve.sourceforge.net/5.5/

ILP Problem and Tools - Example

Assume wcet(bb1)=4; wcet(bb2)=3; wcet(bb3)=1; wcet(bb4) = 12; wcet(bb5) = 5; wcet(bb6) = 1;
File `wcetinputlpsolve.txt` :

```
max:  4 bb1 + 3 bb2 + bb3 + 12 bb4 + 5 bb5 + 2 bb6;
edge1 = 1 ;
bb1 = edge1 + edge8 ; bb1 = edge2 + edge3 ;
bb6 = edge3 ; bb6 = edge9 ;
edge9 = 1 ;
bb2 = edge2 ; bb2 = edge4+edge5 ;
bb3 = edge4 ; bb3 = edge6 ;
bb4 = edge5 ; bb4 = edge7 ;
bb5 = edge6+edge7 ; bb5 = edge8 ;
edge8 <= 100 ;
```

Command: `lp_solve wcetinputlpsolve.txt`

ILP Problem and Tools - Result

```
Value of objective function: 2406
Actual values of the variables:
bb1                        101
bb2                        100
bb3                          0
bb4                        100
bb5                        100
bb6                          1
edge1                        1
edge8                      100
edge2                      100
edge3                        1
edge9                        1
edge4                        0
edge5                      100
edge6                        0
edge7                      100
```


Additional Problems

- Where to find the constants wcet(BB_i)?
  (This is where you need to know the behavior of the HW, see later)
- Are the wcet(BB_i) independent of the path in the CGF?
- How to take into account infeasible paths?

Infeasible Paths: Example 1

```
...
if (condition (x, y)) {
   some code A
} else {
   some code B
}
if (!condition (x, y)) {
   some code C
} else {
   some code D
}
```

The portions of code A, B, C, D DO NOT modify x, y.

The path that executes A and then C is infeasible. Same for B and then D.

If these paths have an execution time higher than that of A;D and B;C, then removing them improves the WCET.

Infeasible Paths: Example 2

```
for i in 1..100 loop
    if i mod 2 = 0 then
         // cost s
    else
         // cost B
end loop ;
```

If we unroll the loop, we obtain a sequence of 100 instances of the same block of code. How many paths in this unrolled code? How many of them are infeasible? Note that, contrary to the previous example, the execution of the loop body DOES MODIFY the value of i which is used in the test.

Infeasible Paths: Summary

- Some simple cases can be encoded into the ILP problem. Try and encode the previous case...
- In general, need information on the values of the variables at each program point, hence problem similar to software verification.
- ...

ILP Problem - Example Infeasible Path

Assume wcet(bb1)=4; wcet(bb2)=3; wcet(bb3)=1; wcet(bb4) = 12; wcet(bb5) = 5; wcet(bb6) = 1;
File `wcetinputlpsolve.txt` :

```
max:  4 bb1 + 3 bb2 + bb3 + 12 bb4 + 5 bb5 + 2 bb6;
edge1 = 1 ;
bb1 = edge1 + edge8 ; bb1 = edge2 + edge3 ;
bb6 = edge3 ; bb6 = edge9 ;
edge9 = 1 ;
bb2 = edge2 ; bb2 = edge4+edge5 ;
bb3 = edge4 ; bb3 = edge6 ;
bb4 = edge5 ; bb4 = edge7 ;
bb5 = edge6+edge7 ; bb5 = edge8 ;
edge8 <= 100 ; 2 bb4 <= bb2 ;
```

Command: `lp_solve wcetinputlpsolve.txt`

ILP Problem and Tools - Result

```
Value of objective function: 1856
Actual values of the variables:
bb1                          101
bb2                          100
bb3                           50
bb4                           50
bb5                          100
bb6                            1
edge1                          1
edge8                        100
edge2                        100
edge3                          1
edge9                          1
edge4                         50
edge5                         50
edge6                         50
edge7                         50
```

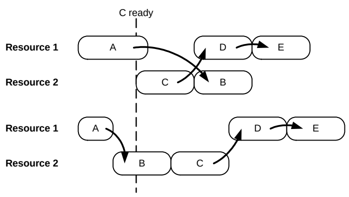

Critical Embedded Code: An Interesting Case

- For critical embedded code, guaranteeing the WCET is very important; users can spend significant time and money for that.
- We need to compute the WCET of the body of the control loop. It does not contain while loops.

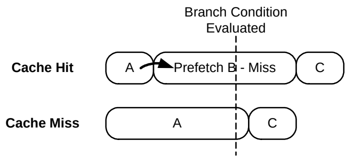
The control flow analysis is “reduced” to the problem of identifying infeasible paths;
Modeling the hardware is still compulsory.

Before We Start: Timing anomalies

- When local worst-case does not lead to the global worst-case



Scheduling anomaly.



Speculation anomaly.

Classification of architectures

- *Timing compositional*
  - No timing anomalies
  - e. g., ARM7
- *Compositional with bounded effects*
  - Timing anomalies but no domino effects
  - e. g., TriCore (probably)
- *Non-compositional architectures*
  - Timing anomalies, domino effects
  - e. g., PPC 755

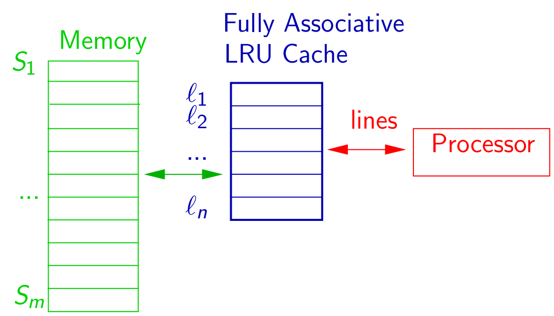
from Wilhelm et al.: Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-critical Embedded Systems, IEEE TCAD, July 2009

The Cache Example (Simple Cases)

- No Timing anomalies
- LRU replacement policy (Least Recently Used)
- Fully associative caches
  (any line of the memory can go in any line of the cache)

*See: Applying Compiler Techniques to Cache Behavior Prediction
Christian Ferdinand and Florian Martin and Reinhard Wilhelm
Proc. ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems, pp. 37-46. 1997*

Definition of the cache



When the cache is full, and the processor needs a line of the memory that is not in the cache, one line has to be evicted. LRU = the least recently used is evicted and replaced by the new one.

Concrete States of the Cache

A concrete cache state is a function: $c : L \longrightarrow S \cup \{I\}$
where:
L is the set of lines of the cache (the positions)
S is the set of lines of the memory
I is a special element to denote an empty line in the cache.

Encoding the age of a cache line: by the position (the most recent is at index 1). *In the real HW, there's an array of pointers, the cache lines are never moved!*

Question: how many concrete states?

Evolution of the Concrete State

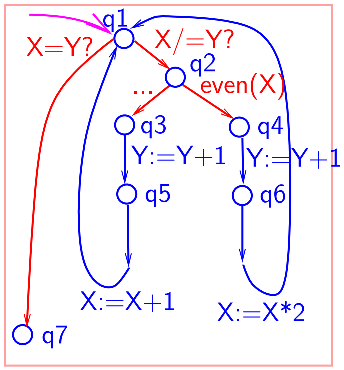
When the processor needs a line S_x of the memory, the concrete cache state is updated, depending on whether s_x is already in the cache.

$$\mathcal{U}(c, s_x) = \begin{cases} (l_1 \mapsto s_x, \\ l_i \mapsto c(l_{i-1}) \mid i = 2..h, \\ l_i \mapsto c(l_i) \mid i = h+1..n); \\ \qquad\qquad \text{if } \exists l_h : c(l_h) = s_x \\ (l_1 \mapsto s_x, \\ l_i \mapsto c(l_{i-1}) \mid i = 2..n); \\ \qquad\qquad \text{otherwise.} \end{cases}$$

the order in the cache represents the age of the cache elements: $c(l_1)$ is the most recently used element.

Detailed Control Flow Graph

```
declare
  X : positive := 2 ;
  Y : positive := 10 ;
begin
  while X /= Y loop
     if even(X)  then
        Y := Y+1 ;
        X := X*2 ;
     else
        Y := Y+1 ;
        X := X+1 ;
     end if ;
  end loop ;
end
```



Detailed Control Flow Graph with Memory Accesses

Each transition is labeled by a set of lines, corresponding to the data that are accessed by the operations of the transition.

Example: for a transition `x > 3` (resp. `y++`) in the original detailed control flow graph, we build a transition $\{l_x\}$ (resp. $\{l_y\}$) in the graph to be analysed.
l_x (resp. l_y) is the line in the memory where variable `x` (resp, `y`) has been installed.

Abstract (Must) Analysis

Idea: for each state q of the detailed control flow graph, compute an over-approximation of the cache state, i.e. a set of memory lines that are guaranteed to be in the cache, for any execution that led to q.

Tehn, for any transition t sourced in q that accesses a memory line m, if m is in the cache in q, then count the cost of a cache-hit, otherwise the cost of a cache-miss.

This is conservative: we compute an over-approximation of the execution time (for time-compositional architectures).

Abstract States of the Cache for a "Must" analysis

$c^* : L \longrightarrow 2^S$

$c^*(l_x) = \{s_y, ..., s_z\}$ means the memory blocks $s_y ... s_z$ are in the cache, and will stay in the cache at least during the next $n - x$ references to memory blocks that are not in the cache, or are *older* that $s_y ... s_z$.

As before, the order in c^* represents the age of the memory blocks: older means present in the cache with a greater index.

Abstraction: Example 1

```
access a ;
access b ;
access c ;
access d ;
access e ;
```

Assume the cache has 4 lines. What do we know on the contents of the cache at each step? Is the information exact?

Abstraction: Example 2

```
if x {
  access u
} else {
  access v
}
```

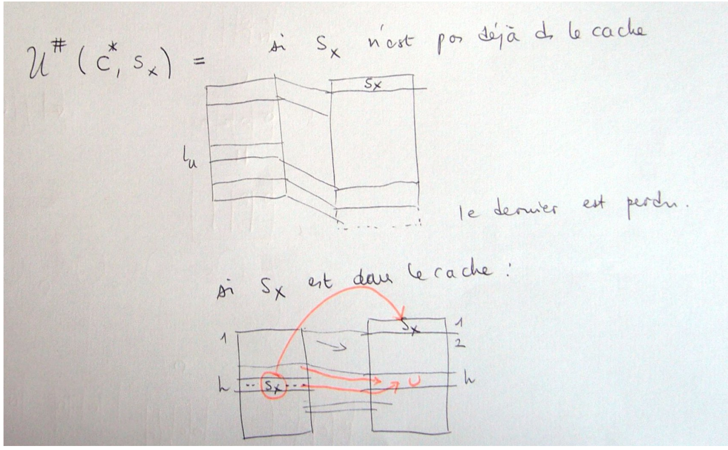
Assume the cache has 4 lines. What do we know on the contents of the cache at the end of the IF statement? Is the information exact?

Updating the Abstract Cache State for One Access to S_x

$$\hat{c}' = \begin{cases} [l_1 \mapsto \{s_x\}, \\ \ l_i \mapsto \hat{c}(l_{i-1}) \mid i = 2 \dots h-1, \\ \ l_h \mapsto \hat{c}(l_{h-1}) \cup (\hat{c}(l_h) - \{s_x\}), \\ \ l_i \mapsto \hat{c}(l_i) \mid i = h+1 \dots n]; \\ \qquad\qquad\qquad\qquad \text{if } \exists l_h : s_x \in \hat{c}(l_h) \\ [l_1 \mapsto \{s_x\}, l_i \mapsto \hat{c}(l_{i-1}) \text{ for } i = 2 \dots n]; \\ \qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

From: Applying Compiler Techniques to Cache Behavior Prediction, op.cit.

Updating the Abstract Cache State for One Access to S_x



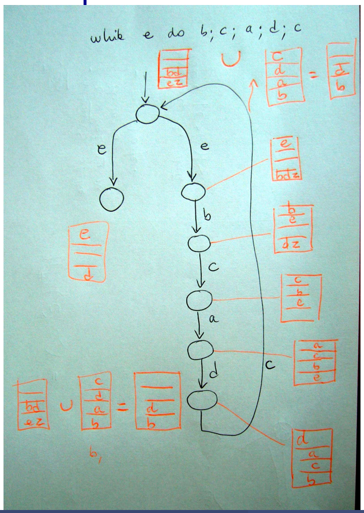
Joining Abstract Cache States

At the end of an IF statement (or at the testing point of a loop), several paths converge. The information on the contents of the cache (for a "must" analysis) should reflect what we know at this point, which is true for ANY of the paths that led to this point.

- The set of elements that are in the cache is ... the intersection
- The age of the elements that are in the cache is ... the greatest

Example (again): `if x { access u } else { access v }`

Example



More Details on the Cache Analysis

The general technique is an instance of abstract interpretation. Some more details later.

Collecting Semantics [Floyd67, Hoare69]

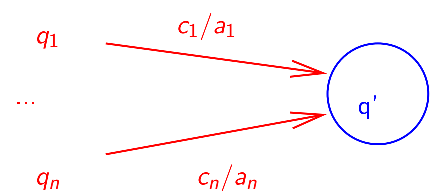
Idea: associate with each control point the set of possible valuations of the variables.
If q is a control point,
We denote by V(q) the set of valuations at this point.

If $q \xrightarrow{c} q'$ then $V(q') = V(q) \cap c$

if $\{q_i \xrightarrow{a_i} q'\}_I$ then $V(q') = \bigcup_I \text{Post}_{a_i}(V(q_i))$

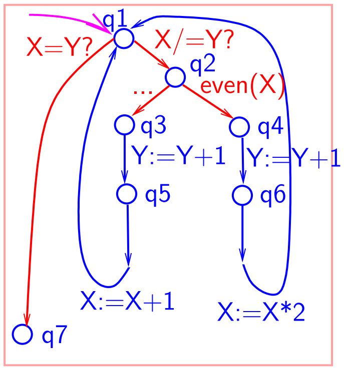
On the compact form

If $\{q_i \xrightarrow{c_i/a_i} q'\}_I$ then $V(q') = \bigcup_I \text{Post}_{a_i}(V(q_i) \wedge c_i)$

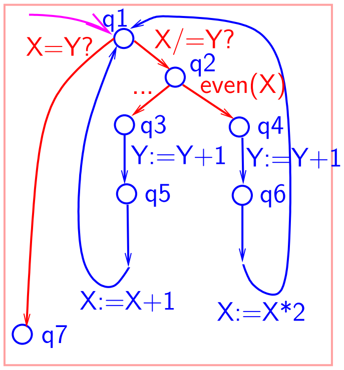


Example

```
declare
  X : positive := 2 ;
  Y : positive := 10 ;
begin
  while X /= Y loop
     if even(X)  then
        Y := Y+1 ;
        X := X*2 ;
     else
        Y := Y+1 ;
        X := X+1 ;
     end if ;
  end loop ;
end
```



Example (cont'd)



$V(q_1) =$
$V_0 \quad \cup$
$\text{Post}_{X:=X+1}(V(q_5)) \quad \cup$
$\text{Post}_{X:=X*2}(V(q_6))$
... Gives a system of fix-point equations

Can we compute the solution of this kind of system?

The system:
$V(q_1) =$
$V_0 \quad \cup$
$\text{Post}_{X:=X+1}(V(q_5)) \quad \cup$
$\text{Post}_{X:=X*2}(V(q_6))$

The solution :
$V(q_0), V(q_1), ..., V(q_n)$

In general no.
We'll use approximate techniques for computing an over-approximation of the fix-point, i.e.,
$V(q_0) \subseteq V'(q_0), ... V(q_n) \subseteq V'(q_n)$

Abstract Interpretation

`http://www.di.ens.fr/~cousot/COUSOTpapers/TSI00.shtml`

`http://www.polyspace.com/`

`http://www.absint.com`

How to compute, for each state of an interpreted automaton, a superset of the possible values of the variables?

Abstract Interpretation Provides Conservative Analyses

If we can build an intepreted automaton (with one variable x), where:

$$q \xrightarrow{c/a} \text{ERROR}$$

And:
— the set of possible values of x in q is $V(q)$
— the condition c is such that $c \cap V(q) = \emptyset$

Then we have proved that the ERROR state cannot be reached from q.

How to install such a technique?

- Find an abstract domain on which it is "easy" and computationally efficient to perform operations like $\cup$, $\cap$ and POST
- Build the interpreted automaton of program $\times$ property ($\times$ environment)
- Compute the solution of the system of equations
- Look at V(ERROR)

The difficult part is to compute the solution of the system of equations, and to justify the conservativity result.
This is the object of the abstract interpretation theory.

The analysis gives conservative results

- The abstract operations and
- the way a solution is built for the system of equations

both compute over-approximations of the sets of values attached to the control points.

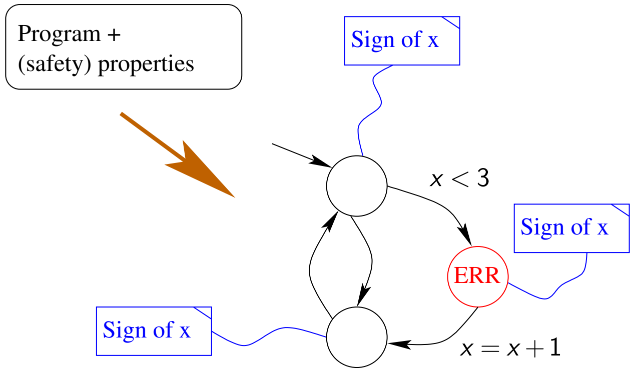
If the set we compute is empty, then the "real" one is also empty = If we declare the property is true, then it is also the case on the real system.

Part IV

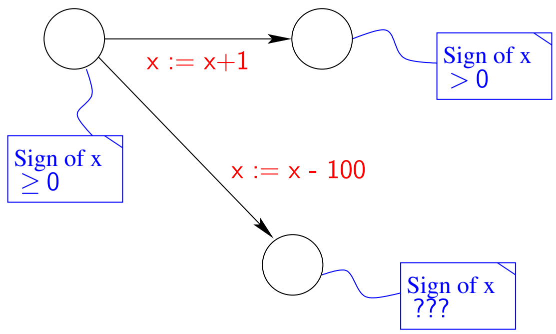
Abstract Interpretation

Outline

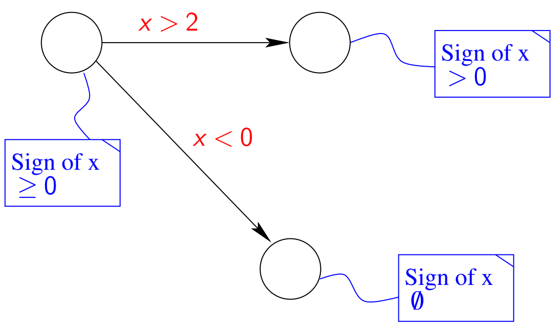
Computing with abstract values



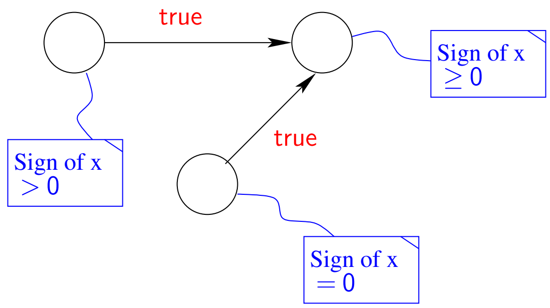
Computing with abstract values: 1) assignments (Post function)



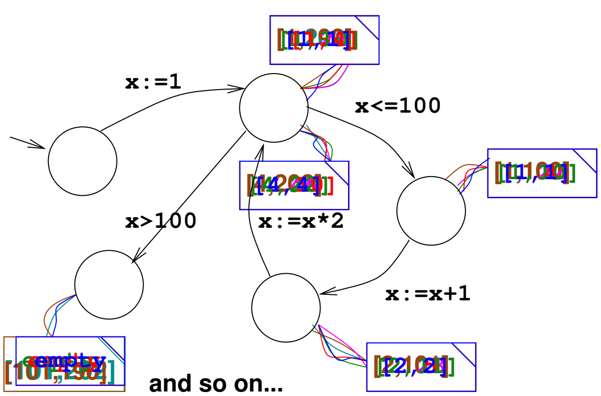
Computing with abstract values: 2) conditions



Computing with abstract values: 3) unions



Computing with abstract values: 4) loops



Main questions

- How to chose an abstract domain (signs, ...)?
- How to propagate the information on states (assignments, conditions, unions)?
- If the information attached to a state is $\emptyset$, what can we deduce?
- What about loops? Does the analysis terminate?

Union of Abstract Information: Ordering Abstract Values

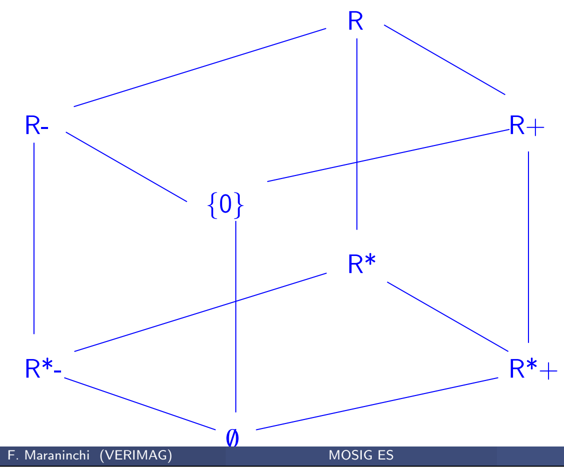
If:

- I know *"this"* coming from the 1st branch
- I know *"that"* coming from the 2nd branch

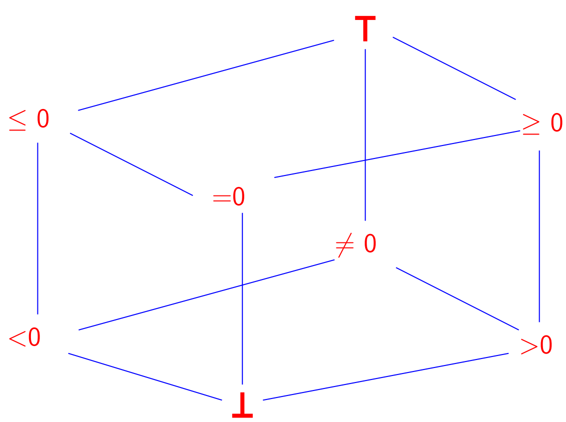
Then I know *"this"* $\cup$ *"that"* at the join point.

What is the union? In general, needs to structure the abstract values into a lattice.

Structuring the Domain of Abstract Values - Signs and subsets of R



Structuring the Domain of Abstract Values - Signs and subsets of R



Signs, Intervals, Convex Polyhedra

$\{ < 0, =0, > 0, \leq 0, \neq 0, \geq 0, ?\ , \perp\}$

INT $= (\mathrm{R} \cup \{ -\infty, +\infty \}) \times (\mathrm{R} \cup \{ -\infty, +\infty \})$

n-dimensions convex polyhedra.
`cf. nicolas-polyedres-a.pdf`

A language of expressions EE

Abstract grammar of EE:

```
E ---> E * E | k | i | abs(E)
```

`k` : integer constants in Z
`i` : identifiers
abs : absolute value

Usual interpretation: value

Function `V : EE ---> Z` :

V (`k`) = k
V (`i`) given elsewhere
V (`E1 * E2`) = V(`E1`) * V(`E2`)
V (`abs(E)`) = abs (V(`E`))

red *, abs are syntactic notations.
blue *, abs are the semantic operations.

Sign interpretation

Function `S : EE ---> { Z, P, N }`

S (k) = if k > 0 then P elsif k < 0 then N else Z
S (i) given elsewhere

S (E1 * E2) =
let s1 = S(E1), s2 = S(E2) in
if s1 = Z or s2 = Z then Z
else if s1=s2 then P else N

S (abs(E)) = if S(E) = Z then Z else P

Where are the abstractions?

The sign interpretation is an abstraction w.r.t. the reference interpretation that computes the value of the expression...
which is, itself, an abstraction w.r.t. the syntactic identity.
`EE ---> Z` values
`EE ---> { Z, P, N }` signs

There exist E1, E2 such that V(E1) $\neq$ V(E2) and S(E1) = S(E2)

A language of expressions: adding +

Abstract grammar of EE:

```
E ---> E * E | E + E | k | i | abs(E)
```

`k` : integer constants in Z
`i` : identifiers

Sign interpretation

Function `S : EE ---> { Z, P, N, ? }`

S (k) = if k > 0 then P elsif k < 0 then N else Z
S (i) given elsewhere

S (E1 * E2) = let s1 = S(E1), s2 = S(E2) in
if s1 = Z or s2 = Z then Z
elsif s1 = ? or s2 = ? then ?
else if s1=s2 then P else N

S (abs(E)) = if S(E) = Z then Z
elsif S(E) = P or S(E) = N then P
else ?

Discussion

S (abs(E)) = ... if S(E) = ? then ? ...

Why ? and not P ????

Because, if S(E) = ? (the sign of E is unknown), then abs(E) may be 0 or positive. Since there is no special value representing this information in the set `{ Z, P, N, ? }`, the only possible (correct) answer is ?.

Abstract Operations

In general:

S (E1 op S2) = opa (S(E1), S(E2))

opa is the abstract operation corresponding to op.

Example : +a

+a	Z	N	P	?
Z	Z	N	P	?
N	N	N	?	?
P	P	?	P	?
?	?	?	?	?

$+a : \{ Z, P, N, ? \} \times \{ Z, P, N, ? \}$
$\longrightarrow \{ Z, P, N, ? \}$

Generalization

We can compute signs with more precision using:
$\{ < 0, =0, > 0, \leq 0, \neq 0, \geq 0, ? \}$

Exercise: write the tables of the abstract operations.

+a for $\{ < 0, =0, > 0, \leq 0, \neq 0, \geq 0, ? \}$

+a	< 0	$=0$	> 0	≤ 0	$\neq 0$	≥ 0	?
< 0		< 0					?
$=0$	< 0	$=0$	> 0	≤ 0	$\neq 0$	≥ 0	?
> 0		> 0					?
≤ 0		≤ 0					?
$\neq 0$		$\neq 0$					?
≥ 0		≥ 0					?
?	?	?	?	?	?	?	?

information loss for "< 0" +a "> 0"

*a for $\{ <0, =0, >0, \leq 0, \neq 0, \geq 0, ? \}$

*a	<0	$=0$	>0	≤ 0	$\neq 0$	≥ 0	?
<0		$=0$					?
$=0$	$=0$	$=0$	$=0$	$=0$	$=0$	$=0$	$=0$
>0		$=0$					?
≤ 0		$=0$					?
$\neq 0$		$=0$					?
≥ 0		$=0$					?
?	?	$=0$	?	?	?	?	?

information gain for "$= 0$" *a ...

More detailed information about signs

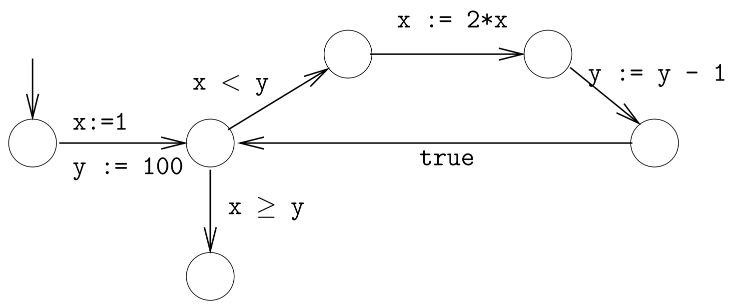
If we also consider the conditional expression:
e = if c then e1 else e2
the sign of e is the "union" of the signs of e1 and e2.
Example :

Sign P, N ou Z	Sign $\{ <0, =0, >0, \leq 0, \neq 0, \geq 0, ? \}$
sign(e1) = P	sign (e1) = >0
sign(e2) = N	sign (e2) = <0
sign(e) = ?	sign (e) = $\neq 0$

A program

```
x := 1 ; y := 100 ;
while x < y loop
      x := 2 * x ;
      y := y - 1 ;
end loop ;
```


The corresponding automaton



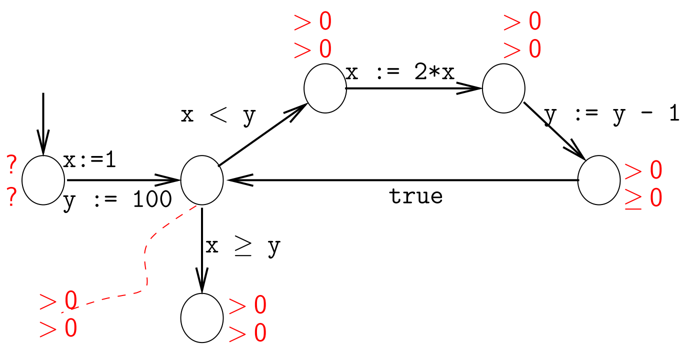
Exercise

Use the abstract domain $\{ <0, =0, >0, \leq 0, \neq 0, \geq 0, ? \}$

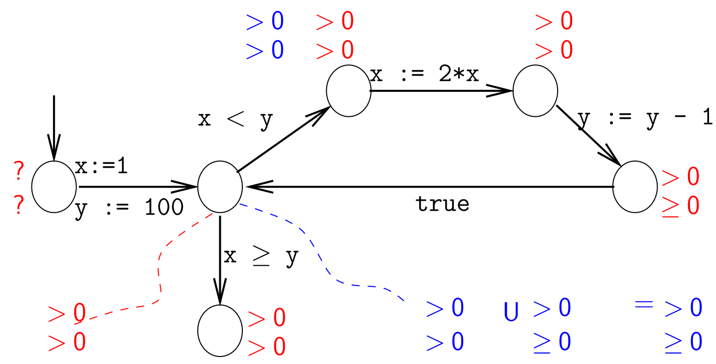
Label the states with the best information you know about the signs of x and y.

— What about the loop ?

Sign information



Sign information (cont'd)



Termination of the analysis

The best sign information we can get on this program has been obtained in two steps.

- The set of abstract values is finite
- The set of values attached to each state may only grow, from one step to the next step.

Does it help?

For instance to determine whether the program stops...

No, because there is no information about the relationship between `x` and `y`.
We would need something more powerful.

Intervals

the domain:

INT = (R ∪ { - ∞, + ∞ }) × (R ∪ { - ∞, + ∞ })

an interval is denoted by: [x, y]

Interval interpretation on a language of expressions

int : EE $\longrightarrow$ INT

Example for + : int (E1 + E2) = +a (int (E1), int (E2))

Where :

[a, b] +a [c, d] = [a ++ c, b ++ d]

x ++ y =
- $-\infty$ if x = $-\infty$ or y = $-\infty$
- $+\infty$ if x = $+\infty$ or y = $+\infty$
- x + y otherwise

Interval interpretation on a language of instructions

Assignment axiom:

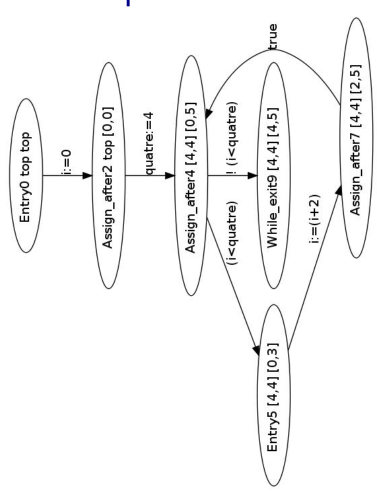
If int (x) = [a, b] just before x := expr,
what is the value of int (x) after that?

Simple examples:
- After x := 0, int (x) = [0, 0]
- After x := x+1, int (x) = [a+1, b+1]

A program

```
declare
    T : array (0..3) of integer ;
begin
    i:=0
    while i < 4
       T[i] := ...
       i := i+2
   end
end
```


The Graph



Exercise

Use the abstract domain of intervals with integer bounds.

Label the states with the best information you can compute about the intervals of `i` and `j`.

- What about the loop (does it stop?, in how many steps?, ...)
- Can we use this information to prove that the array is always accessed correctly?

The Complete Computation

CP	0	2	4	5	7	9
0	$\bot\bot$	$\bot\bot$	$\bot\bot$	$\bot\bot$	$\bot\bot$	$\bot\bot$
1	$\top\top$	$\bot\bot$	$\bot\bot$	$\bot\bot$	$\bot\bot$	$\bot\bot$
2	$\top\top$	$\top[0,0]$	$\bot\bot$	$\bot\bot$	$\bot\bot$	$\bot\bot$
3	$\top\top$	$\top[0,0]$	$[4,4][0,0]$	$\bot\bot$	$\bot\bot$	$\bot\bot$
4	$\top\top$	$\top[0,0]$	$[4,4][0,0]$	$[4,4][0,0]$	$\bot\bot$	$\bot\bot$
5	$\top\top$	$\top[0,0]$	$[4,4][0,0]$	$[4,4][0,0]$	$[4,4][2,2]$	$\bot\bot$
6	$\top\top$	$\top[0,0]$	$[4,4][0,2]$	$[4,4][0,0]$	$[4,4][2,2]$	$\bot\bot$
7	$\top\top$	$\top[0,0]$	$[4,4][0,2]$	$[4,4][0,2]$	$[4,4][2,2]$	$\bot\bot$
8	$\top\top$	$\top[0,0]$	$[4,4][0,2]$	$[4,4][0,2]$	$[4,4][2,4]$	$\bot\bot$
9	$\top\top$	$\top[0,0]$	$[4,4][0,4]$	$[4,4][0,2]$	$[4,4][2,4]$	$\bot\bot$
10	$\top\top$	$\top[0,0]$	$[4,4][0,4]$	$[4,4][0,3]$	$[4,4][2,4]$	$[4,4][4,4]$
11	$\top\top$	$\top[0,0]$	$[4,4][0,4]$	$[4,4][0,3]$	$[4,4][2,5]$	$[4,4][4,4]$
12	$\top\top$	$\top[0,0]$	$[4,4][0,5]$	$[4,4][0,3]$	$[4,4][2,5]$	$[4,4][4,4]$
13	$\top\top$	$\top[0,0]$	$[4,4][0,5]$	$[4,4][0,3]$	$[4,4][2,5]$	$[4,4][4,5]$
14	$\top\top$	$\top[0,0]$	$[4,4][0,5]$	$[4,4][0,3]$	$[4,4][2,5]$	$[4,4][4,5]$

Does it help?

What kind of property may be proved by such an analysis?

Concrete Collecting Semantics [Floyd67, Hoare69]

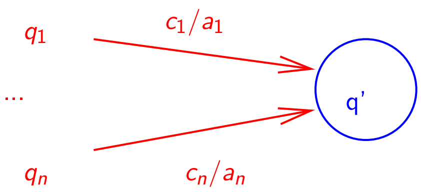
Idea: associate with each control point the set of possible valuations of the variables.
If q is a control point,
We denote by $\mathrm{V}(q)$ the set of valuations at this point.

If $q \xrightarrow{c} q'$ then $V(q') = V(q) \cap c$

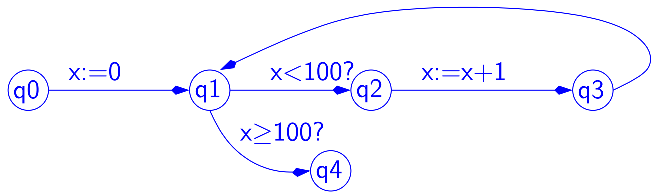
if $\{q_i \xrightarrow{a_i} q'\}_I$ then $V(q') = \bigcup_I \mathrm{Post}_{a_i}(V(q_i))$

On the compact form

If $\{q_i \xrightarrow{c_i/a_i} q'\}_I$ then $V(q') = \bigcup_I \mathrm{Post}_{a_i}(V(q_i) \wedge c_i)$



Abstract Collecting Semantics - Example



Abstract Collecting Semantics - Example

$$
\begin{aligned}
\text{INTERV (q0)} &= [-\infty, +\infty] \\
\text{INTERV (q1)} &= \text{Post (x:=0) (INTERV (q0))} \cup \text{INTERV (q3)} \\
\text{INTERV (q2)} &= \text{INTERV (q1)} \cap [-\infty, 99] \\
\text{INTERV (q3)} &= \text{Post (x:=x+1) (INTERV (q2))} \\
\text{INTERV (q4)} &= \text{INTERV (q1)} \cap [100, +\infty]
\end{aligned}
$$

We have to compute the solution of this system of equations.
It is a non-usual (and abstract) interpretation of the program.

This is a Fix-Point Equation

$$X = F(X)$$

Where:

$$X = \langle \text{INTERV(q0)}, \text{INTERV(q1)}, \text{INTERV(q2)}, \text{INTERV(q3)}, \text{INTERV(q4)} \rangle$$

And we need the least-fix-point (LFP) of this equation.
F is made of: Post functions, $\cup$, and $\cap$ with some interval.

Orders

A relation R $\subseteq$ E $\times$ E is an order iff it is:

- reflexive : $\forall x \in E.\quad (x, x) \in R$
- antisymmetrical : $(x, y) \in R$ and $(y, x) \in R$ imply $x=y$
- transitive : $(x, y) \in R$ and $(y, z) \in R$ imply $(x, z) \in R$

Partially Ordered Set

$(E, \leq)$

Where $\leq$ is a (partial) order on E

Upper and Lower Bounds

(ensembles majorants, minorants)

For $X \subseteq E$:

$\text{Maj}(X) = \{ y \in E \mid \forall x \in X.\ x \leq y \}$
$\text{Min}(X) = \{ y \in E \mid \forall x \in X.\ y \leq x \}$

Minimal and Maximal Elements

$X \subseteq E$ has a maximal element iff :

$$X \cap \text{Maj}(X) \neq \emptyset = \{ \max(X) \}$$

$X \subseteq E$ has a minimal element iff :

$$X \cap \text{Min}(X) \neq \emptyset = \{ \min(X) \}$$

Note that $x, y \in X \cap \text{Maj}(X)$ imply $x=y$
But $X \cap \text{Maj}(X)$ may be empty.

Least Upper Bound (supremum), Greatest Lower Bound (infimum)

$X \subseteq E$ has a Least Upper Bound B iff:
Maj(X) has a minimal element B.

$X \subseteq E$ has a Greatest Lower Bound B iff:
Min(X) has a maximal element B.

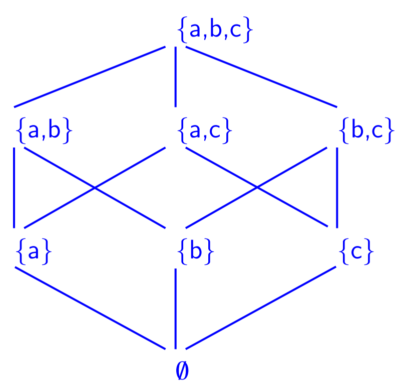
Complete Lattices

$(E, \leq)$ is a Complete Lattice iff:

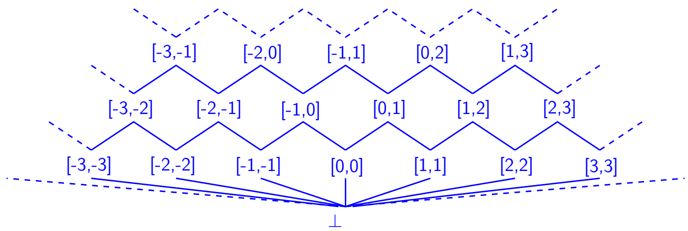
> Every subset $X \subseteq E$ has a Least Upper Bound, and a Greatest Lower Bound

$\perp$ denotes the GLB, and $\top$ the LUB.

Typical Example: Subsets of a set $\{a, b, c\}$



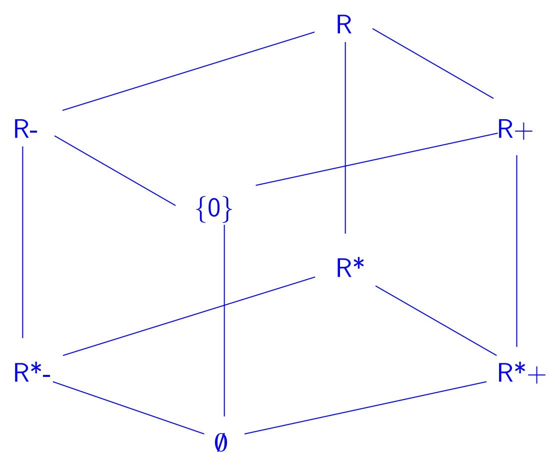
Intervals of R with integer bounds



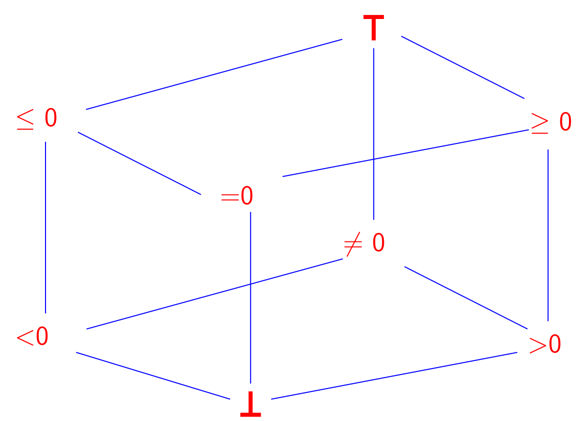
Signs

$\{ <0, =0, >0, \leq 0, \neq 0, \geq 0, ? \}$,
where $? = \top$,
to which we add $\emptyset$ or $\perp$,
ordered as the subsets of R they represent

Signs and subsets of R



Signs and subsets of R



Composing Lattices: Cartesian Product

The Cartesian Product of two Lattices is a Lattice:
$(a, b) \leq (a', b')$ iff $a \leq a'$ et $b \leq b'$

Needed because we have several control points and several variables in a program.

Fix-Points

$x \in E$ is a fix-point of $f : (E, \leq) \longrightarrow (E, \leq)$ iff:

$$x = f(x)$$

It is a pre-fix point iff :

$$x \leq f(x)$$

It is a post-fix point iff :

$$f(x) \leq x$$

Tarski Theorem

If $(E, \leq)$ is a complete lattice and
$f : (E, \leq) \longrightarrow (E, \leq)$ is total and monotonous,

Then f has a least fix-point denoted by lfp(f) :
$$\text{lfp}(f) = \text{supremum} \ (\ \text{postFP} \ (f))$$

and f has a greatest fix-point denoted by gfp(f) :
$$\text{gfp}(f) = \text{infimum} \ (\ \text{preFP} \ (f))$$

Kleene Theorem

If $(E, \leq)$ is a complete lattice and
$f : (E, \leq) \longrightarrow (E, \leq)$ is total and continuous

Then
$$\text{lfp} \ (f) = \text{supremum} \ (\ \{ \ f^n \ (\bot) \ \} \ )$$

$$\text{gfp} \ (f) = \text{infimum} \ (\ \{ \ f^n \ (\top) \ \} \ )$$

Computing the LFP by iterations

lfp (f) is the "limit" of: $x0 = \bot, \quad x(n+1) = f\ (x(n))$

```
fp := bot ;
loop
      new_fp := f(fp) ;
      exit when fp = new_fp ;
      fp := new_fp ;
end loop;
-- fp is the least-fix-point of f
```


Remember The Fix-Point Equation (Abstract Collecting Semantics)

$$\begin{aligned}
\text{INTERV (q0)} &= [-\infty, +\infty] \\
\text{INTERV (q1)} &= \text{Post (x:=0) (INTERV (q0))} \ \cup \ \text{INTERV (q3)} \\
\text{INTERV (q2)} &= \text{INTERV (q1)} \ \cap \ [-\infty, 99] \\
\text{INTERV (q3)} &= \text{Post (x:=x+1) (INTERV (q2))} \\
\text{INTERV (q4)} &= \text{INTERV (q1)} \ \cap \ [100, +\infty]
\end{aligned}$$

Remember The Fix-Point Equation (Abstract Collecting Semantics)

$$X = F(X)$$

Where:
$$X \ = < \ \text{INTERV(q0), INTERV(q1), INTERV(q2), INTERV(q3), INTERV(q4)} >$$

And we need the least-fix-point (LFP) of this equation.
F is made of: Post functions, $\cup$, and $\cap$ with some interval.

Iterative Computation

Snapshot $=<$ INTERV(q_0),...INTERV(q_4) $>$
To be computed by iterations: Snapshot^0, Snapshot^1, Snapshot^2, ...

Snapshot^0 = $< \perp, \perp, \perp, ... \perp >$
Snapshot^{n+1} = Abstract Collecting Semantics Equations Applied to (Snapshot^n)

In particular:
Snapshot^1 $=< \top, \perp, \perp, ... \perp >$

Example Computation (for another program)

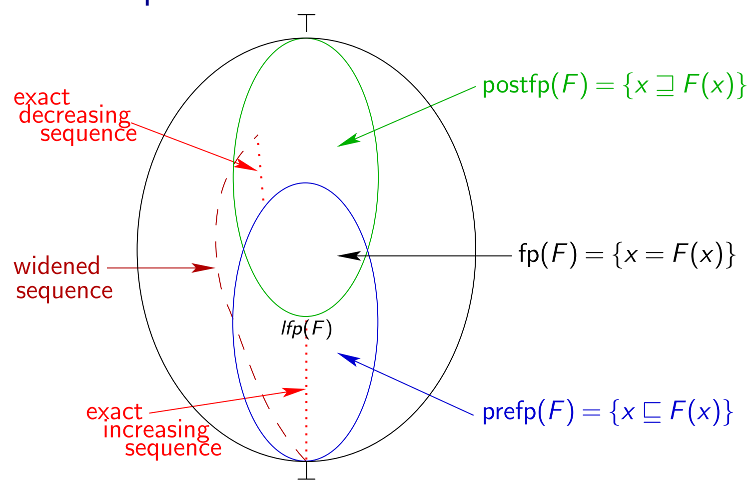
CP	0	2	4	5	7	9
0	$\perp\perp$	$\perp\perp$	$\perp\perp$	$\perp\perp$	$\perp\perp$	$\perp\perp$
1	$\top\top$	$\perp\perp$	$\perp\perp$	$\perp\perp$	$\perp\perp$	$\perp\perp$
2	$\top\top$	$\top[0,0]$	$\perp\perp$	$\perp\perp$	$\perp\perp$	$\perp\perp$
3	$\top\top$	$\top[0,0]$	$[4,4][0,0]$	$\perp\perp$	$\perp\perp$	$\perp\perp$
4	$\top\top$	$\top[0,0]$	$[4,4][0,0]$	$[4,4][0,0]$	$\perp\perp$	$\perp\perp$
5	$\top\top$	$\top[0,0]$	$[4,4][0,0]$	$[4,4][0,0]$	$[4,4][2,2]$	$\perp\perp$
6	$\top\top$	$\top[0,0]$	$[4,4][0,2]$	$[4,4][0,0]$	$[4,4][2,2]$	$\perp\perp$
7	$\top\top$	$\top[0,0]$	$[4,4][0,2]$	$[4,4][0,2]$	$[4,4][2,2]$	$\perp\perp$
8	$\top\top$	$\top[0,0]$	$[4,4][0,2]$	$[4,4][0,2]$	$[4,4][2,4]$	$\perp\perp$
9	$\top\top$	$\top[0,0]$	$[4,4][0,4]$	$[4,4][0,2]$	$[4,4][2,4]$	$\perp\perp$
10	$\top\top$	$\top[0,0]$	$[4,4][0,4]$	$[4,4][0,3]$	$[4,4][2,4]$	$[4,4][4,4]$
11	$\top\top$	$\top[0,0]$	$[4,4][0,4]$	$[4,4][0,3]$	$[4,4][2,5]$	$[4,4][4,4]$
12	$\top\top$	$\top[0,0]$	$[4,4][0,5]$	$[4,4][0,3]$	$[4,4][2,5]$	$[4,4][4,4]$
13	$\top\top$	$\top[0,0]$	$[4,4][0,5]$	$[4,4][0,3]$	$[4,4][2,5]$	$[4,4][4,5]$
14	$\top\top$	$\top[0,0]$	$[4,4][0,5]$	$[4,4][0,3]$	$[4,4][2,5]$	$[4,4][4,5]$

Problems

In general:

- We cannot compute on E, nor decide =
- The sequence may be infinite (the mathematical definition is ok, but the algorithm does not terminate)

The Complete Picture



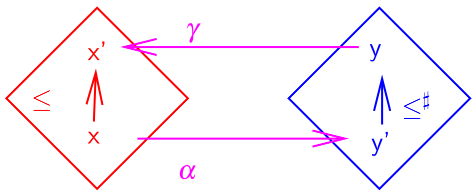
Definition

A Galois connection between:
a concrete complete lattice (E, $\leq$, $\cup$, $\cap$, $\perp$, $\top$) and
an abstract complete lattice ($E^\sharp$, $\leq^\sharp$, $\cup^\sharp$, $\cap^\sharp$, $\perp^\sharp$, $\top^\sharp$)

is a pair of functions α (abstraction),
and γ (concretization) such that :

$$\forall x \in E, \forall y \in E^\sharp, \quad \alpha(x) \leq^\sharp y \iff x \leq \gamma(y)$$

Galois Connection



Relationships between abstraction and concretization

Of course, $\alpha \neq \gamma^{-1}$.

α is an abstraction, hence non injective.

But x and $\gamma(\alpha$ (x)) are linked by the partial order.

Properties of Galois Connections: $\gamma \circ \alpha$ is extensive

$$\forall x \in E . x \leq \gamma(\alpha(x))$$

Because $\alpha(x) \leq^\sharp \alpha(x)$ implies $x \leq \gamma(\alpha(x))$.

Properties of Galois Connections: $\alpha \circ \gamma$ is retractive

$$\forall y \in E^\sharp . \alpha(\gamma(y)) \leq^\sharp y$$

Properties of Galois Connections: α is monotonous

Since $x \leq x'$ implies $x \leq \gamma(\alpha(x'))$
($\gamma \circ \alpha$ is extensive)

hence $\alpha(x) \leq^\sharp \alpha(x')$
(definition of a Galois connection)

Properties of Galois Connections: γ is monotonous

similar proof.

Properties of Galois Connections: Triple compositions

$$\gamma \circ \alpha \circ \gamma = \gamma$$
$$\alpha \circ \gamma \circ \alpha = \alpha$$

Properties of Galois Connections: Each function defines the other one

$$\alpha(x) = \text{supremum}\ (\ \{y \mid x \leq \gamma(y)\}\ )$$

$$\gamma(y) = \text{infimum}\ (\ \{x \mid \alpha(x) \leq^\sharp y\}\ )$$

Properties of Galois Connections: Preservations of the limits

$$\alpha(\text{infimum}(\ \{x \mid x \in X\}\ )) = \text{infimum}(\ \{\alpha(x) \mid x \in X\}\ )$$

Similarly:

$$\gamma(\text{supremum}^\sharp(\ \{y \mid y \in Y\}\ )) = \text{supremum}^\sharp(\ \{\gamma \mid y \in Y\}\ )$$

Subsets of R and Signs

Abstraction

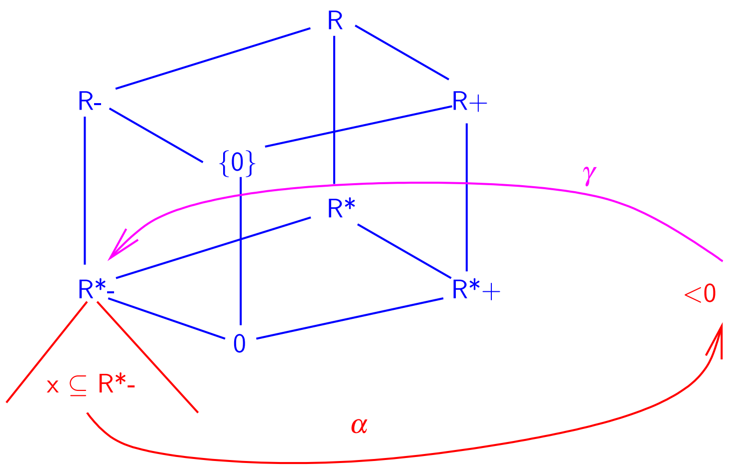
$$\forall X \subseteq R, \quad \alpha(X) = \begin{cases} \bot^\sharp \text{ if } X = \emptyset \\ \text{else } < 0 \text{ if } X \subseteq R^{*-} \\ \text{else } > 0 \text{ if } X \subseteq R^{*+} \\ \text{else } = 0 \text{ if } X = \{0\} \\ \text{else } \leq 0 \text{ if } X \subseteq R^- \\ \text{else } \geq 0 \text{ if } X \subseteq R^+ \\ \text{else } \neq 0 \text{ if } 0 \notin X \\ \text{else } \top^\sharp \end{cases}$$

Subsets of R and Signs

Concretization

$$\begin{array}{ll} \gamma(\bot^\sharp) = \emptyset & \gamma(<0) = R^{*-} \\ \gamma(\leq 0) = R^- & \gamma(=0) = \{0\} \\ \gamma(\neq 0) = R^* & \gamma(\geq 0) = R^+ \\ \gamma(>0) = R^{*+} & \gamma(\top^\sharp) = R \end{array}$$

Subsets of R and Signs



Subsets of R and intervals

Order : (a,b) $\leq^\sharp$ (a',b') iff a $\geq$ a' and b $\leq$ b'

Abstraction

$\forall$ X $\subseteq$ R α (X) = (inf (X), sup (X))

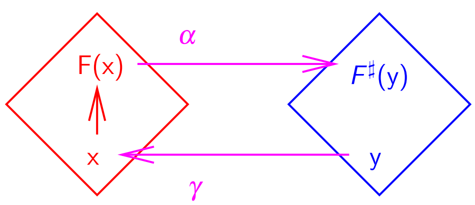
Subsets of R and intervals

Concretization

γ((a,b)) = { x $\in$ R | a $\leq$ x $\leq$ b}

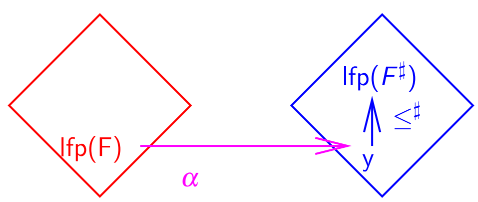
Defining a function on $E^\sharp$ from a function on E

To compute the fix-point of a function F : E $\longrightarrow$ E, we may try to use the function $F^\sharp$ (y) = α (F (γ (y))) on the abstract side



(note that $F^\sharp$ is monotonous, because $F^\sharp = \alpha \circ$ F $\circ\, \gamma$ and these 3 functions are monotonous)

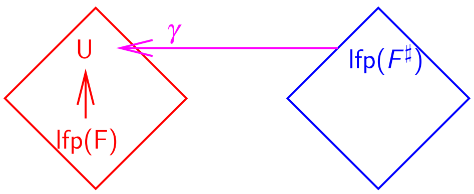
Theorem: abstracting fix-points



α (lfp(F)) $\leq^\sharp$ lfp ($F^\sharp$)

Because $\forall x \in E, \forall y \in E^\sharp, \quad \alpha(x) \leq^\sharp y \iff x \leq \gamma(y)$
...

We also have...



A First Approximation

To find an approximation of lfp(F) on the concrete side, it is sufficient to take:

$$\gamma\ (\mathrm{lfp}\ (\alpha \circ \mathrm{F} \circ \gamma))$$

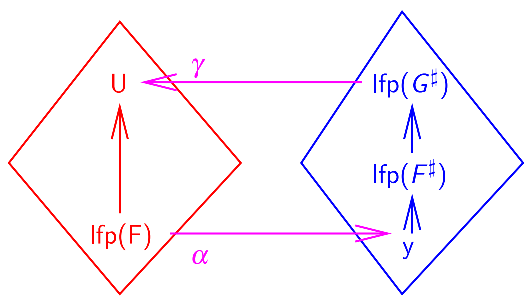
But, remember we cannot compute F...

Moreover, recall that...

If $\mathrm{f} \leq \mathrm{g}$ (f, g monotonous), then $\mathrm{lfp}\ (\mathrm{f}) \leq \mathrm{lfp}\ (\mathrm{g})$

Let us take a function $G^\sharp$ greater than (according to $\leq^\sharp$) $F^\sharp$.

The Complete Picture



The fix-point lfp(F) is $\leq$ than $\mathrm{U} = \gamma\ (\mathrm{lfp}\ (G^\sharp))$

Where $F^\sharp \leq^\sharp G^\sharp$ and $F^\sharp = \alpha \circ \mathrm{F} \circ \gamma$.

A Second Approximation

To find an approximation of lfp(F) on the concrete side, it is sufficient to take:

$$\gamma\ (\mathrm{lfp}\ (G^\sharp))$$

where $G^\sharp$ is greater than $F^\sharp = \alpha \circ \mathrm{F} \circ \gamma$

We cannot compute F, but we can find a $G^\sharp$ by decomposing F.

Lemma on the abstraction of a composition

$$(u \circ v)^\sharp \leq^\sharp u^\sharp \circ v^\sharp$$

Because $u^\sharp \circ v^\sharp = \alpha \circ \mathrm{u} \circ \gamma \circ \alpha \circ \mathrm{v} \circ \gamma$

and $\gamma \circ \alpha$ is extensive and monotonous.

Decomposing F, the Idea

Assume $F = u \circ v$.

Then $F^\sharp = (u \circ v)^\sharp \leq^\sharp u^\sharp \circ v^\sharp$

Hence we can take $G^\sharp = u^\sharp \circ v^\sharp$.

Interesting if it is simpler to obtain $u^\sharp$ and $v^\sharp$ than $F^\sharp$.

The Concrete Lattice

For a program with n states and m variables of type `T1...Tm`, the lattice is $(T1 \times T2 \times ... \times Tm)^n$.
With each state q, we associate V(q), a set of values of the tuple of variables.

The function F whose fix-point has to be approximated

The collecting semantics is a system of equations of the form:

$$\begin{array}{lcl} V\ (q0) & = & f0\ (V(q0),\ ...,\ V(qn)) \\ V\ (q1) & = & f1\ (V(q0),\ ...,\ V(qn)) \\ \quad ... & & \\ V\ (qn) & = & fn\ (V(q0),\ ...,\ V(qn)) \end{array}$$

or:
$< V(q0), ...\ V(qn) > = F\ (< V(q0), ...\ V(qn) >)$.

We need to (upper-)approximate the least fix point of F.

Is F monotonous?

F is made of: $V(q') = \bigcup_I \text{Post}_{a_i}(V(q_i) \cap c_i)$

where $\{q_i \xrightarrow{c_i/a_i} q'\}_I$

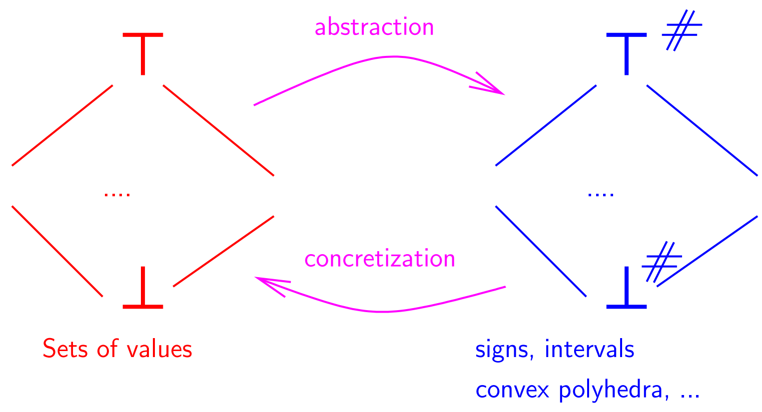
$\cap$ c, Post(a) and $\cup$ are monotonous

V $\subseteq$ V' implies V $\cap$ c $\subseteq$ V' $\cap$ c

V $\subseteq$ V' implies Post(a) (V) $\subseteq$ Post(a) (V')
since Post(a) (V) = { Post(a)(u) | u $\in$ V }

$\cup$ is monotonous... trivial.

We choose an abstract lattice



Computing in the abstract lattice

We search for a function $G^\sharp$ greater than $F^\sharp = \alpha \circ \mathrm{F} \circ \gamma$

Then we'll have: $\mathrm{lfp}\ (G^\sharp) \geq \mathrm{lfp(F)}$

We cannot compute F, so we have to define $G^\sharp$ directly, and then prove that $G^\sharp$ is greater than $\alpha \circ \mathrm{F} \circ \gamma$.

Using the Lemma on the abstraction of a composition

Lemma: $(u \circ v)^\sharp \leq^\sharp u^\sharp \circ v^\sharp$

F is some composition of: Post(a), $\cap$ c, $\cup$.

Where a is some assignment and c is some condition.

Defining $G^\sharp$ directly

We define $G^\sharp$ as a composition of:
— the abstract version of Post(a): $\mathrm{Post(a)}^\sharp$
— the abstract version of $\cap$ c: $\cap c^\sharp$
— the abstract version of $\cup$: $\cup^\sharp$ (the infimum of the abstract lattice)

Defining $\mathrm{Post(a)}^\sharp$

— Direct definition and
— Proof of: $Post(a)^\sharp \geq^\sharp \alpha \circ \mathrm{Post(a)} \circ \gamma$

Defining $\cap c^\sharp$

— Direct definition and
— Proof of: $(\cap c)^\sharp \geq^\sharp \alpha \circ (\cap\ \mathrm{c}) \circ \gamma$

Example: Intervals

Direct definition of Post(a)$^\sharp$:

If int (x) = [a, b] just before x := expr,
what is the value of int (x) after that?

Simple examples:
After x := 0, int (x) = [0, 0]
After x := x+1, int (x) = [a+1, b+1]

General definition: ...

Example: Intervals

Abstraction: $\forall$ X $\subseteq$ R α (X) = (inf (X), sup (X))
Concretization: γ((a,b)) = { x $\in$ R | a $\leq$ x $\leq$ b}

We have to prove: $Post(a)^\sharp \geq^\sharp \alpha \circ \mathrm{Post(a)} \circ \gamma$

Defining Post(a)$^\sharp$: Example with Intervals

INT = (R $\cup$ { - ∞, + ∞ }) $\times$ (R $\cup$ { - ∞, + ∞ })

Order: [a,b] $\leq^\sharp$ [a',b'] iff a $\geq$ a' and b $\leq$ b'

Abstraction: $\forall$ X $\subseteq$ R α (X) = [inf (X), sup (X)]

Concretization: γ([a,b]) = { x $\in$ R | a $\leq$ x $\leq$ b}

Composition:

$$(\alpha \circ \mathrm{Post(a)} \circ \gamma)(J) =$$
$$\text{let } G = \{a(x) \mid x \in J\} \text{ in } [\inf(G), \sup(G)]$$

Example

Direct definition:
Post(`x:=x+1`)$^\sharp$([3,4]) = [4,5]

Should be greater than:
$[\inf(G), \sup(G)]$ where: $G = \{x+1 \mid x \in [3,4]\} = \{4,5\}$.

Decomposing Assignments

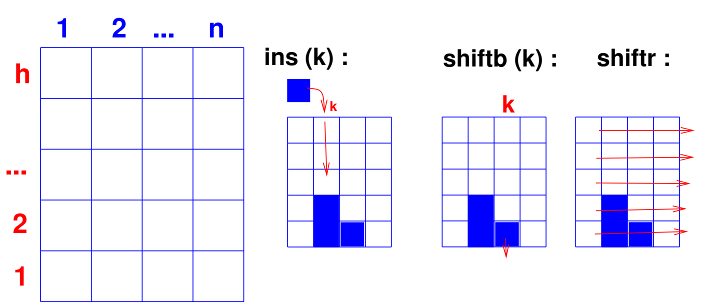
Direct definition:
Post(`x:=x*x-x`)$^\sharp$([3,5]) = [6,20]
But we cannot give direct definition for all of these possible assignments (proving the direct definition is right).

Hence we compute separately the two operations: `y := x*x ; x := y - x`
which gives [4,22].

15 Program Validation Examples
- Mini Tetris
  - A Data structure and the associated operations
  - A program
  - Abstractions

15 Program Validation Examples
- Mini Tetris
  - A Data structure and the associated operations
  - A program
  - Abstractions

The data structure



Initial state and operations

`initial state :` all columns are empty .

`ins(k)` : insert in column k, the element falls until it reaches a non-empty place. Null-operation if the column is full.

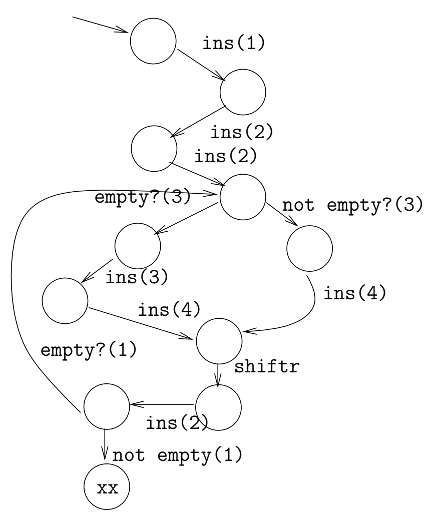
`shiftb (k)` : shift column k towards bottom. Null-operation if column k is empty

`shiftr`: shift all lines towards right. Null-operation for empty lines

`Empty?(k)`: determines whether column k is empty.

15 Program Validation Examples
- Mini Tetris
  - A Data structure and the associated operations
  - A program
  - Abstractions


```
ins (1) ;
ins (2) ; ins (2) ;
loop
    if empty? (3)
    then
       ins (3) ;
       ins (4) ;
    else
       ins (4) ;
    endif ;
    shiftr ;
    ins (2) ;
    if not empty?(1)
    then exit ;
endloop ;
-- point xx
```



15 Program Validation Examples
- Mini Tetris
  - A Data structure and the associated operations
  - A program
  - Abstractions

Abstraction of a set of values

Concrete State of the data structure:
a function from $[1, n] \times [1, h]$ to Boolean values.
$X(k)(u)$ means: the place column k, line u, is non empty.

Let $E = \{X_1, X_2, ...X_p\}$ be a set of such concrete states.

Abstraction: a vector $< b_1, ..., b_n >$ such that:

$$\forall i \in [1, n], b_i = \max\{k | \exists u \in [1, p].X_u(i)(k)\}$$

Order on abstract values, and the lattice

- Define the order on the abstract values
- Define $\perp$ and $\top$ in order to obtain a complete lattice

Abstract operations

For `ins(k)`, `shiftb (k)`, `shiftr`