Program analysis using 3-Valued Logic

Mooly Sagiv

Tel Aviv University

http://www.cs.tau.ac.il/~tvla

Tentative Schedule

1. Abstract interpretation in the nutshell
2. Canonical Abstractions of dynamically allocated storage
3. Abstract interpretation using canonical abstraction
4. Abstract interpretation using canonical abstraction
5. Applications & TVLA
6. Advanced Topics

Abstract Interpretation in the nutshell

- Principles of Program Analysis
  F. Nielson, H. Nielson, C.L. Hankin (2, 4)
- N.D. Jones and F. Nielson. *Abstract Interpretation: a Semantics-Based Tool for Program Analysis*. 1994
- Patrick Cousot's homepage

Abstract Interpretation Static Analysis

- Automatically identify program properties
  - No user provided loop invariants
- Sound but incomplete methods
  - But can be rather precise
- Non-standard interpretation of the program operational semantics
- Usages
  - Compiler optimization
    - » Collect static information for program transformations
  - Code quality tools
    - » Identify potential bugs
    - » Prove the absence of runtime errors
    - » Partial correctness

Memory Leakage

```
List* reverse(List *head)
{
    List *rev, *n;
    rev = NULL;
    while (head != NULL) {
        n = head →next;
        head → next = rev;
        head = n;
        rev = head;
    }
    return rev;
}
```

```
typedef struct List {
    int d;
    struct List* next;
} List;
```

leakage of address pointed to by head

Foundation of Static Analysis

- Static analysis can be viewed as interpreting the program over an “abstract domain”
- Execute the program over larger set of execution paths
- Guarantee sound results
  - Every identified constant is indeed a constant
  - But not every constant is identified as such

Even/Odd Abstract Interpretation

- Determine if an integer variable is even or odd at a given program point

Example Program

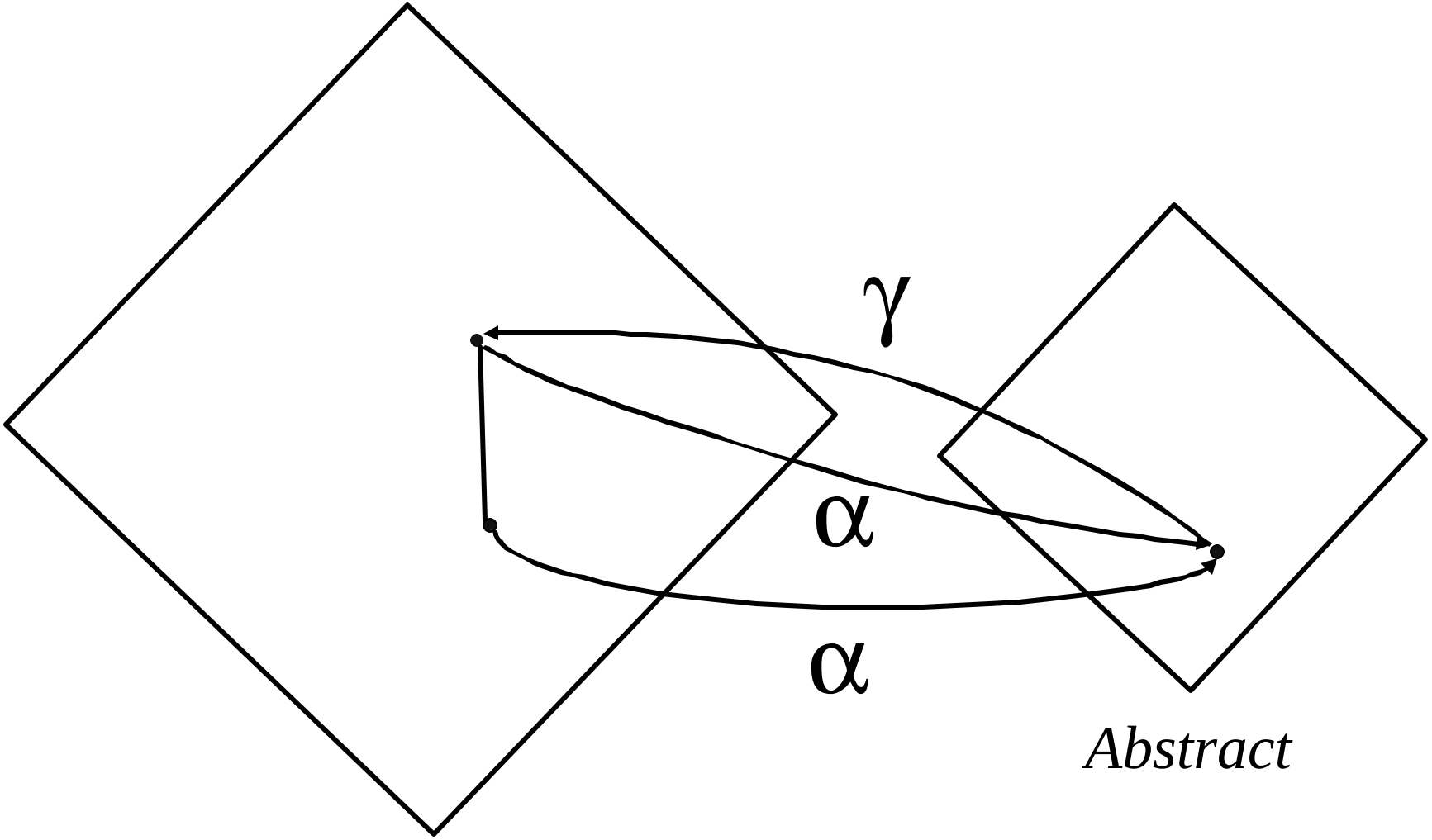
```
   /* x=? */

while  (x !=1)  do {   /* x=? */
          if   (x %2) == 0
   /* x=E */              { x := x / 2; }        /* x=? */
                    else
   /* x=O */             { x := x * 3 + 1;       /* x=E */
                         assert (x %2 ==0); }
}
/* x=O*/
```

Example Program

```
while  (x !=1)  do {
        if   (x %2) == 0
                        { x := x / 2; }
                else
                        { x := x * 3 + 1;
                          x := x /2; }
}
```

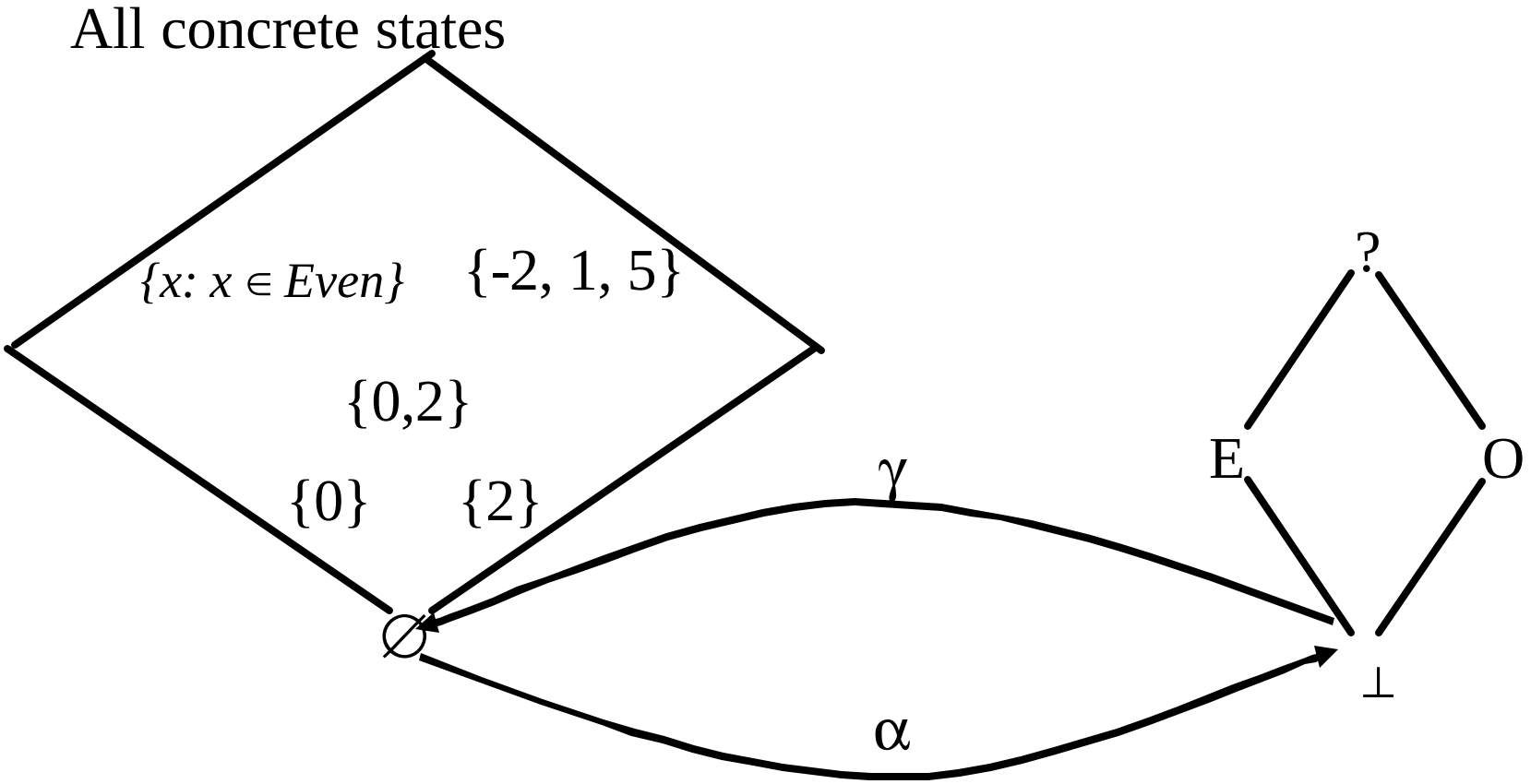
Abstract Interpretation



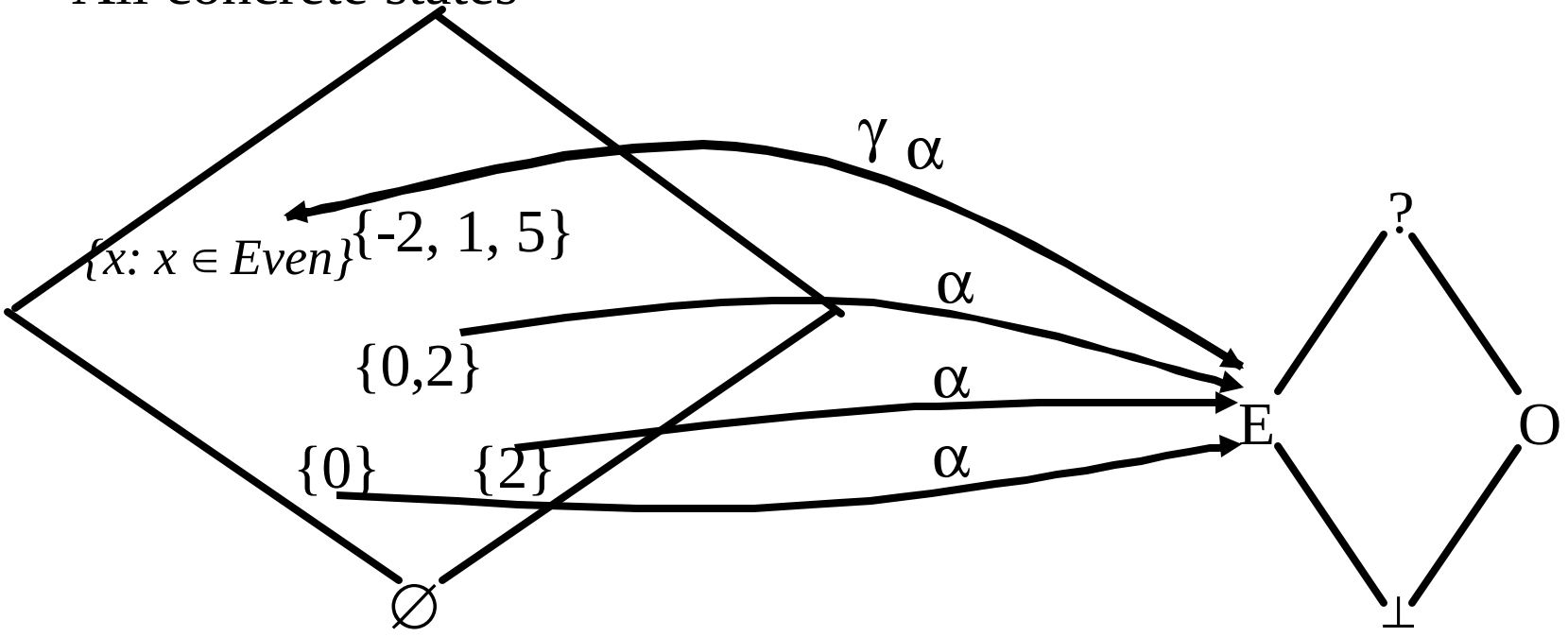
α

Sets of stores → *Descriptors of sets of stores*

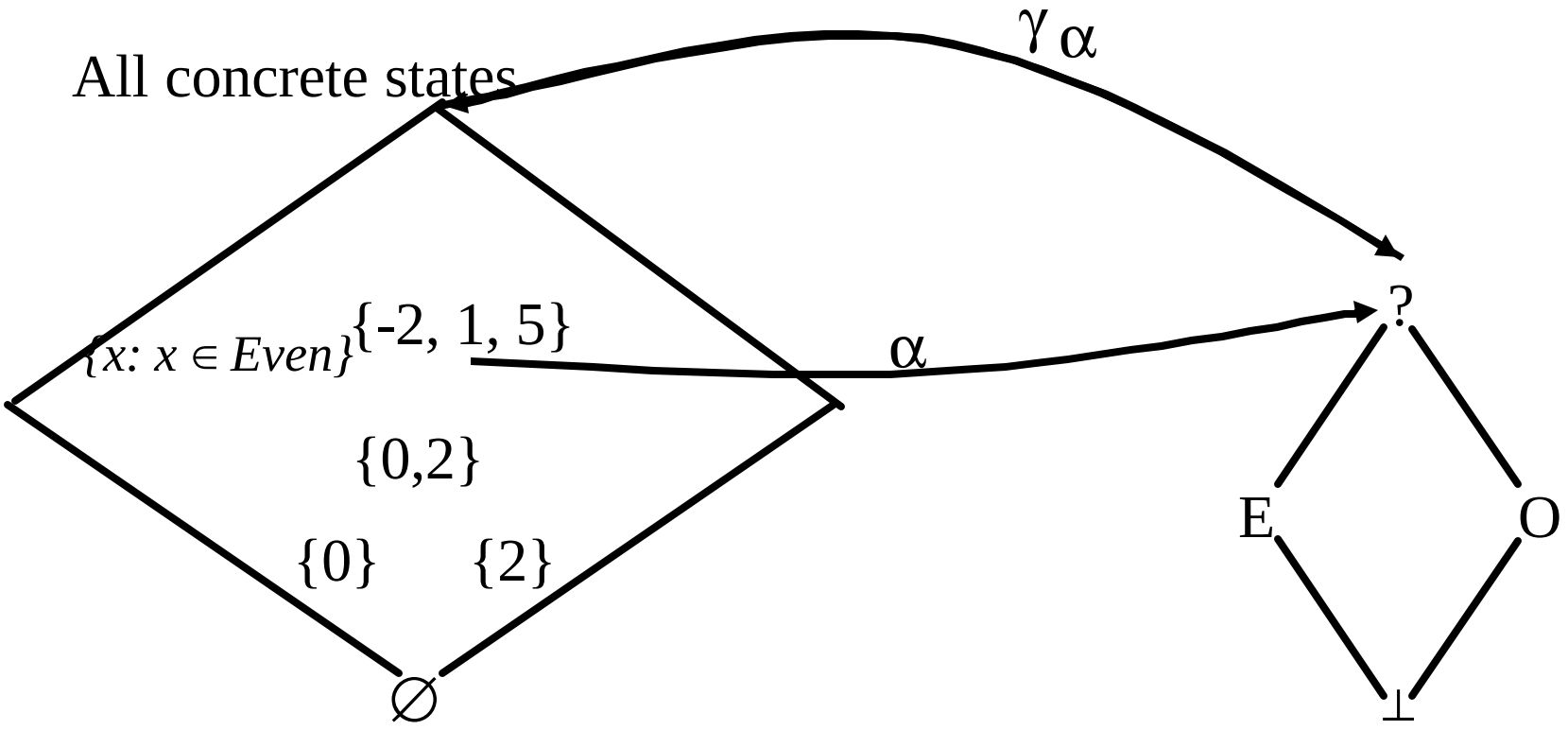
Odd/Even Abstract Interpretation



Odd/Even Abstract Interpretation



Odd/Even Abstract Interpretation



Odd/Even Abstract Interpretation

$\alpha(X)$ = if $X = \emptyset$ return $\perp$

else if for all z in X (z%2 == 0)

return E

else if for all z in X (z%2 == 0)

return O

else return ?

$\gamma(a)$ = if $a = \perp$ return $\emptyset$

else if a = E return Even

else if a= O return Odd

else return Natural

Example Program

```
while  (x !=1)  do {
          if   (x %2) == 0
                              { x := x / 2; }
                    else
   /* x=O */                { x := x * 3 + 1;       /* x=E */
                               assert (x %2 ==0); }
}
```

Concrete and Abstract Interpretation

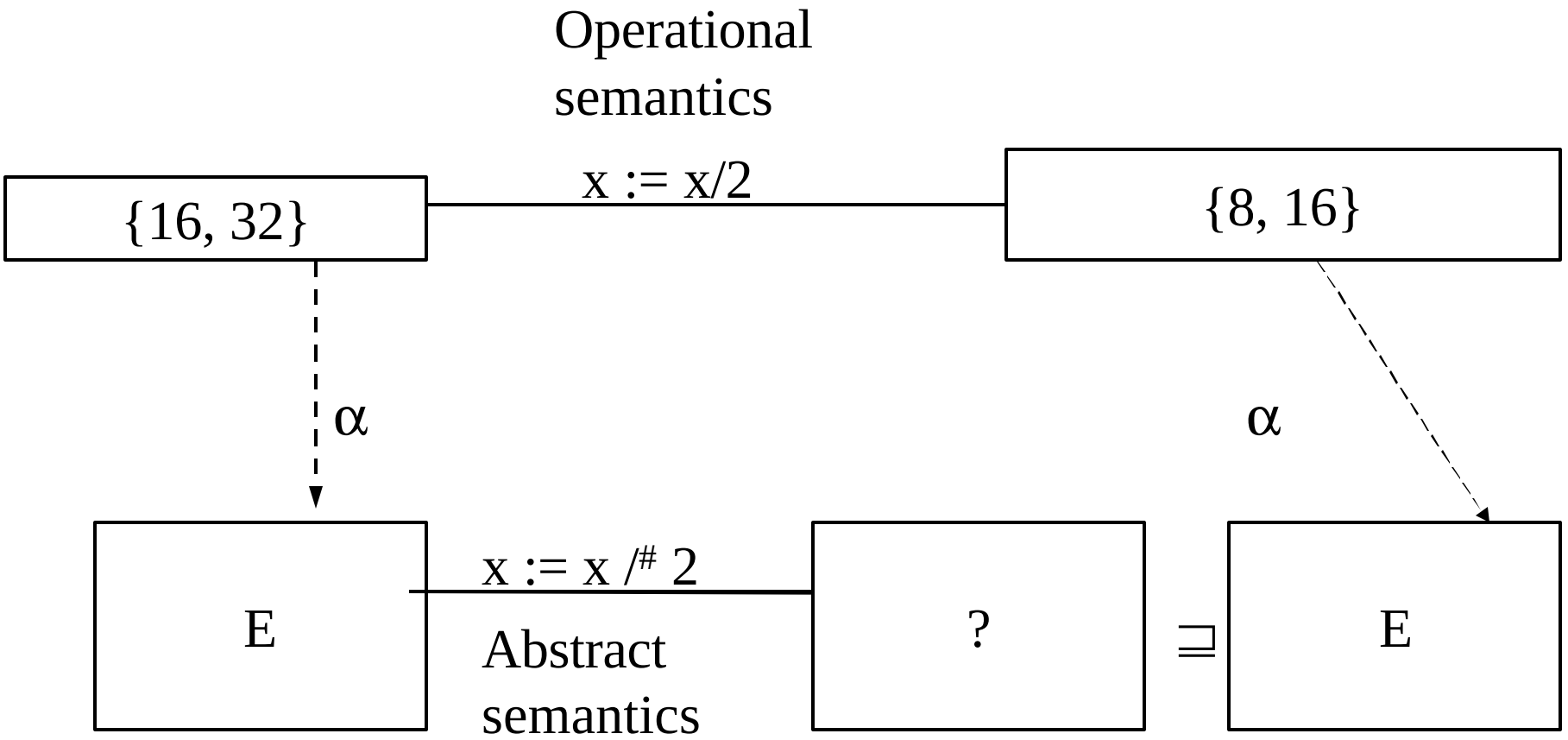
+	0	1	2	3	…
0	0	1	2	3	…
1	1	2	3	4	…
2	2	3	4	5	…
3	3	4	5	6	…
⋮	⋮	⋮	⋮	⋮	

*	0	1	2	3	…
0	0	0	0	0	…
1	0	1	2	3	…
2	0	2	4	6	…
3	0	3	6	9	…
⋮	⋮	⋮	⋮	⋮	

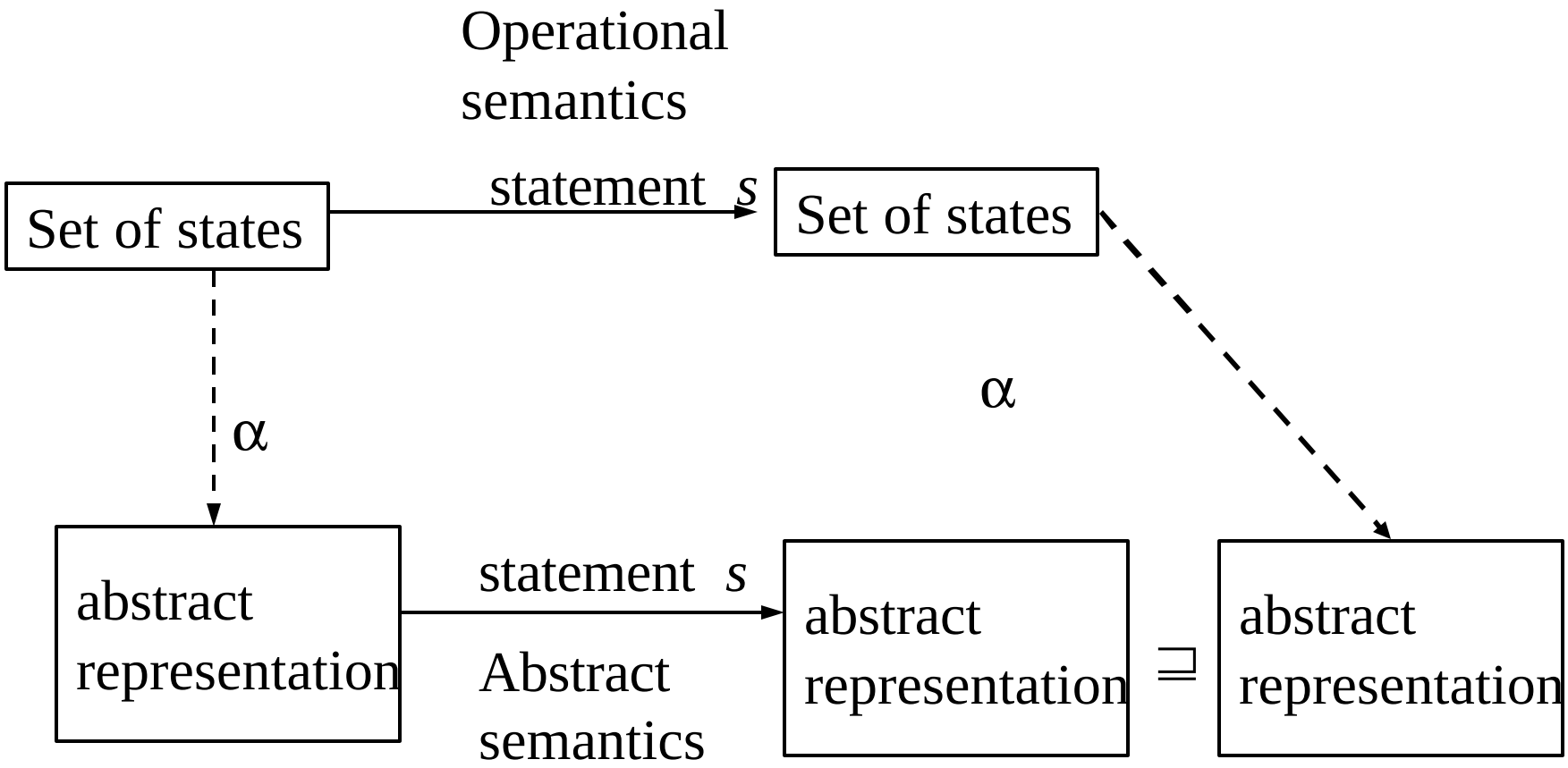
+’	?	O	E
?	?	?	?
O	?	E	O
E	?	O	E

*’	?	O	E
?	?	?	E
O	?	O	E
E	E	E	E

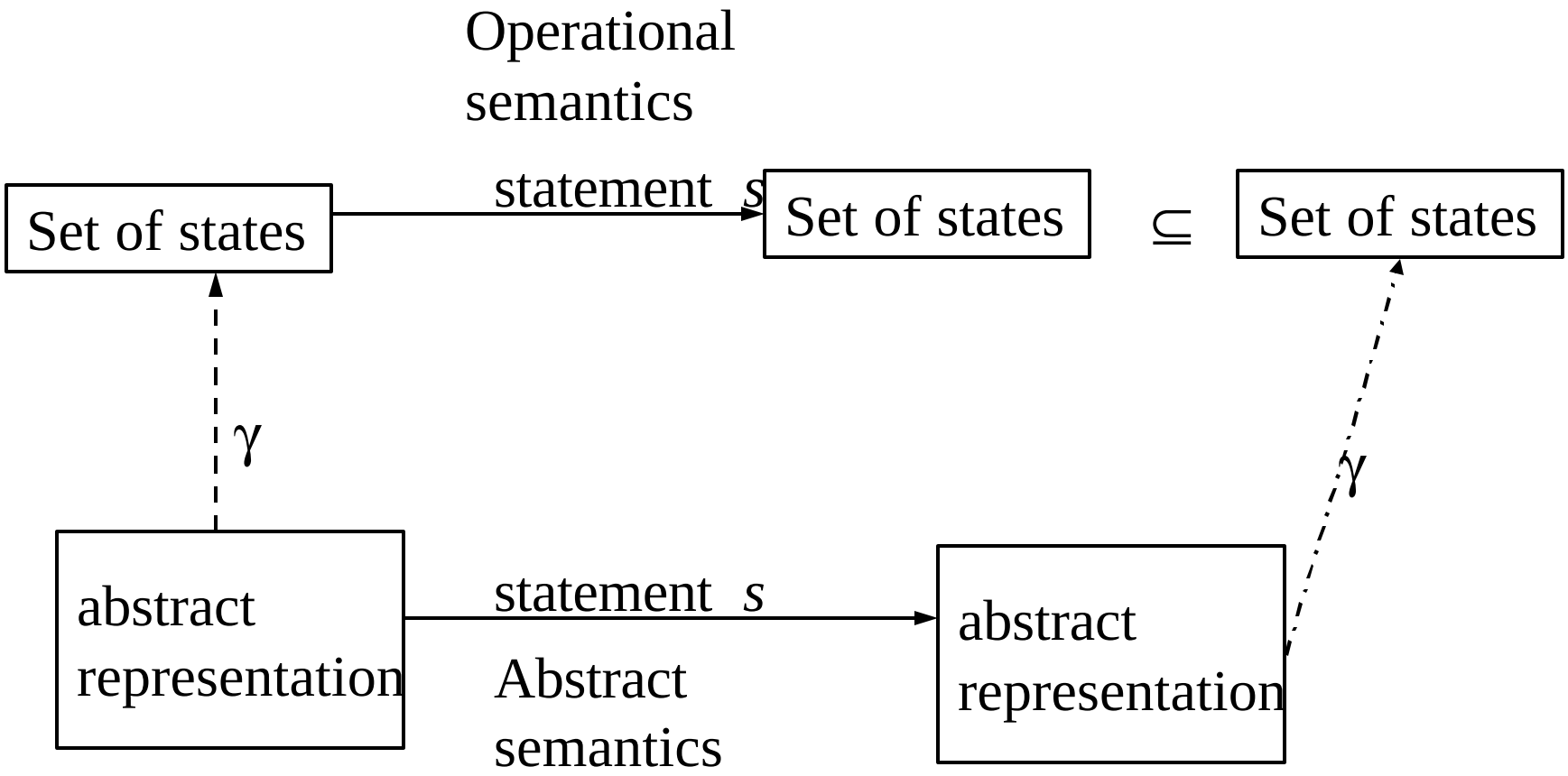
Abstract interpretation cannot be always homomorphic (Odd/Even)



Abstract (Conservative) interpretation



Abstract (Conservative) interpretation



Challenges in Abstract Interpretation

- Finding appropriate program semantics (runtime)
- Designing abstract representations
  - What to forget
  - What to remember
    - Summarize crucial information
- Designing Abstract Transformers
- Scalability
  - Large programs
  - Missing source code
- Precise enough

Runtime vs. Abstract Interpretation (Software Quality Tools)

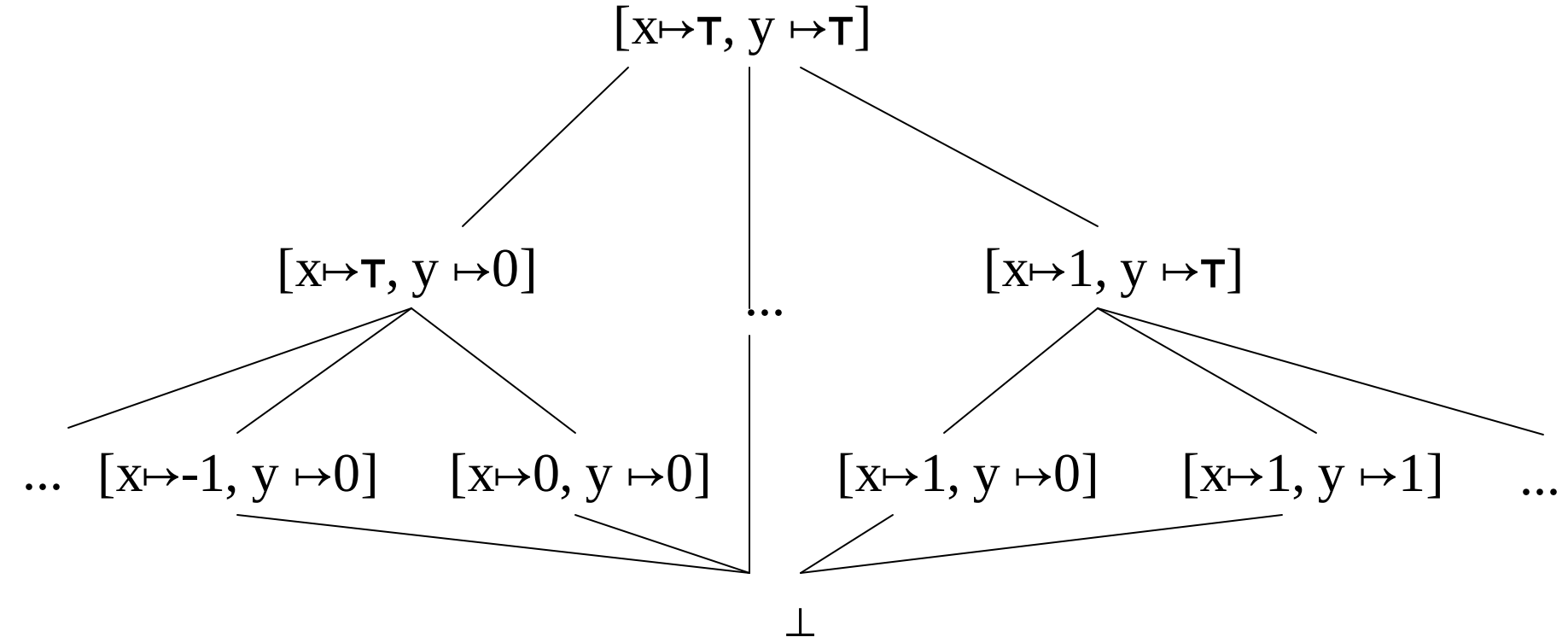
	Runtime	Abstract
Effectiveness	Missed Errors	False alarms
		Locate rare errors
Cost	Proportional to program's execution	Proportional to program's size

Constant Propagation Abstract Interpretation

- Determine if a variable has a constant value at a given program point

Constant Propagation Lattice

$[\text{Var} \rightarrow Z^{\top}]^{\perp}$



Example Constant Propagation

- Abstract representation set of integer values and and extra value "⊤" denoting variables not known to be constants
- Conservative interpretation of +

+#	⊤	0	1	2
⊤	⊤	⊤	⊤	⊤
0	⊤	0	1	2
1	⊤	1	2	3
2	⊤	2	3	4
…	⊤	…	…	…

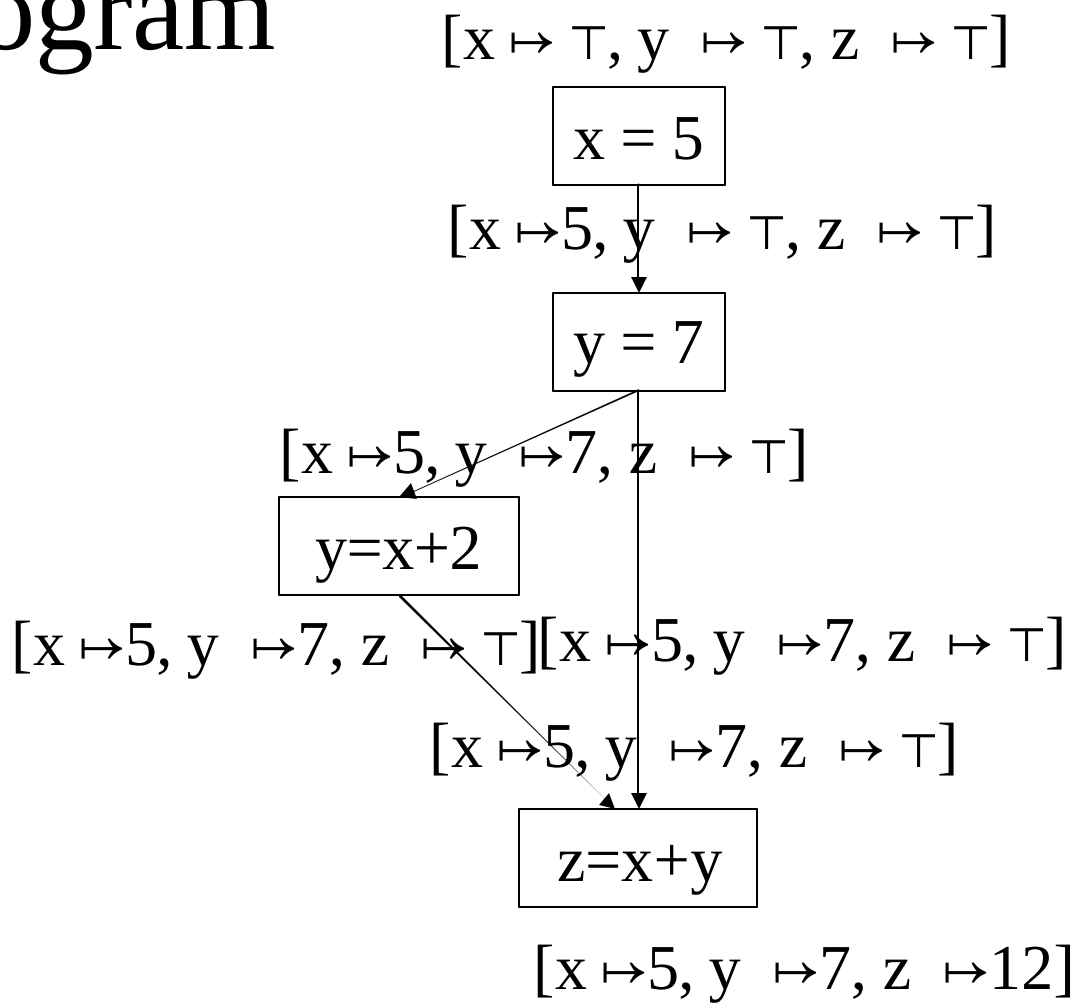
Example Constant Propagation (Cont)

- Conservative interpretation of *

*#	⊤	0	1	2
⊤	⊤	0	⊤	⊤
0	0	0	0	0
1	⊤	0	1	2
2	⊤	0	2	4
…	⊤	0	…	…

Example Program

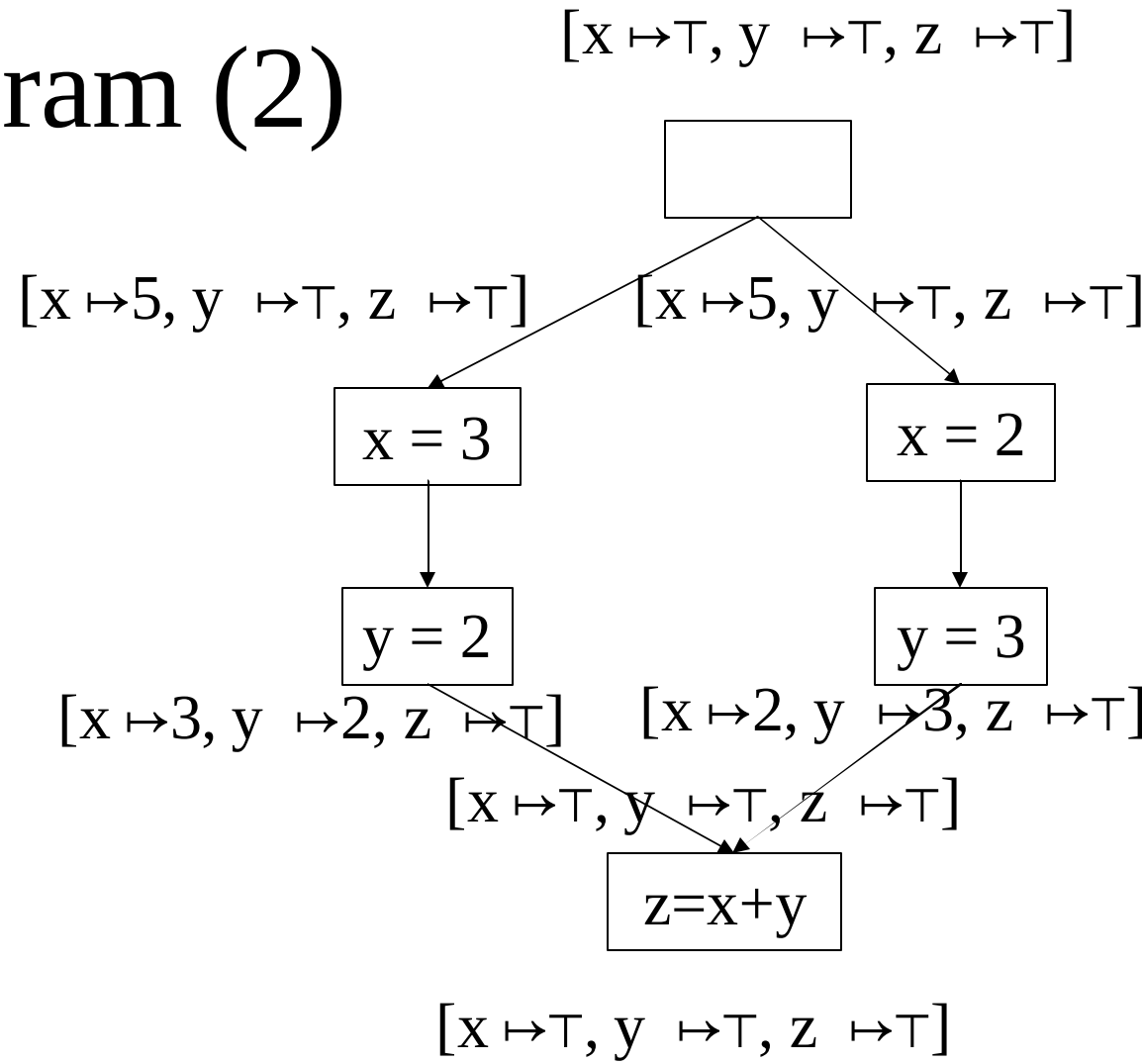
```
x = 5;
y = 7;
if (getc())
    y = x + 2;
z = x +y;
```



Example Program (2)

```
if (getc())
    x= 3 ; y = 2;
else
    x =2; y = 3;
z = x +y;
```

[x ↦5, y ↦7, z ↦?]



Undecidability Issues

- It is undecidable if a program point is reachable in some execution
- Some static analysis problems are undecidable even if the program conditions are ignored

The Constant Propagation Example

```
while (getc()) {
        if (getc()) x_1 = x_1 + 1;
        if (getc()) x_2 = x_2 + 1;
        ...
       if  (getc()) x_n = x_n + 1;
        }
y =  truncate (1/ (1 + p^2(x_1, x_2, ..., x_n))
/* Is y=0 here? */
```

Coping with undecidabilty

- Loop free programs
- Simple static properties
- Interactive solutions
- Conservative (sound) estimations
  - Every enabled transformation cannot change the meaning of the code but some transformations are not enabled
  - Non optimal code
  - Every potential error is caught but some “false alarms” may be issued

Analogies with Numerical Analysis

- Approximate the exact semantics
- More precision can be obtained at greater computational costs
  - But sometimes more precise can also be more efficient

Origins of Abstract Interpretation

- [Naur 1965] The Gier Algol compiler
  "`A process which combines the operators and operands of the source text in the manner in which an actual evaluation would have to do it, but which operates on descriptions of the operands, not their value"
- [Reynolds 1969] Interesting analysis which includes infinite domains (context free grammars)
- [Syntzoff 1972] Well foudedness of programs and termination
- [Cousot and Cousot 1976,77,79] The general theory
- [Kamm and Ullman, Kildall 1977] Algorithmic foundations
- [Tarjan 1981] Reductions to semi-ring problems
- [Sharir and Pnueli 1981] Foundation of the interprocedural case
- [Allen, Kennedy, Cock, Jones, Muchnick and Scwartz]

Some Industrial Success Stories

- [Array bound checks for IBM PL.8 Compiler]
- Polyspace
- AbsInt
- Prefix/Intrinsa
- SLAM

Conclusions

- Abstract interpretation is a powerful technique
- Scales to different programming styles
- Allows specifications
- Proving near-commutativity becomes hard for complicated language constructs and abstractions
  - Pointers
  - Destructive updates