

Title	Abstraction of programs in PML (Pointer Manipulation Language)
Author(s)	Takahashi, Koichi; Tanabe, Yoshinori; Sekizawa, Toshifusa; Yuasa, Yoshifumi
Citation	
Issue Date	2009-09-22
Type	Presentation
Text version	publisher
URL	<a href="http://hdl.handle.net/10119/8332">http://hdl.handle.net/10119/8332</a>
Rights	
Description	1st VERITE : JAIST/TRUST-AIST/CVS joint workshop on VERification TEchnologyでの発表資料, 開催 : 2005年9月21日 ~ 22日, 開催場所 : 金沢市文化ホール 3F

# Abstraction of programs in PML (Pointer Manipulation Language)

Koichi Takahashi  
Yoshinori Tanabe  
Toshifusa Sekizawa  
Yoshifumi Yuasa

AIST

22 Sep 2005

# Overview

- Research Interest: Abstraction of graph transformation systems using modal logics.
  - Garbage Collection, Cellular Automata
- Automatic verification tool for pointer manipulation programs
  - Main issue: abstraction of heap
- Use of modal logic to describe heap
  - Seeds for predicate abstraction are described in modal formula
- Development of abstraction tool based on this idea

# Whole picture

**Input**

**Program**

**Requirement  
Property**

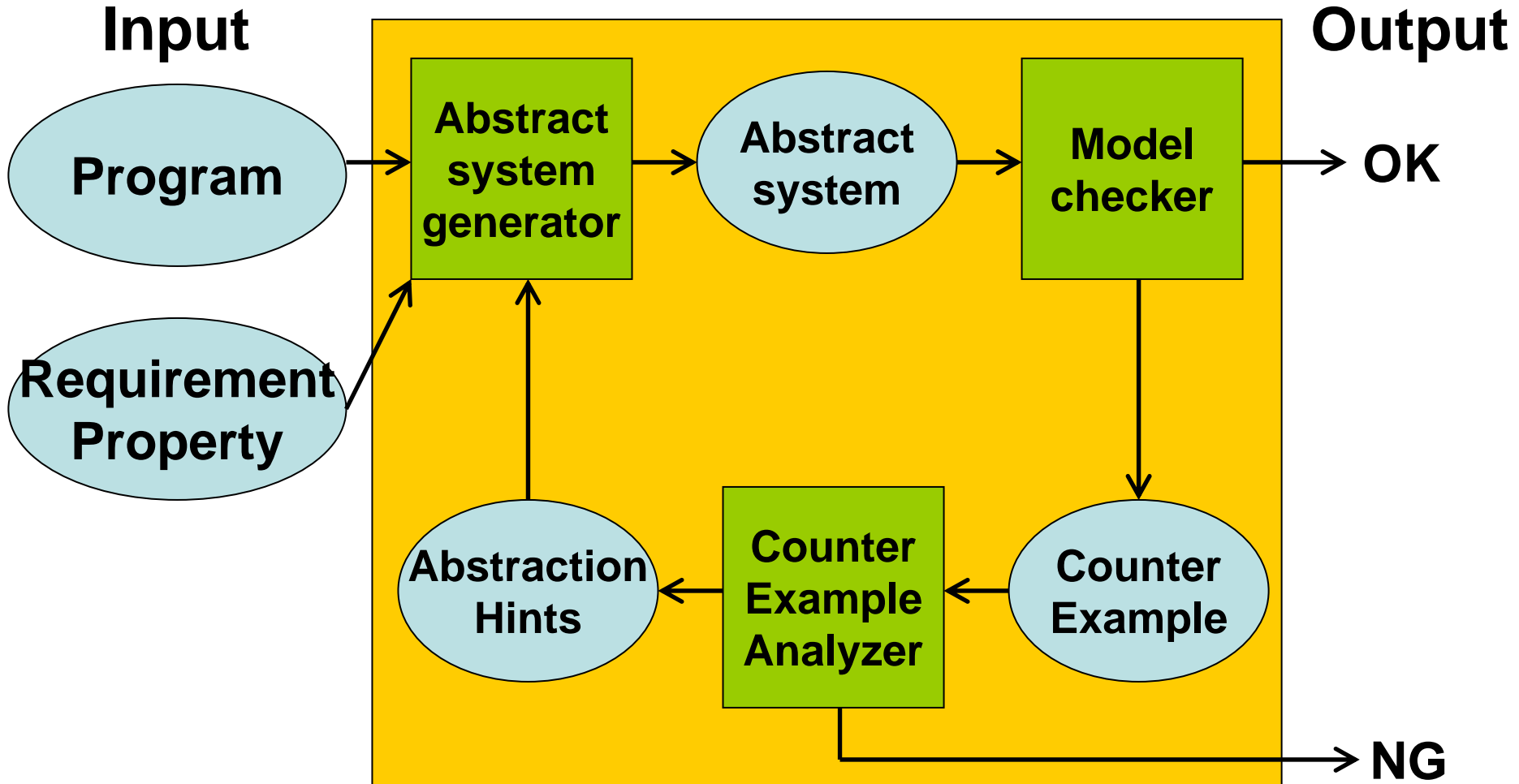
**Output**

**→ OK**

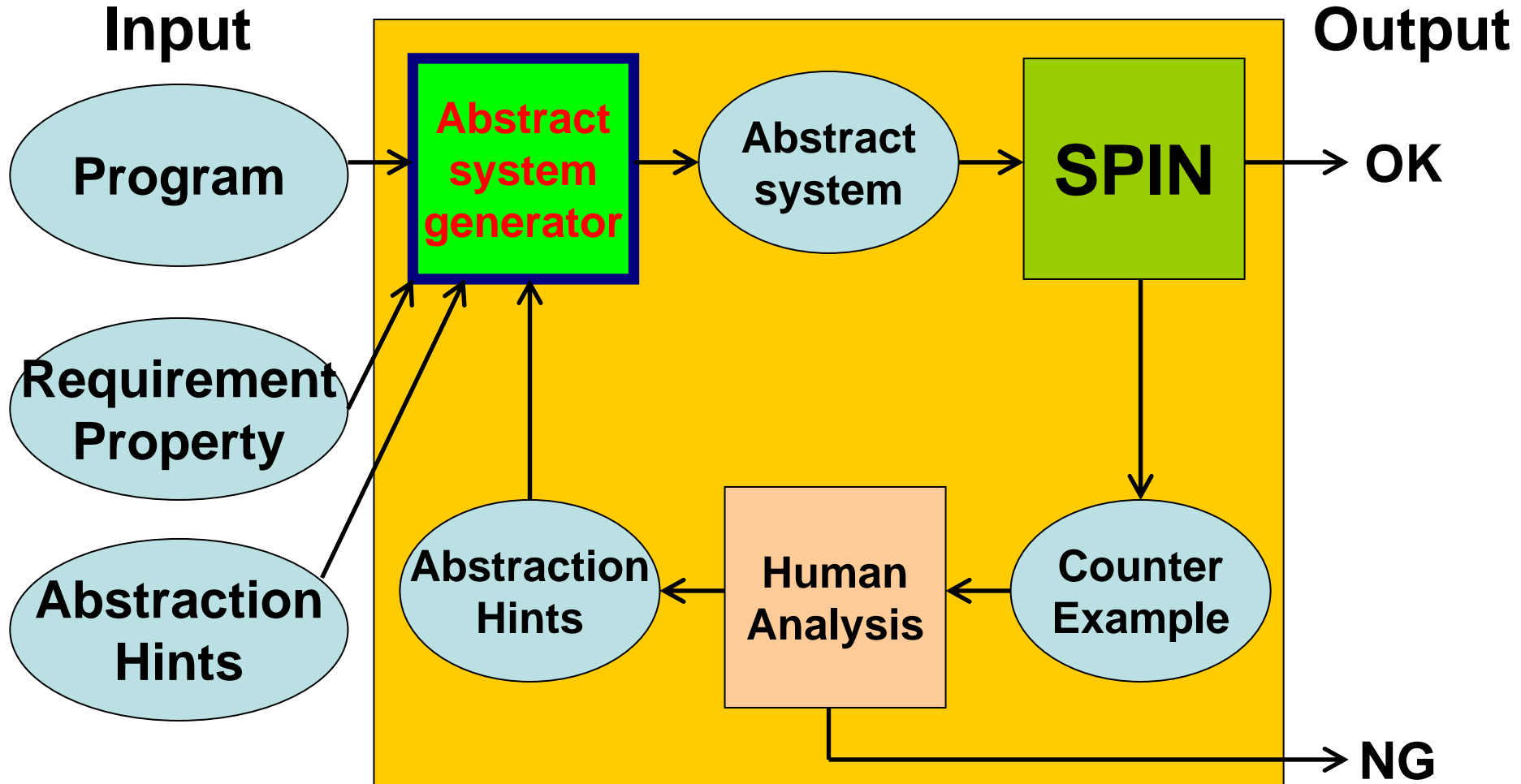
**→ NG**



# Whole picture



# Current Development

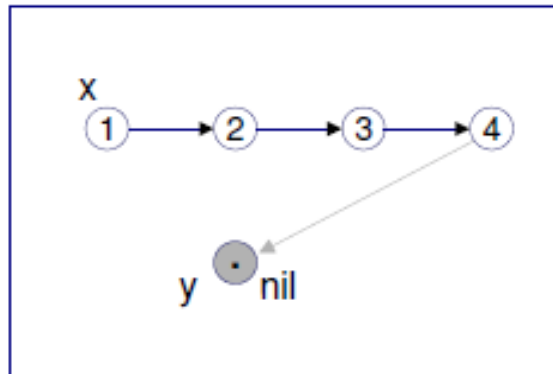


# Idea

- Predicate Abstraction Framework
  - Most of tools developed in the early days handle properties on the value of variables as predicates used in abstraction
  - It was difficult to express properties on the shape of the heap of programs
- We use **modal formulas** as a method for abstracting heap structures
  - another idea: separation logic?

# Model of Heap: Pointer Structure

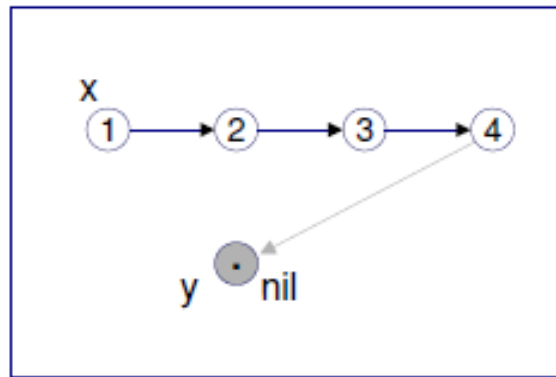
- Heap consists of cells
- Each cell has a pointer and a value
  - to simplify explanation
- Pointer variables





# Pointer Structure as Kripke Structure

- Pointer Structure can be seen as a Kripke structure
- Atomic propositions are values and variables



**AP = {1,2,3,4,x,y,nil}**

# 2CTL

$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{E}_A \mathbf{X} \varphi \mid \mathbf{A}_A \mathbf{X} \varphi$   
 $\mid \mathbf{E}_A \mathbf{F} \varphi \mid \mathbf{A}_A \mathbf{F} \varphi \mid \mathbf{E}_A \mathbf{G} \varphi \mid \mathbf{A}_A \mathbf{G} \varphi$

where

$p$ : atomic proposition,

$A \subseteq \text{Mod}$ : set of modality,

$\bar{a} \in \text{Mod}$ ,

$\bar{\bar{a}} = a$  for  $a \in \text{Mod}$ .

# Properties

- Many properties of heap can be described

- Confluence

$$x \wedge \mathbf{E}_f \mathbf{F} \mathbf{E}_{\bar{f}} \mathbf{F} y$$

- Reachable

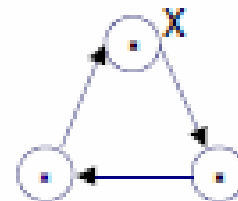
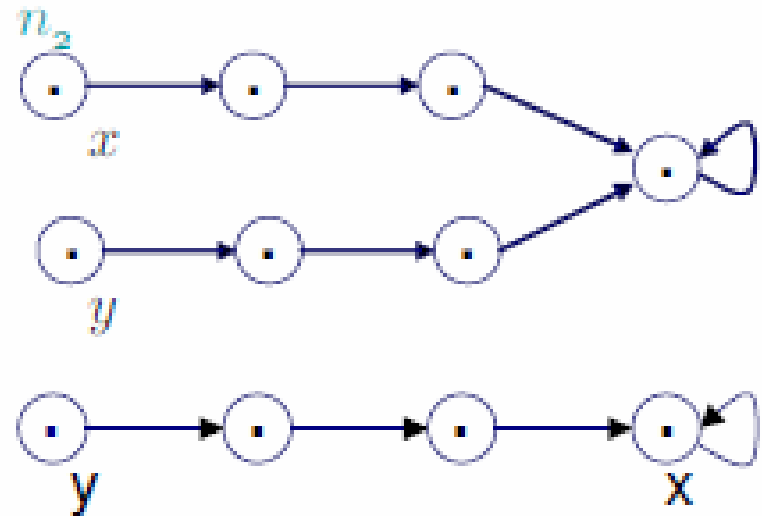
- x is reachable from y

$$y \rightarrow \mathbf{E} \mathbf{F} x$$

- Loop

- x is in loop

$$x \rightarrow \mathbf{E} \mathbf{X} \mathbf{E} \mathbf{F} x$$

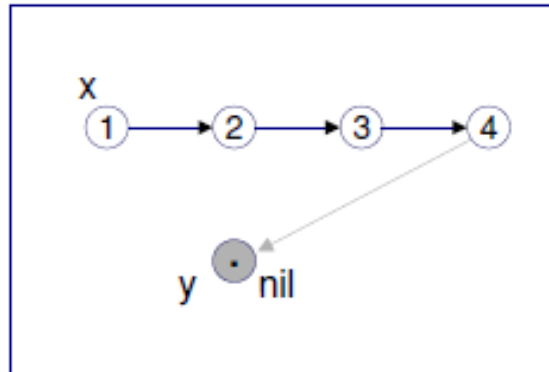


# PML (pointer manipulation language)

- Target programs are written in PML
  - a tiny programming language manipulating heaps
- Statements are following:
  - `x := y`
  - `x := y.next`
  - `x.next := y`
  - `x := new()`
  - `x.val := m`
  - `if (cond) goto line`
- Dynamic logic for PML?
  - ongoing

# a PML program example

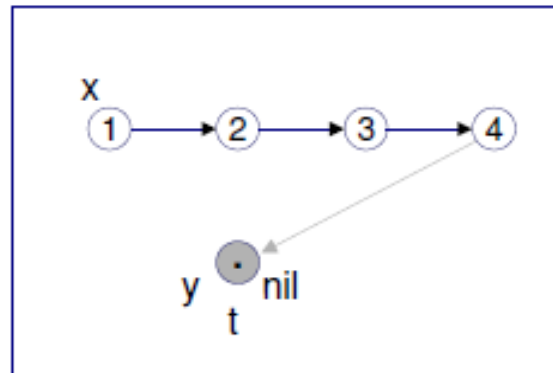
```
0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
```



```
0: y=nil
```

# a PML program example

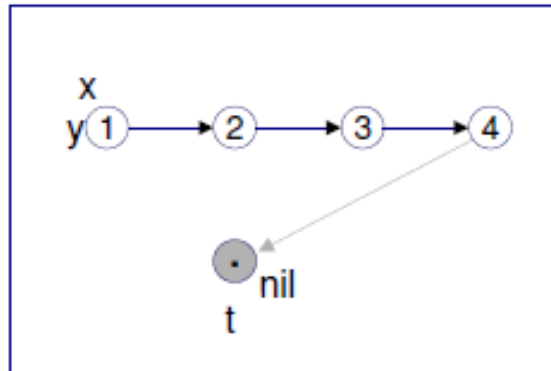
```
0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
```



2: t=y

# a PML program example

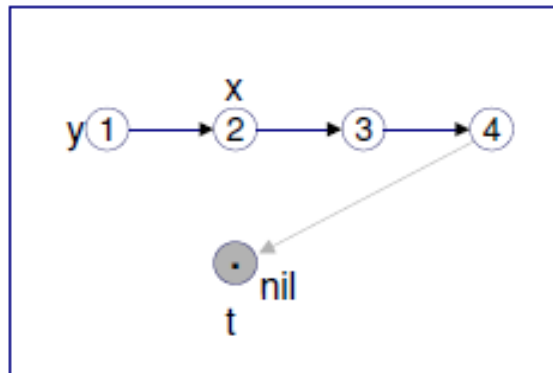
```
0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
```



3: y=x

# a PML program example

```
0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
```

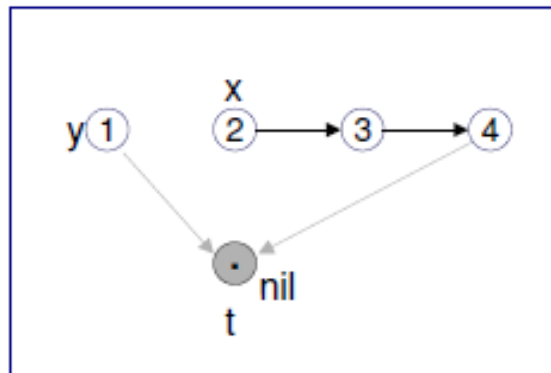


4: x=x.next



# a PML program example

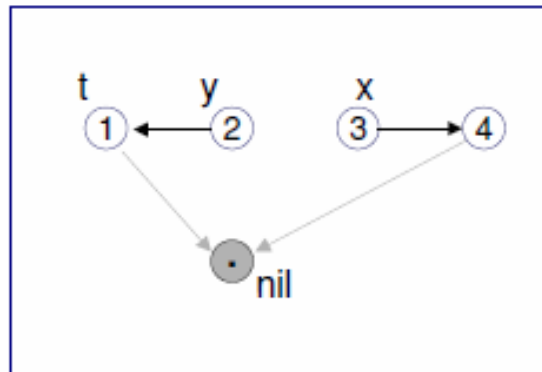
```
0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
```



5: y.next=t

# a PML program example

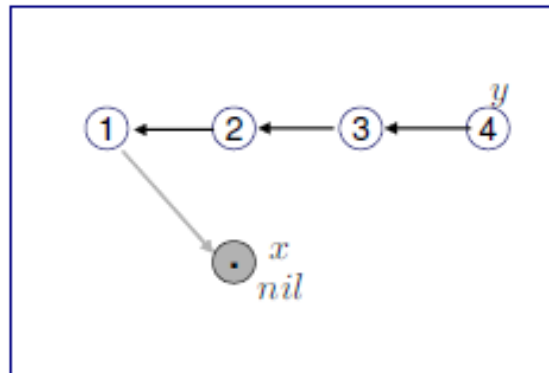
```
0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
```



5: y.next=x

# a PML program example

```
0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
```



7: (end)

# a verification example

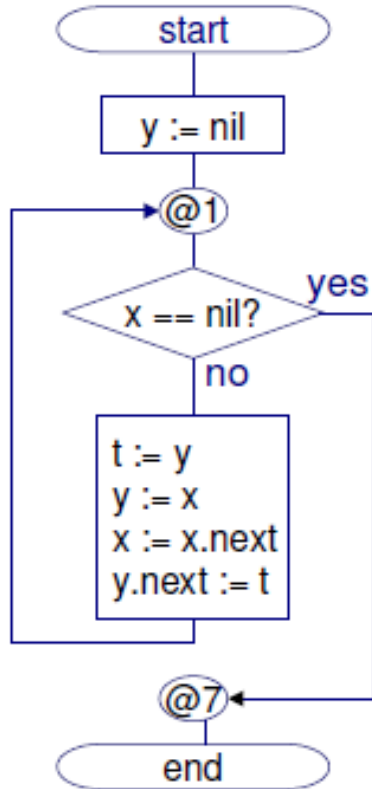
```
0: y := nil
1: if (x == nil) goto 7
2: t := y
3: y := x
4: x := x.next
5: y.next := t
6: goto 1
7: (end)
```

- Verification statement:  
If a node is reachable from  $x$  at line 1,  
then the node is reachable from  $y$  at line 7.

$$Q_1 = x \rightarrow \mathbf{EF} u \quad Q_2 = y \rightarrow \mathbf{EF} u$$

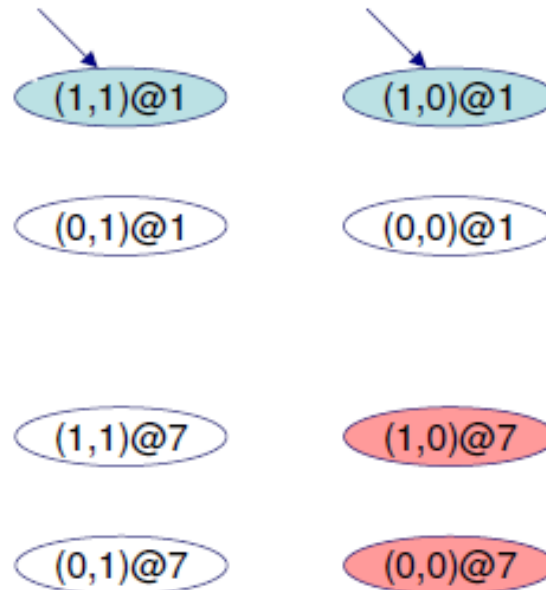
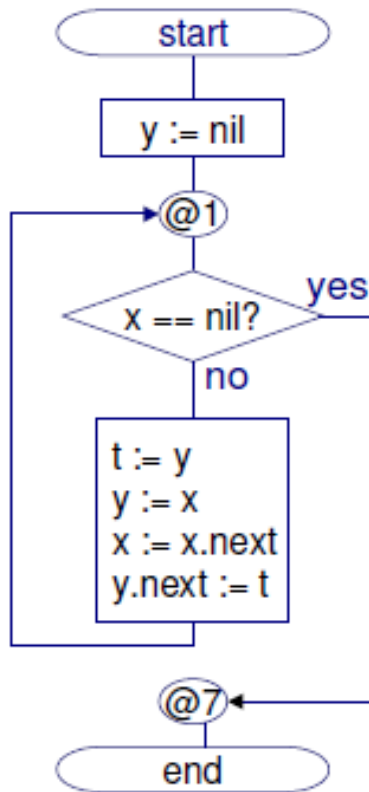
$Q_1$  holds at line 1  $\Rightarrow$   $Q_2$  holds at line 7.

# a verification example



$$Q_1 = x \rightarrow \mathbf{EF} u \quad Q_2 = y \rightarrow \mathbf{EF} u$$

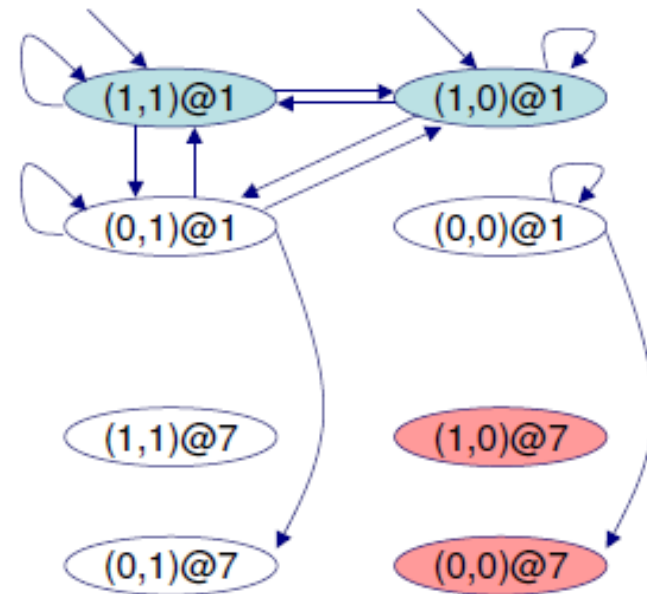
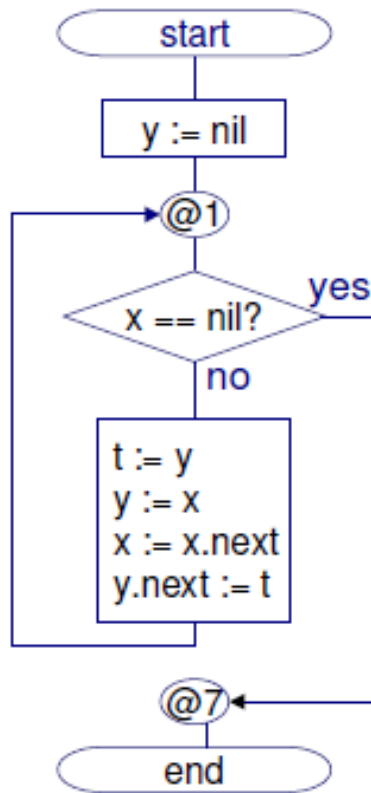
# a verification example



$$Q_1 = x \rightarrow \text{EF } u$$

$$Q_2 = y \rightarrow \text{EF } u$$

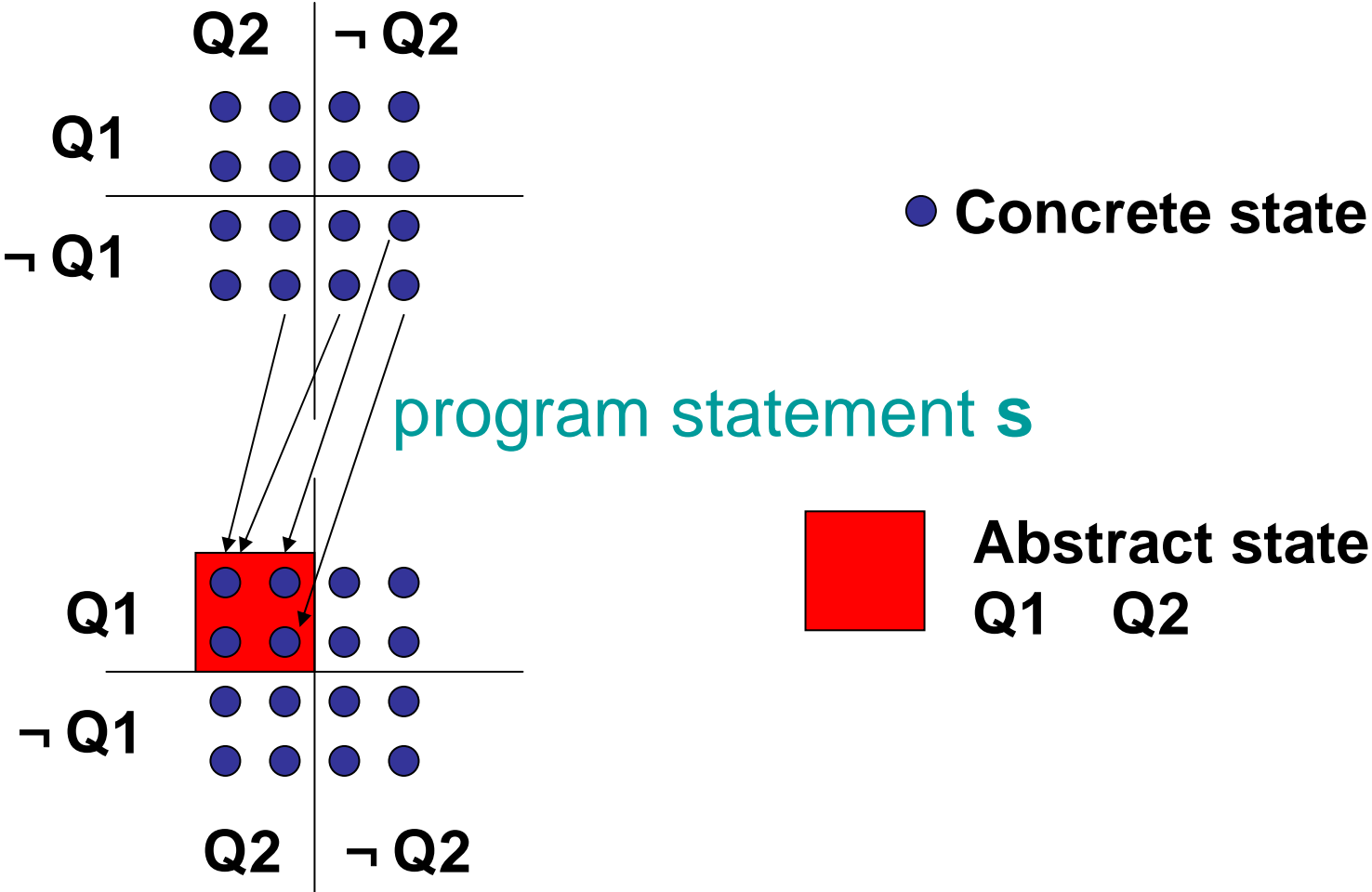
# a verification example



$$Q_1 = x \rightarrow \mathbf{EF} u$$

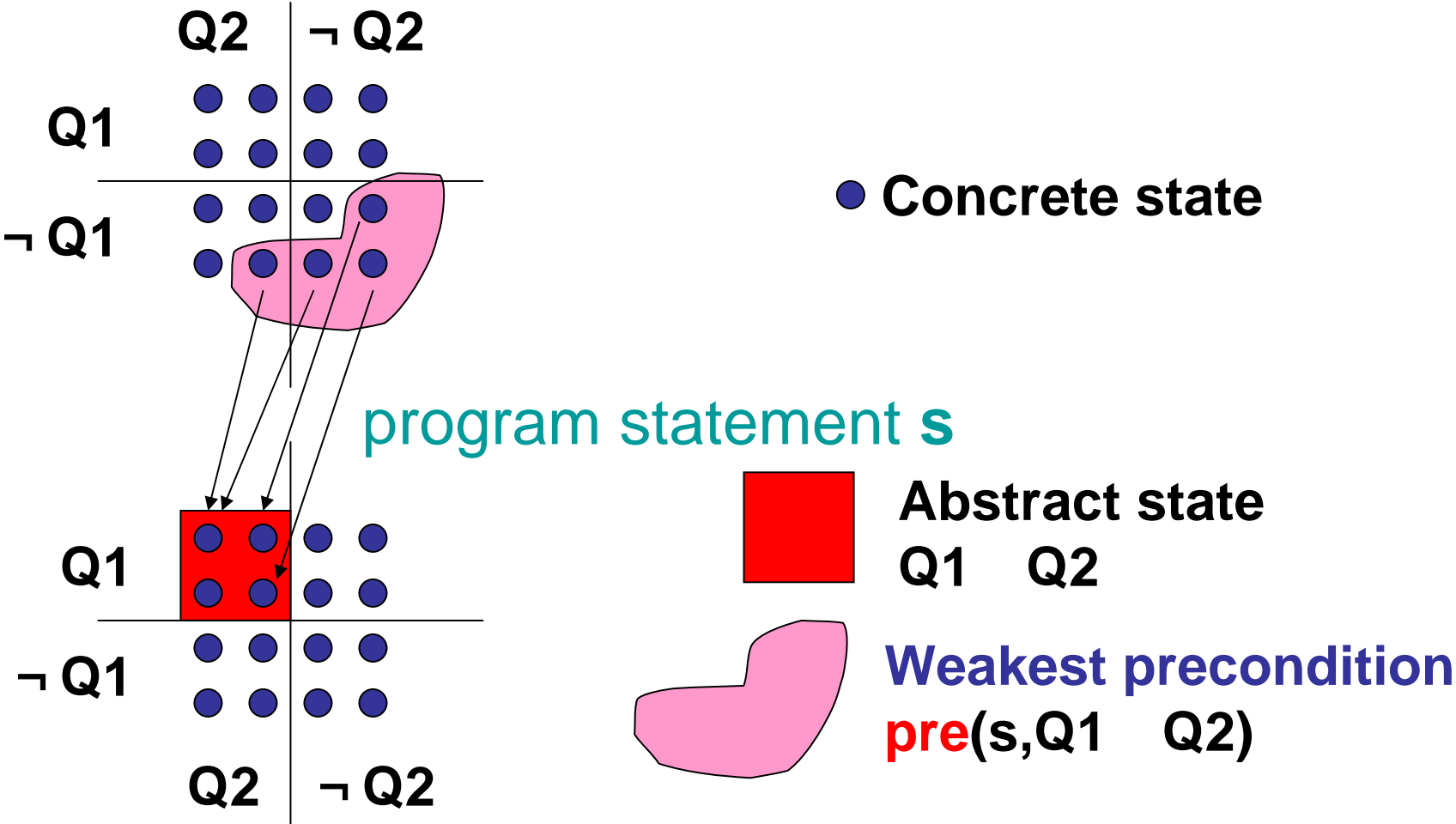
$$Q_2 = y \rightarrow \mathbf{EF} u$$

# Compute abstract transition





# Compute abstract transition



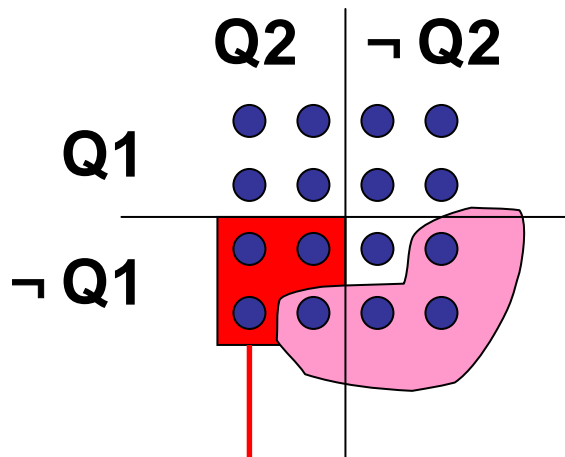
# Compute abstract transition

## Abstract transition

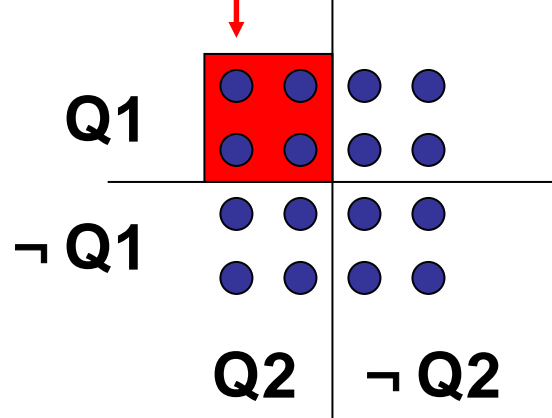
### Intersection

$$\text{sat}(\neg Q1 \quad Q2 \quad \text{pre}(s, Q1 \quad Q2)) = 1$$

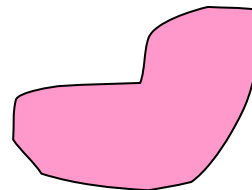
● Concrete state



program statement s



Abstract state  
Q1 Q2



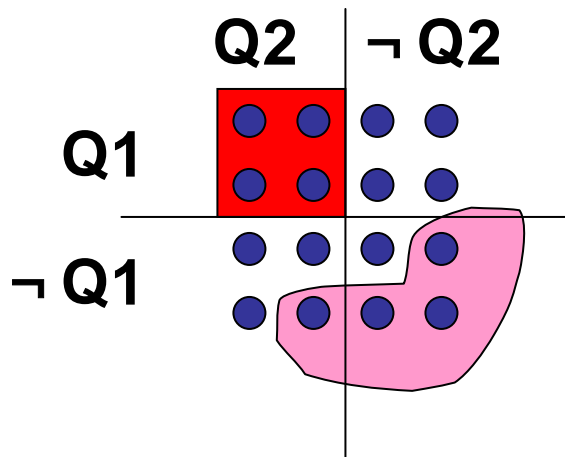
Weakest precondition  
pre(s, Q1 Q2)

# Compute abstract transition

No abstract transition

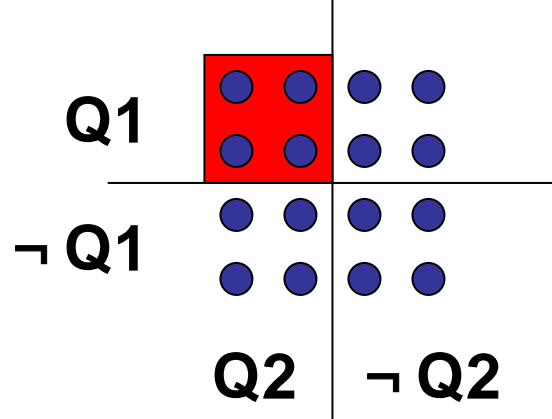
Disjoint

$$\text{sat}(Q1 \ \ Q2 \ \ \text{pre}(s, Q1 \ \ Q2)) = 0$$

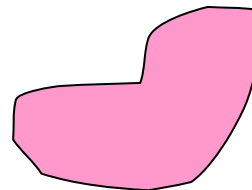


● Concrete state

program statement  $s$



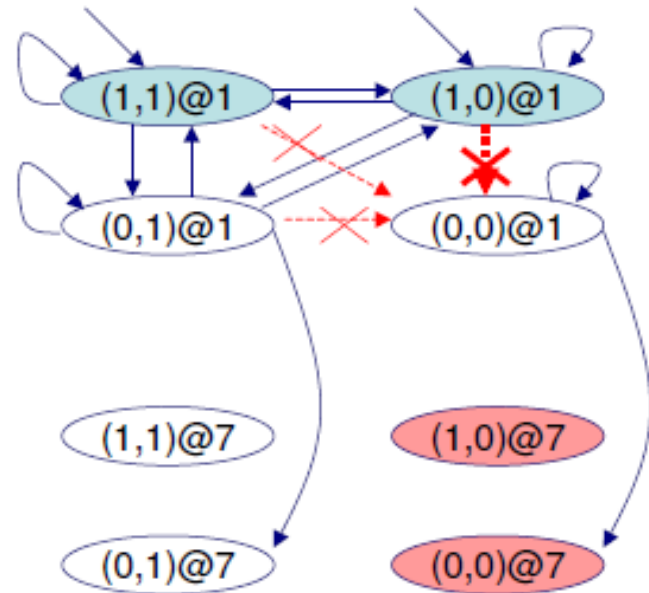
Abstract state  
 $Q1 \ \ Q2$



Weakest precondition  
 $\text{pre}(s, Q1 \ \ Q2)$

# Our case

- Weakest precondition of **2CTL** for **PML**
- Sat checker for **2CTL**



$$Q_1 = \forall(x \rightarrow \mathbf{EF} u)$$

$$\neg Q_2 = \forall(y \rightarrow \neg \mathbf{EF} u)$$

$$\text{pre}(s, \neg Q_1) =$$

$$\forall(x \rightarrow \mathbf{EF} \mathbf{EX}x) \wedge \forall(u \rightarrow \neg(x \vee \mathbf{E} (\neg x \mathbf{U} (\neg x \wedge (\mathbf{EX}x \vee y))))))$$

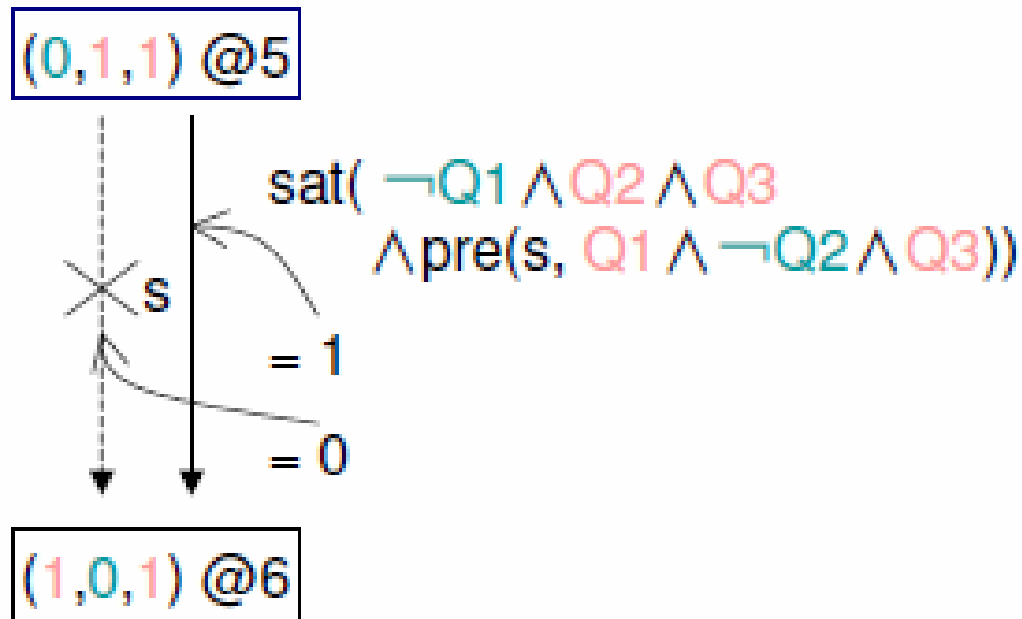
$$\vee \forall(x \rightarrow \neg \mathbf{EF} \mathbf{EX}x) \wedge \forall(u \rightarrow \neg \mathbf{EF} \mathbf{EX}x)$$

$$\text{pre}(s, \neg Q_2) = \forall(u \rightarrow \neg(x \vee \mathbf{E} (\neg x \mathbf{U} (\neg x \wedge y))))$$

$$(s = "t:=y; y:=x; x:=x.next; y.next:=t")$$

# Compute abstract transition

- Precondition and
- Satisfiability checking



# Precondition

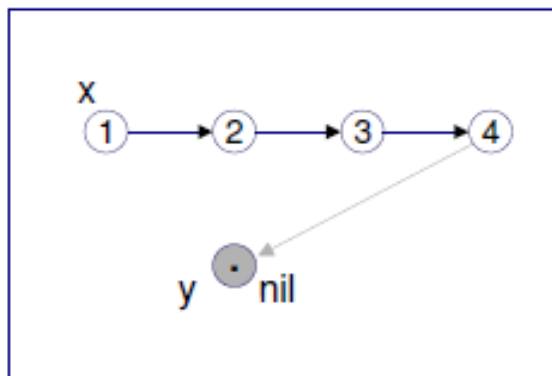
- We restrict 2CTL to p-formula for simplification
  - It is enough to describe important properties
- We have calculated weakest precondition of p-formula for each PML statement
  - weakest precondition of p-formula is p-formula
  - We are now implementing

# Satisfiability check

- Usual:  $\varphi$  is sat there exists a Kripke structure  $K$  s.t.  $K \models \varphi$ 
  - this checking is too rough
  - previous verification example does not work

# Pointer Structure as Kripke Structure

- Pointer Structure can be seen as a Kripke structure
- Atomic propositions are values and variables



$AP = \{1,2,3,4,x,y,nil\}$

- **Variable property holds at most one node**
- **A node has at most one next node**



# Satisfiability check

- Usual:  $\varphi$  is sat there exists a Kripke structure  $K$  s.t.  $K \models \varphi$
- Our modification:  $\varphi$  is sat there exists a **Pointer structure**  $P$  s.t.  $P \models \varphi$ 
  - more accurate
    - previous verification example works
  - We are now implementing
    - BDD

# Current Development

