Preconditions of properties described in CTL for statements manipulating pointers

June 29, 2005

Workshop 1, DSN-2005

Yoshinori TANABE[1,2], Toshinori TAKAI[1,2],
Toshifusa SEKIZAWA[1,2] and Koichi TAKAHASHI[1]

1: Research Center for Verification and Semantics,
National Institute of Advanced Industrial Science and Technology
2: Japan Science and Technology Agency / CREST

Overview

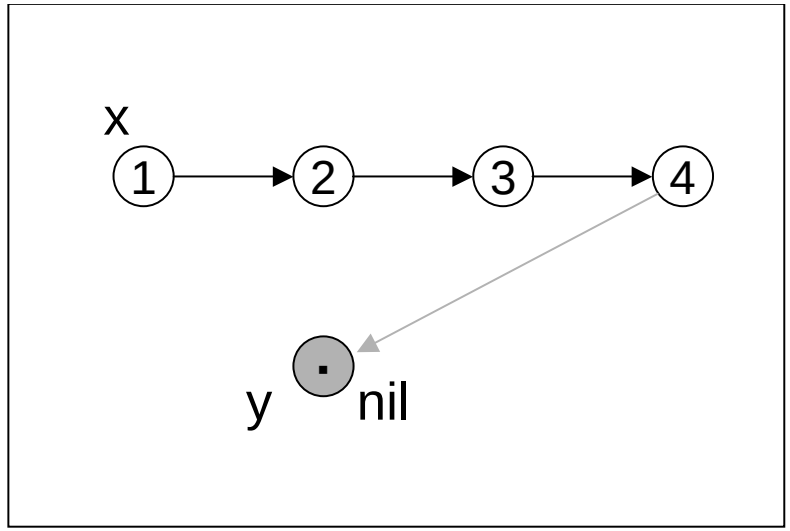
- We are working in an ongoing project that develops a method that verifies properties on graph rewriting systems such as program heaps; and a tool based on the method.
- The framework our method is based upon is Predicate Abstraction, which, from an infinite transition system (TS), creates a finite TS as a sound abstraction, thus we can model-check it to verify the original property.
- We use Temporal Logic formulas to express predicates.
- We need:
  - Calculation of preconditions of predicates for each transition: Done. This talk.
  - Decision procedure to judge satisfiability of predicates: we are working on the issue.

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)
```

Example

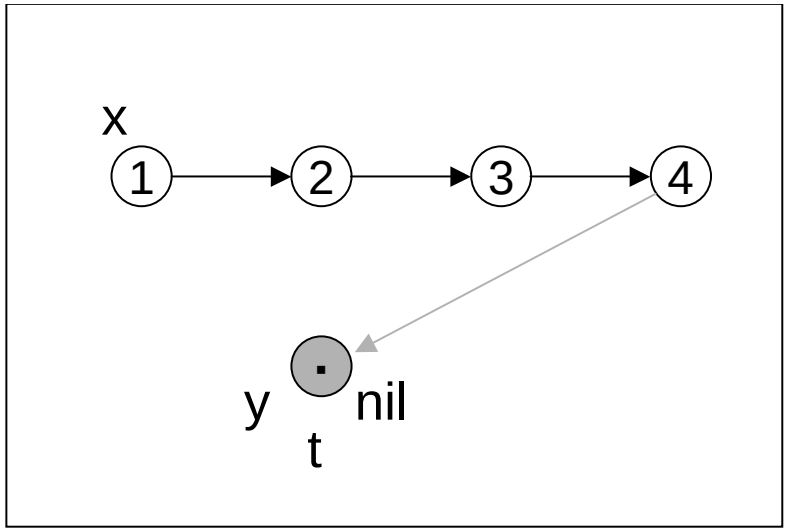
```
0: y := nil
1: if (x == nil) goto 7
2:   t := y
3:   y := x
4:   x := x.next
5:   y.next := t
6:   goto 1
7: (end)
```



0: y=nil

Example

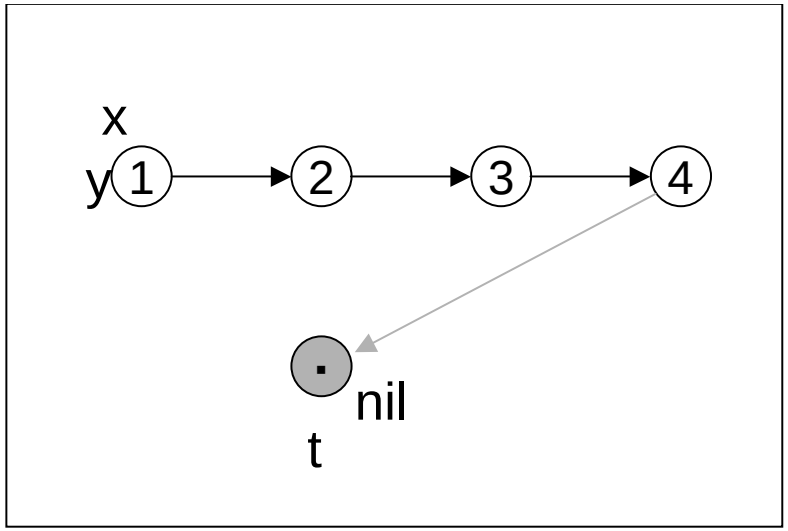
```
0: y := nil
1: if (x == nil) goto 7
2:   t := y
3:   y := x
4:   x := x.next
5:   y.next := t
6:   goto 1
7: (end)
```



2: t=y

Example

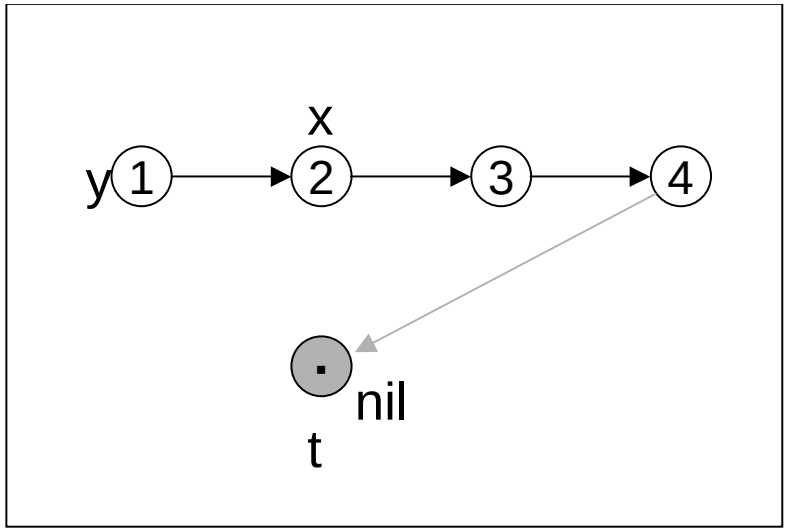
```
0: y := nil
1: if (x == nil) goto 7
2:   t := y
3:   y := x
4:   x := x.next
5:   y.next := t
6:   goto 1
7: (end)
```



3: y=x

Example

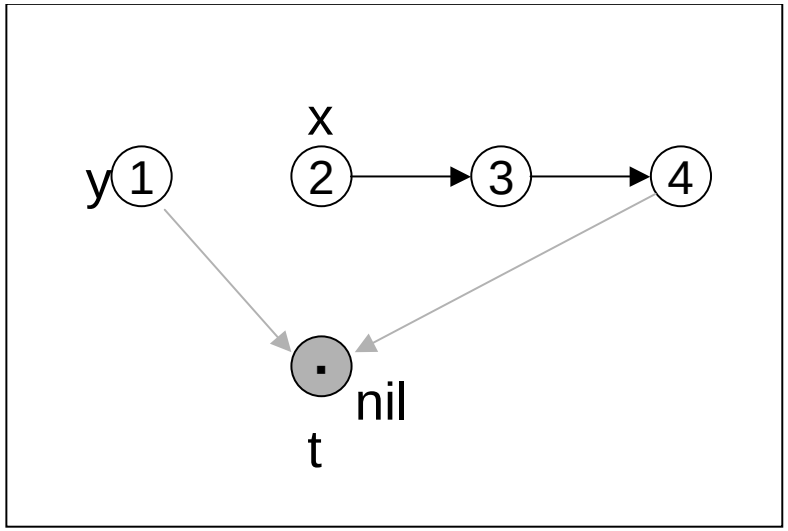
```
0: y := nil
1: if (x == nil) goto 7
2:   t := y
3:   y := x
4:   x := x.next
5:   y.next := t
6:   goto 1
7: (end)
```



4: x=x.next

Example

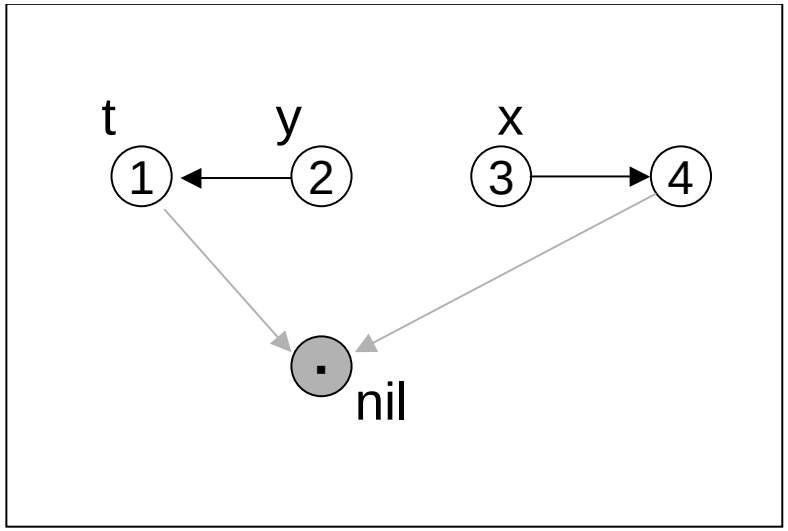
```
0: y := nil
1: if (x == nil) goto 7
2:   t := y
3:   y := x
4:   x := x.next
5:   y.next := t
6:   goto 1
7: (end)
```



5: y.next=t

Example

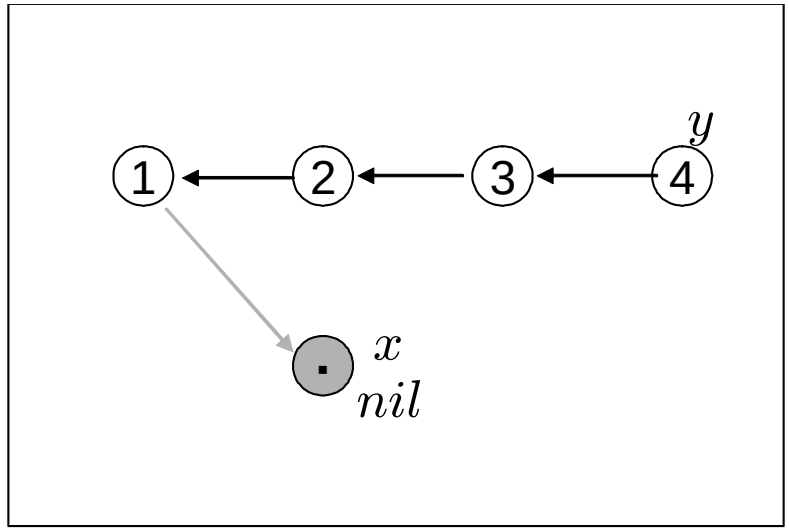
```
0: y := nil
1: if (x == nil) goto 7
2:   t := y
3:   y := x
4:   x := x.next
5:   y.next := t
6:   goto 1
7: (end)
```




```
5:  y.next=x
```

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)
```

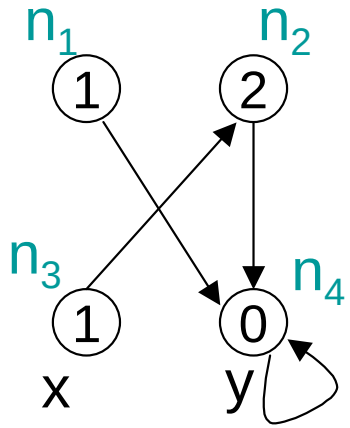
Pointer Structure

Var: the set of variables

Val: the set of values

pointer structure: tuple $S = (N, p, v, \rho)$ s.t.

- N is a finite set of nodes.
- $p : N \longrightarrow N$
- $v : N \longrightarrow \mathsf{Val}$
- $\rho : \mathsf{Var} \longrightarrow N$



$N = \{n_1, n_2, n_3, n_4\}$, $\mathsf{Var} = \{x, y\}$,
$p(n_1) = n_4$, $v(n_2) = 2$, $\rho(x) = n_3$,
etc

Pointer Manipulation Language (PML)

- Assignment Statements
  ($x, y \in$ Var, $m \in$ Val)
  - `x := y`
  - `x := y.next`
  - `x.next := y`
  - `x := new()`
  - `x.val := m`
- Conditional Goto Statements
  - `if (cond) goto` *line*

CTL

CTL formula φ:

$$\varphi = p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{EX}\varphi \mid \mathbf{AX}\varphi$$
$$\mid \mathbf{E}\,(\varphi\,\mathbf{U}\,\varphi) \mid \mathbf{A}\,(\varphi\,\mathbf{U}\,\varphi) \mid \mathbf{E}\,(\varphi\,\mathbf{R}\,\varphi) \mid \mathbf{A}\,(\varphi\,\mathbf{R}\,\varphi)$$
$$(p \in \mathsf{Var} \cup \mathsf{Val})$$

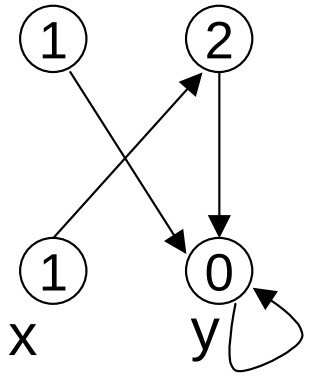
Abbrev: $\mathbf{EF}\,\varphi = \mathbf{E}\,(\mathsf{true}\,\mathbf{U}\,\varphi)$ etc

Form := the set of CTL formulas.

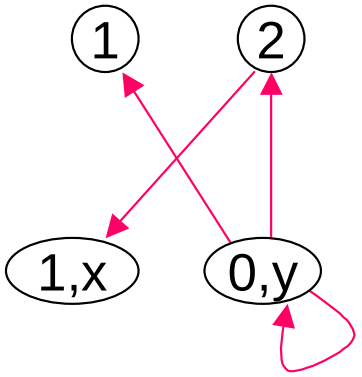
Kripke Structure and Pointer Structure

Kripke structure $K(S) = (N, R, L)$ induced by a pointer structure $S = (N, p, v, \rho)$.

- $R = \{(n, n') \mid p(n') = n\}$ reverse direction
- $L(x) = \{\rho(x)\}$ $(x \in \mathsf{Var})$
- $L(m) = \{n \in N \mid v(n) = m\}$ $(m \in \mathsf{Val})$



S



$K(S)$

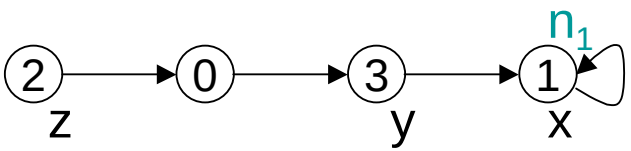
Properties described in CTL

$K(S_1), n_1 \models x$

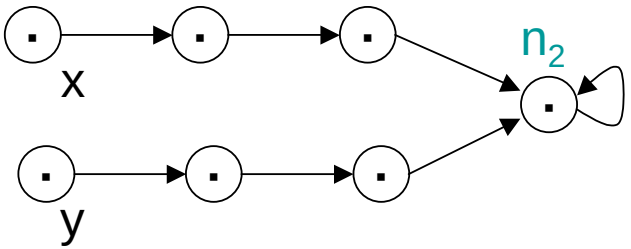
$K(S_1), n_1 \models 1$

$K(S_1), n_1 \models \mathbf{EX}y$

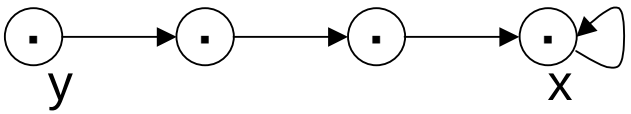
$K(S_1), n_1 \models \mathbf{EF}\ z$



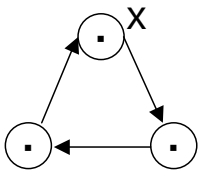
$K(S_2), n_2 \models \mathbf{EF}\ x \wedge \mathbf{EF}\ y$
(confluence)



$K(S_3) \models x \rightarrow \mathbf{EF}\ y$
(x is reachable from y.)



$K(S_4) \models x \rightarrow \mathbf{EXEF}\ x$
(x is in a cycle.)



Predicates - p-formulas

We define predicates for our abstraction method. First we define syntactically p-formula:

1. $\forall\varphi$ is a p-formula for a CTL formula φ.
2. $\neg Q_1$ and $Q_1 \vee Q_2$ are p-formulas for p-formulas Q_1 and Q_2.

Abbrev: $Q_1 \wedge Q_2$, $Q_1 \to Q_2$, $Q_1 \leftrightarrow Q_2$, $\exists\varphi (= \neg\forall\neg\varphi)$

Example of p-formula: $\forall(0) \vee \forall(x \to \mathbf{AX}1)$

cf: $\forall(0 \vee (x \to \mathbf{AX}1))$

A p-formula Q induces a predicate $\cdot \models^* Q$ on PtrStr:

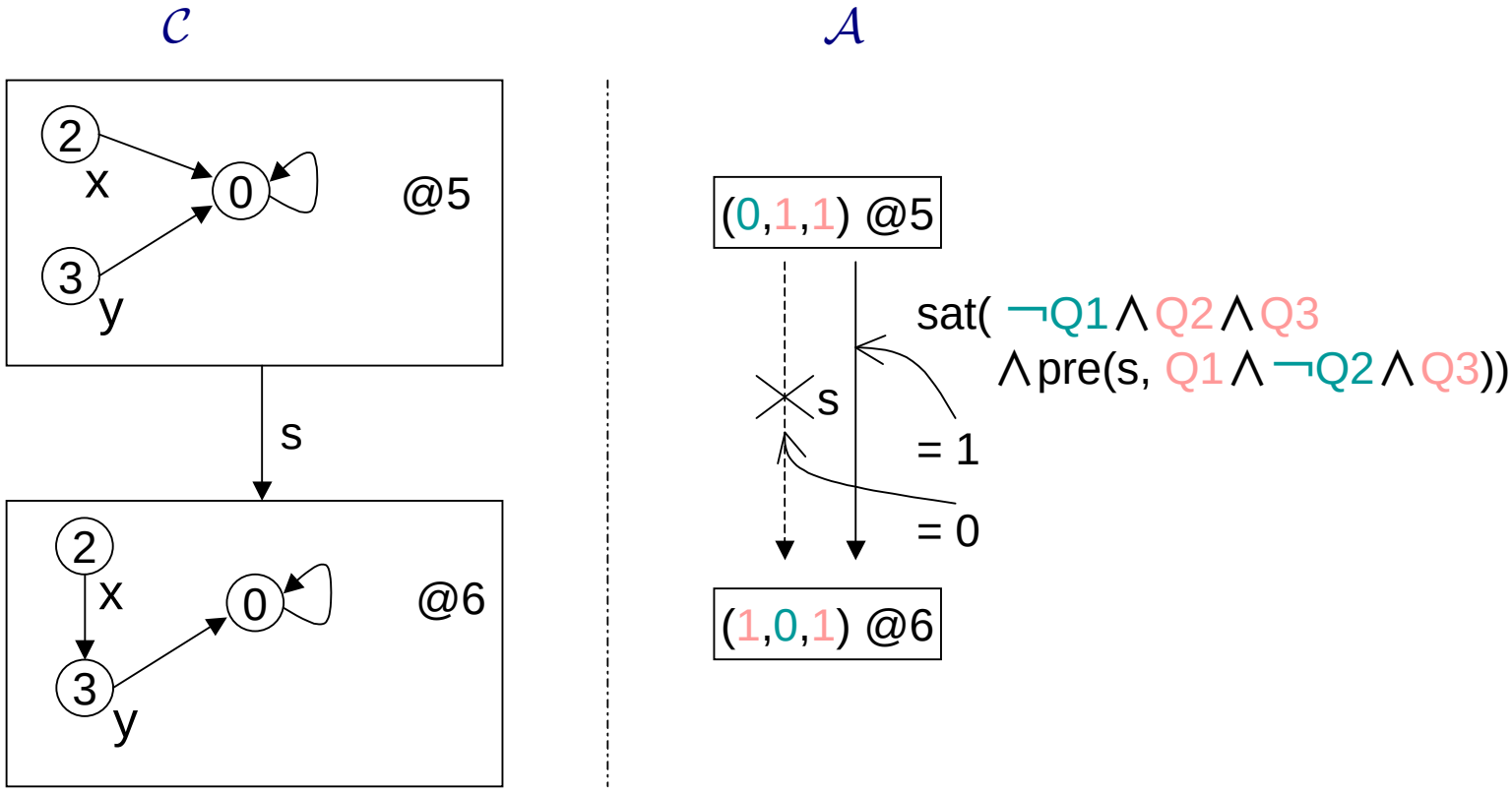
- $S \models^* \forall\varphi \iff$ For all $n \in N$ s.t. $K(S), n \models \varphi$.
- $S \models^* \neg Q \iff S \not\models^* Q$.
- $S \models^* Q_1 \vee Q_2 \iff S \models^* Q_1$ or $S \models^* Q_2$.

Predicate Abstraction

P: given PML program.

$Q_1, \ldots, Q_n$: selected p-formulas.

e.g. $Q_1 = \forall(y \to \mathbf{EX}x)$, $Q_2 = \forall(0 \to \mathbf{EX}2)$, $Q_3 = \forall(x \to 2)$



$s =$ "`x.next:=y`" at line 5

Predicate Abstraction (cont)

Two functions used to define $\rightarrow_{\mathcal{A}}$ should satisfy:

- pre : AStmt $\times$ PForm $\rightarrow$ PForm

$$[\![s]\!](S) \models^{*} Q \implies S \models^{*} \mathsf{pre}(s, Q).$$

- sat : PForm $\rightarrow \{0, 1\}$

$$Q \text{ is satisfiable} \implies \mathsf{sat}(Q) = 1.$$

FACT

The transition system $\mathcal{A}$ thus defined is a sound abstraction of $\mathcal{C}$ with respect to ACTL* properties.

Preconditions

We will define pre : AStmt $\times$ PForm $\to$ PForm (in the following slides).

Theorem

For $S \in$ PtrStr, $s \in$ AStmt and $Q \in$ AStmt:

$$[\![s]\!](S) \models^{*} Q \iff S \models^{*} \mathrm{pre}(s, Q)$$

Note1: the requirement for pre was:

$$[\![s]\!](S) \models^{*} Q \implies S \models^{*} \mathrm{pre}(s, Q)$$

Note2: Our function pre gives the weakest precondition in the ordinary sense. The Hoare triple

$$\{\mathrm{pre}(s, Q)\}\, s\, \{Q\}$$

holds.

- $\mathsf{pre}(s, \neg Q_1) = \neg\mathsf{pre}(s, Q_1)$
- $\mathsf{pre}(s, Q_1 \vee Q_2) = \mathsf{pre}(s, Q_1) \vee \mathsf{pre}(s, Q_2)$

For $s \neq x\texttt{.next:=}y$, we have $\mathsf{pre}(s, \forall(\varphi)) = \forall(\psi)$:

s	ψ
$x\texttt{:=}y$	$\varphi[y/x]$
$x\texttt{:=}y\texttt{.next}$	$\varphi[\mathbf{EX}y/x]$
$x\texttt{.val:=}m$	$\varphi[m \vee x/m][n \wedge \neg x/n]_{n \in \mathsf{Val}\setminus\{m\}}$
$x\texttt{:=new()}$	$\begin{cases} \varphi[\mathsf{false/x}] & \text{if } \mathsf{satnew}(\varphi) \text{ holds} \\ \mathsf{false} & \text{otherwise} \end{cases}$

satnew is defined as follows:

- satnew$(v) \iff v = x$. (for $v \in$ Var)
- satnew$(m) \iff m = 0$. (for $m \in$ Val)
- satnew$(\neg\varphi) \iff$ satnew(φ) does not hold.
- satnew$(\varphi_1 \vee \varphi_2) \iff$ satnew(φ_1) or satnew(φ_2).
- satnew$(\mathbf{EX}\varphi) \iff$ satnew(φ).
- satnew$(\mathbf{E}\,[\varphi_1\,\mathbf{U}\,\varphi_2]) \iff$ satnew(φ_2).
- satnew$(\mathbf{E}\,[\varphi_1\,\mathbf{R}\,\varphi_2]) \iff$ satnew(φ_2).

Inductive definitions for $s = x.\text{next}{:=}y$. For given CTL formula φ, we construct p-formula Q and CTL formula φ_1 and φ_2 such that $\text{pre}(s, \forall(\varphi)) = (Q \wedge \forall(\varphi_1)) \vee (\neg Q \wedge \forall(\varphi_2))$.

For simplicity here we assume that for formula ψ that appears as the induction hypothesis either Q and $\neg Q$ is valid and $\text{pre}(s, \forall(\psi)) = \forall(\overline{\psi})$. See the paper for the general case.

φ	Q	φ_1	φ_2
v ($v \in \text{Var}$)	true	v	—
m ($m \in \text{Val}$)	true	m	—
$\neg\psi_1$	true	$\neg\overline{\psi_1}$	—
$\psi_1 \vee \psi_2$	true	$\overline{\psi_1} \vee \overline{\psi_2}$	—
$\mathbf{EX}\psi_1$	$\forall(x \to p_{\mathsf{EX}}(\overline{\psi_1}))$	$f^{\top}_{\mathsf{EX}}(\overline{\psi_1})$	$f^{\bot}_{\mathsf{EX}}(\overline{\psi_1})$
$\mathbf{E}\,(\psi_1\ \mathbf{U}\ \psi_2)$	$\forall(x \to p_{\mathsf{EU}}(\overline{\psi_1}, \overline{\psi_2}))$	$f^{\top}_{\mathsf{EU}}(\overline{\psi_1}, \overline{\psi_2})$	$f^{\bot}_{\mathsf{EU}}(\overline{\psi_1}, \overline{\psi_2})$
$\mathbf{E}\,(\psi_1\ \mathbf{R}\ \psi_2)$	$\forall(x \to p_{\mathsf{ER}}(\overline{\psi_1}, \overline{\psi_2}))$	$f^{\top}_{\mathsf{ER}}(\overline{\psi_1}, \overline{\psi_2})$	$f^{\bot}_{\mathsf{ER}}(\overline{\psi_1}, \overline{\psi_2})$

- $p_{\mathsf{EX}}(\psi_1) = \psi_1$,
  $f^{\top}_{\mathsf{EX}}(\psi_1) = y \vee \mathbf{EX}(\neg x \wedge \psi_1)$,
  $f^{\bot}_{\mathsf{EX}}(\psi_1) = \mathbf{EX}\psi_1$.
- $p_{\mathsf{EU}}(\psi_1, \psi_2) = \mathbf{E}\,[\psi_1\ \mathbf{U}\ \psi_2]$,
  $f^{\top}_{\mathsf{EU}}(\psi_1, \psi_2) = x \vee \mathbf{E}\,[(\neg x \wedge \psi_1)\ \mathbf{U}\ (\neg x \wedge \psi_2)] \vee \mathbf{E}\,[(\neg x \wedge \psi_1)\ \mathbf{U}\ (\neg x \wedge (y \wedge \psi_1))]$,
  $f^{\bot}_{\mathsf{EU}}(\psi_1, \psi_2) = \mathbf{E}\,[\psi_1\ \mathbf{U}\ \psi_2]$.
- $q_{\mathsf{ER}}(\psi_1, \psi_2, v_1, v_2) = v_1 \vee \mathbf{E}\,[\{v_2 \vee \psi_1 \vee (\mathbf{EX}x \wedge \mathbf{AX}x)\}\ \mathbf{R}\ \{\psi_2 \wedge \neg x\}]$,
  $p_{\mathsf{ER}}(\psi_1, \psi_2) = \psi_2 \wedge [\psi_1 \vee y \vee \mathbf{AX}\bot \vee \mathbf{EX}q_{\mathsf{ER}}(\psi_1, \psi_2, \bot, y)]$,
  $f^{\top}_{\mathsf{ER}}(\psi_1, \psi_2) = q_{\mathsf{ER}}(\psi_1, \psi_2, x, y)$,
  $f^{\top}_{\mathsf{ER}}(\psi_1, \psi_2) = q_{\mathsf{ER}}(\psi_1, \psi_2, \bot, \bot)$.

Example

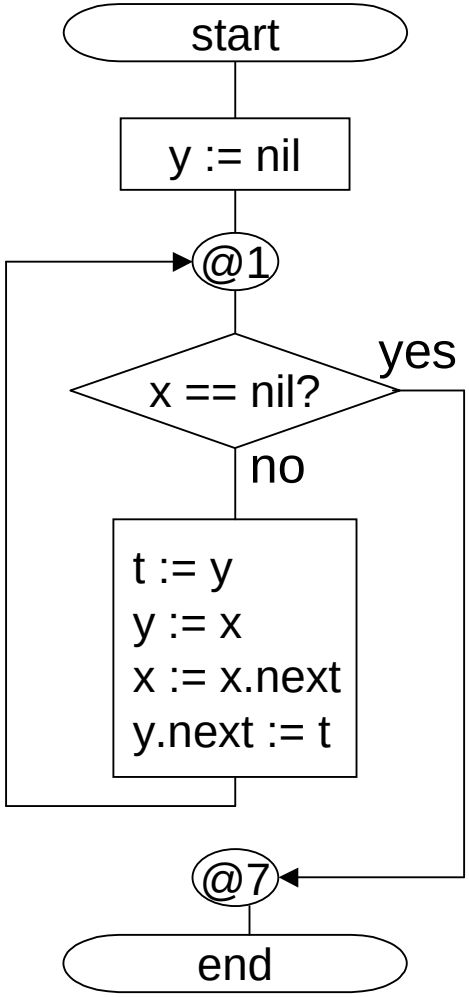
```
0: y := nil
1: if (x == nil) goto 7
2:   t := y
3:   y := x
4:   x := x.next
5:   y.next := t
6:   goto 1
7: (end)
```

"If a node is reachable from x when the program starts, it will be reachable from y when the program ends."

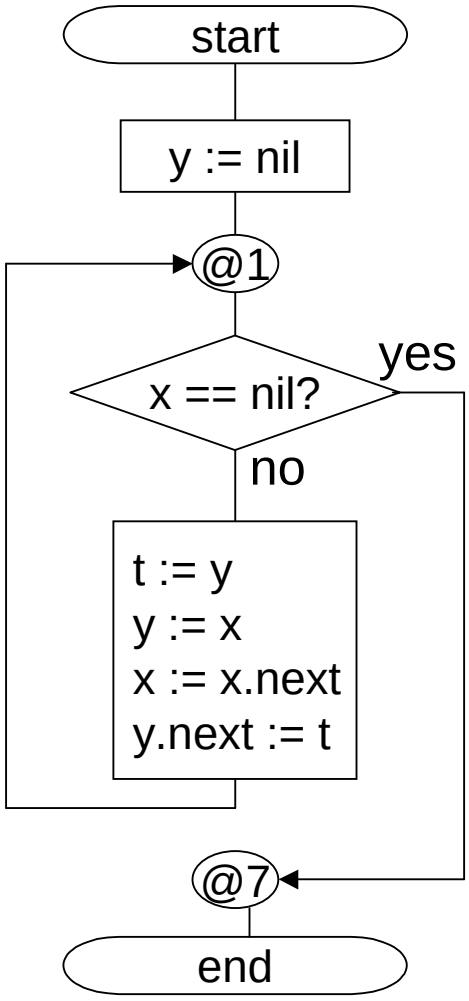
p-formulas to use: (u is a dummy variable.)

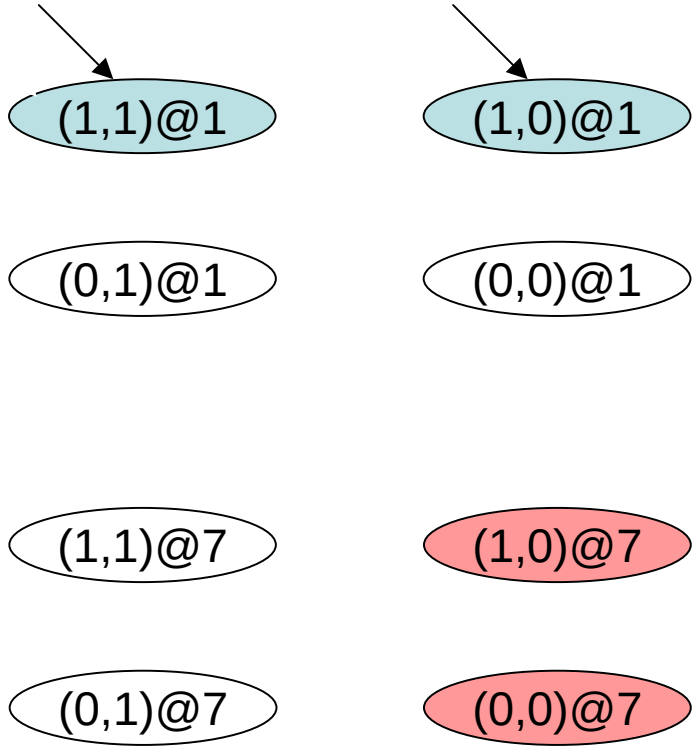
- $Q_1 = \forall(u \rightarrow \mathbf{EF}\, x)$: u is reachable from x.
- $Q_2 = \forall(u \rightarrow \mathbf{EF}\, y)$: u is reachable from y.

To verify: Q_1 holds at line 1 $\implies$ Q_2 holds at line 7.

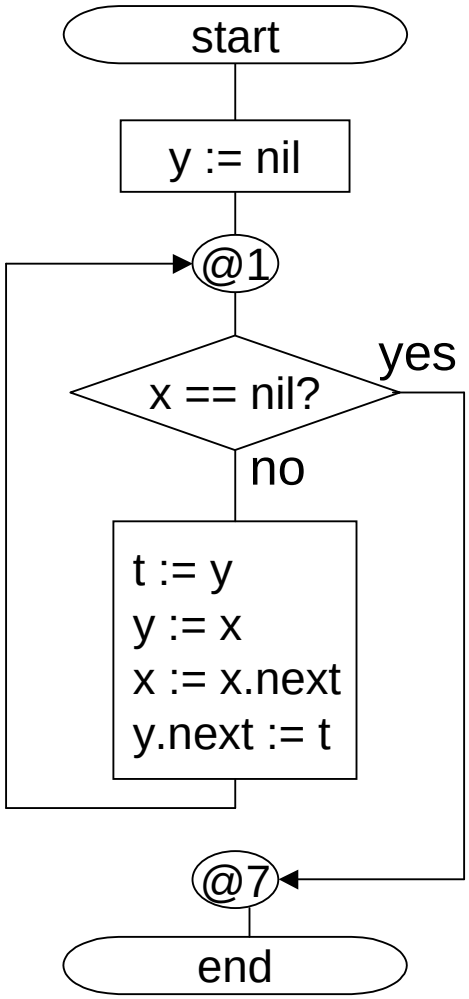


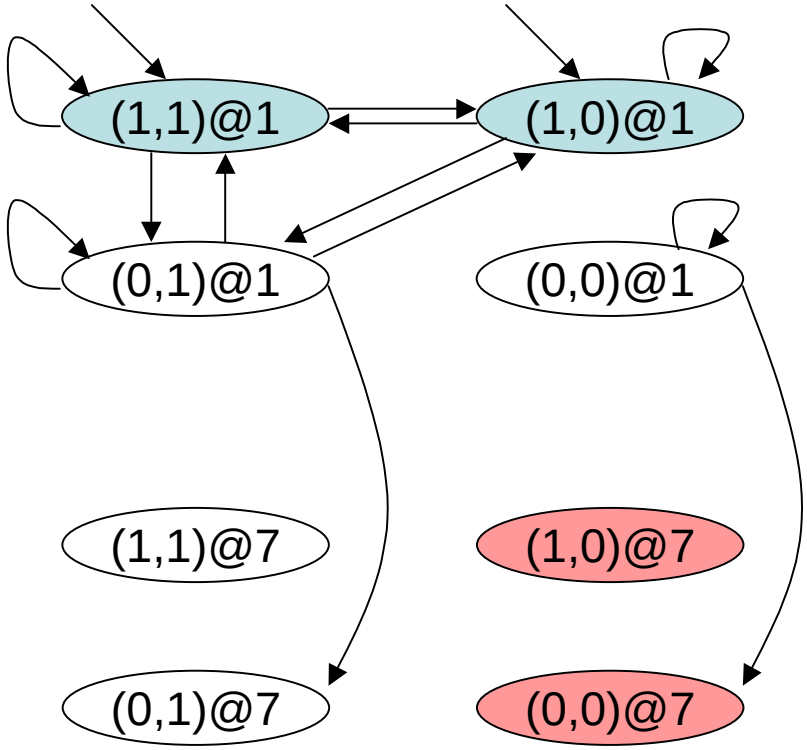
$$Q_1 = \forall(u \rightarrow \mathbf{EF}\ x),\ Q_2 = \forall(u \rightarrow \mathbf{EF}\ y)$$



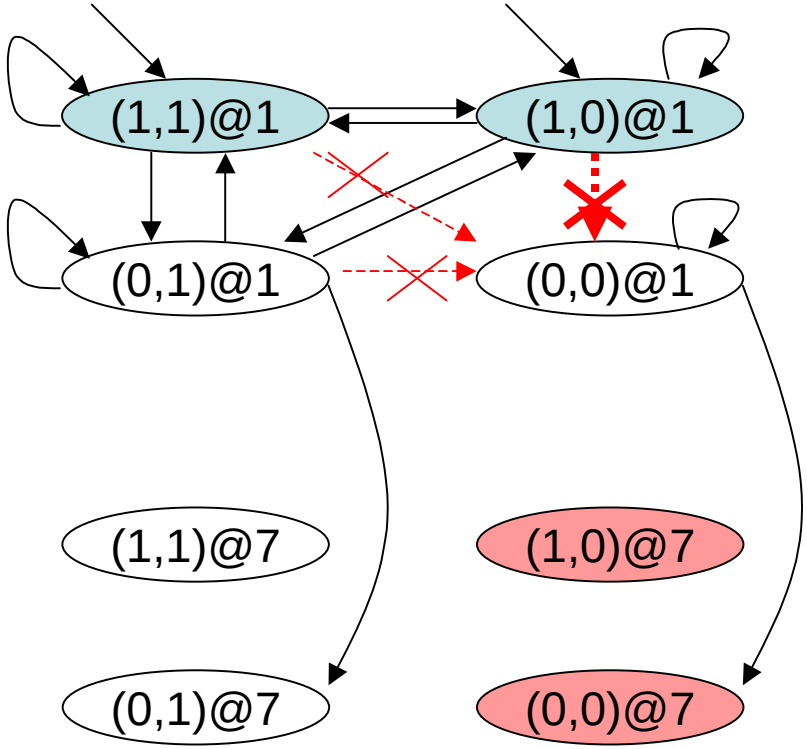


$$Q_1 = \forall(u \rightarrow \mathbf{EF}\ x),\ Q_2 = \forall(u \rightarrow \mathbf{EF}\ y)$$





$$Q_1 = \forall(u \rightarrow \mathbf{EF}\ x),\ Q_2 = \forall(u \rightarrow \mathbf{EF}\ y)$$



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

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

$$\mathsf{pre}(s, \neg Q_1) =$$
$$\forall(x \rightarrow \mathbf{EF}\ \mathbf{EX}x) \wedge \forall(u \rightarrow \neg(x \vee \mathbf{E}\ (\neg x\ \mathbf{U}\ (\neg x \wedge (\mathbf{EX}x \vee y)))))$$
$$\vee\ \ \forall(x \rightarrow \neg\mathbf{EF}\ \mathbf{EX}x) \wedge \forall(u \rightarrow \neg\mathbf{EF}\ \mathbf{EX}x)$$

$$\mathsf{pre}(s, \neg Q_2) = \forall(u \rightarrow \neg(x \vee \mathbf{E}\ (\neg x\ \mathbf{U}\ (\neg x \wedge y))))$$
$$(s = \texttt{"t:=y; y:=x; x:=x.next; y.next:=t"})$$

Summary and Future Work

In an on-going project we

- use CTL formulas to express properties on program heaps.
- develop an algorithm to calculate preconditions for each basic pointer-manipulating operation that are suitable for predicate abstraction.

Future work includes:

- To develop a decision procedure to judge whether a given predicate is satisfiable or not.
- To develop a method that analyzes a counterexample of the model checking on the abstract structure.
- To develop a method to verify liveness properties.

(the end of the slides)