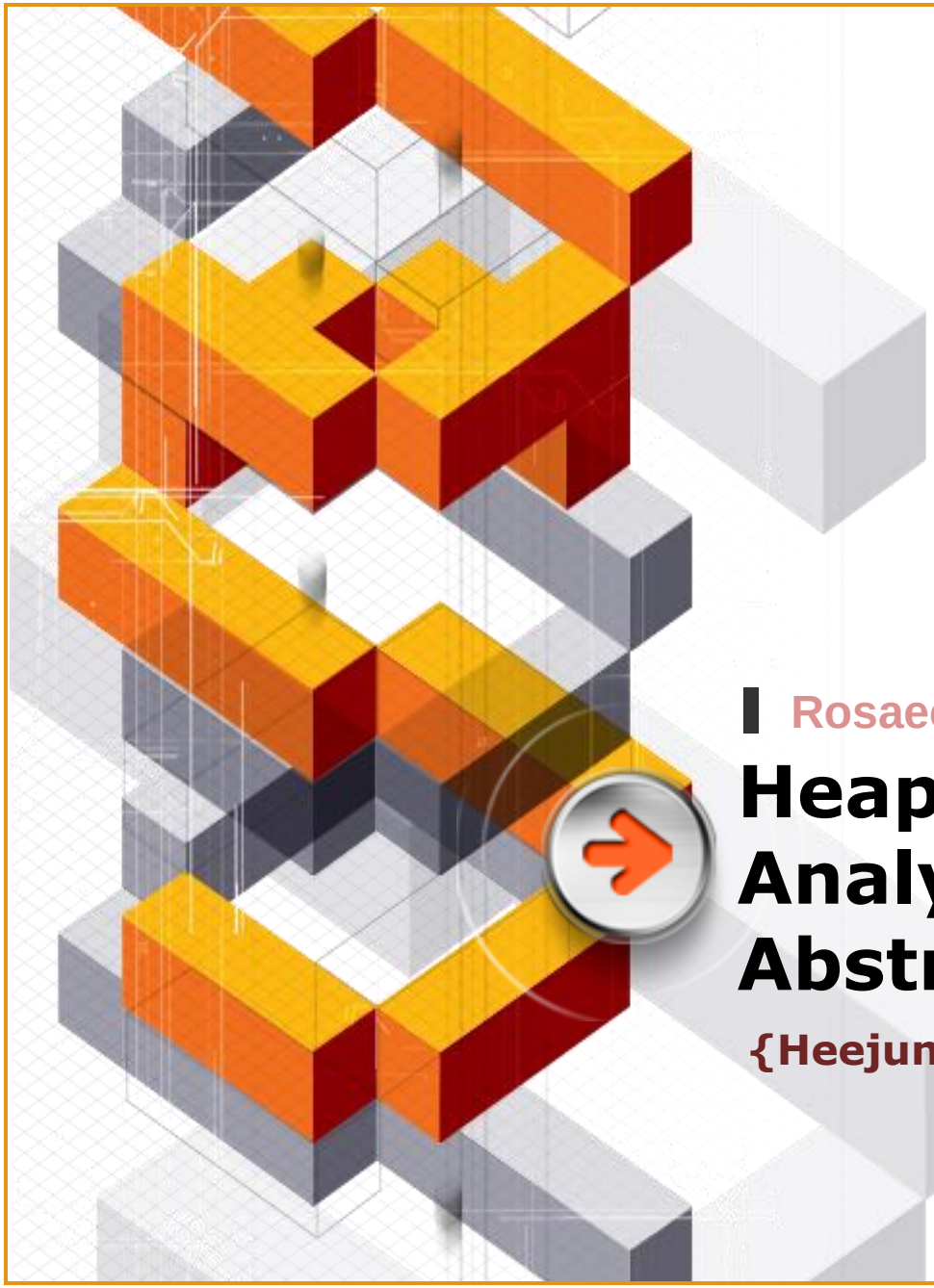


Rosaec 2nd workshop

Heap Data Structure Analysis using Abstract Parsing

{Heejung kim & Woosuk lee}
@ ropas.snu.ac.kr

Contents

Motivation

- To determine the correctness of heap data structures generated during the program execution.

Approach

- Transform the procedure of generating data structures into generating strings.
- Then, parsing those strings to determine correctness of data structures.
- To achieve our goal, we'll use 'Abstract Parsing' technique.

Reference

- Kyung-Goo Doh, Hyunha Kim and David Schmidt. *Abstract parsing: static analysis of dynamically generated string output using LR-parsing technology.* SAS 2009: The 16th International Static Analysis Symposium, LA, August 7-9, 2009. (to appear)

Background

- What is the 'Abstract Parsing?'

```
$s = "SELECT name WHERE class =".$class."
FROM students;";
$result = db_query($s);
```

To check syntactic correctness or harmlessness of $s

Background (cont.)

Abstract Parsing : Idea

- Instead of executing the program and parsing the result,

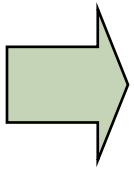
$$\mathcal{V}_0 e \Sigma = \{c_1, c_2, \ldots, c_n\} \qquad parse(c_i) = O/X$$

- Execute the program on abstract semantics using LR parsing technology.

$$\widehat{\mathcal{V}}_0 e \widehat{\Sigma} = \{O, X\}$$

Background (cont.)

```
x = 'a'
r = ']'
while …
    x = '[' . X . R
print x
```



X0 = a

R =]

X1 = X0 ∪ X2

X2 = [. X1 . R

X3 = X1

Each flow equations is interpreted as functions P → P

(input parse stack → output parse stack)

Grammar

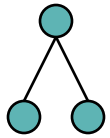
- Grammar of correct binary tree

S → n () () (*type S1*)

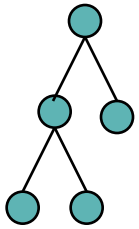
| n (S) (S) (*type S2*)

- Example

n (n () ()) (n () ())



n (n (n ()()) (n ()())) (n ()())



Grammar (cont.)

- Grammar of incorrect binary tree

I →	n (S) ()	(*type I1*)
	| n () (S)	(*type I2*)
	| n () (I)	(*type I3*)
	| n (I) ()	(*type I4*)
	| n (S) (I)	(*type I5*)
	| n (I) (S)	(*type I6*)
	| n (I) (I)	(*type I7*)

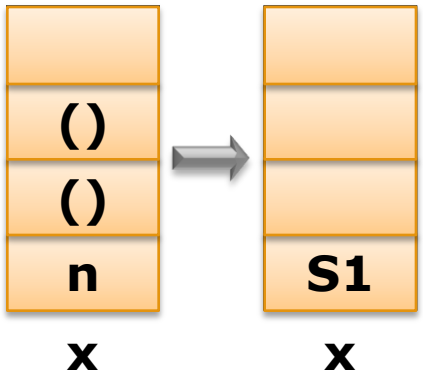
Language

c → x := e
| x.1 := e
| x.2 := e
| c1 ; c2
| if e c1 c2
| while e c

e → x
| malloc
| nil

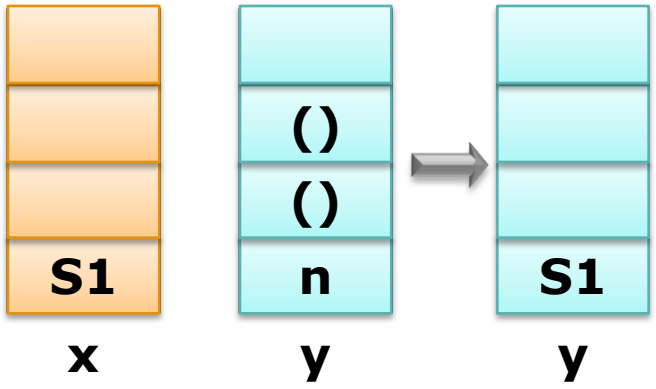
Example : Case 1

```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    y.2 := malloc;
    x := y
}
```



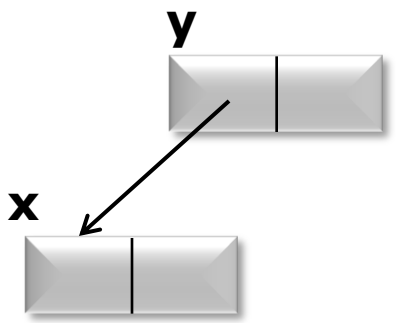
Case 1 (cont.)

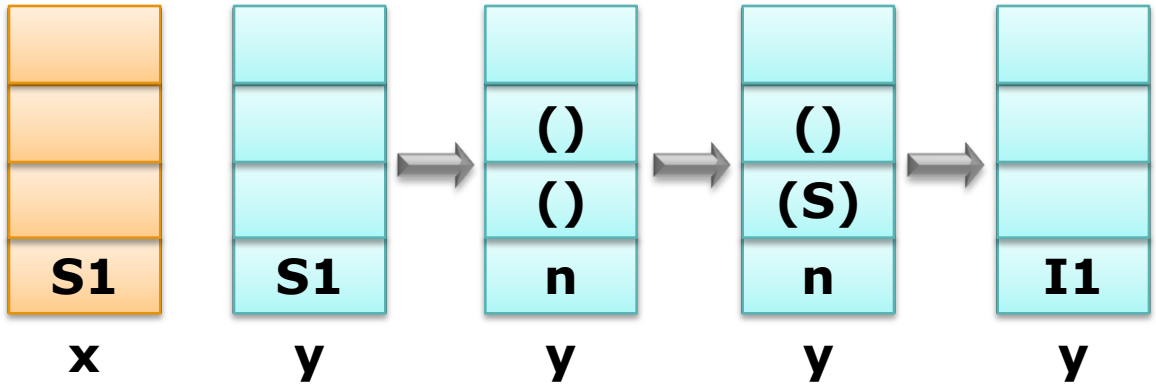
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    y.2 := malloc;
    x := y
}
```



Case 1 (cont.)

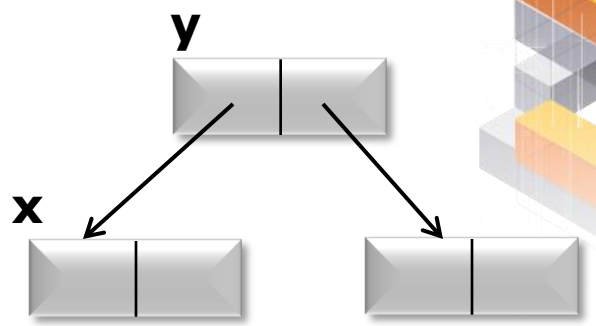
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    y.2 := malloc;
    x := y
}
```

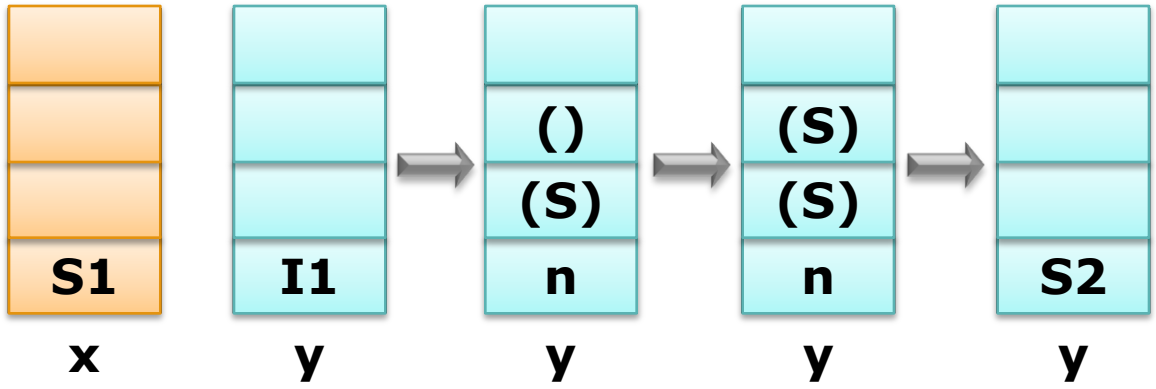




Case 1 (cont.)

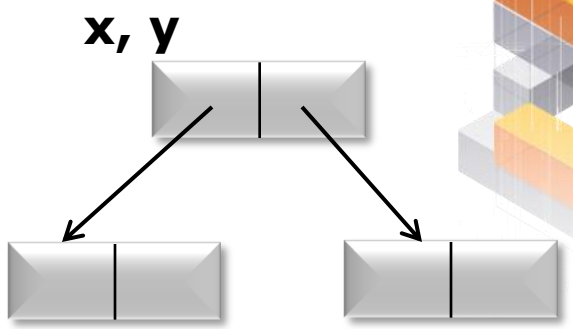
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    y.2 := malloc;
    x := y
}
```

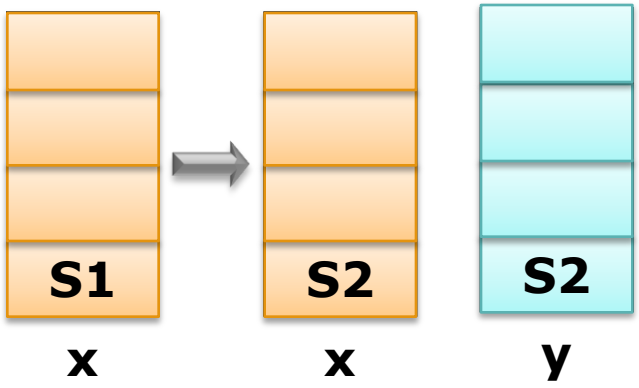




Case 1 (cont.)

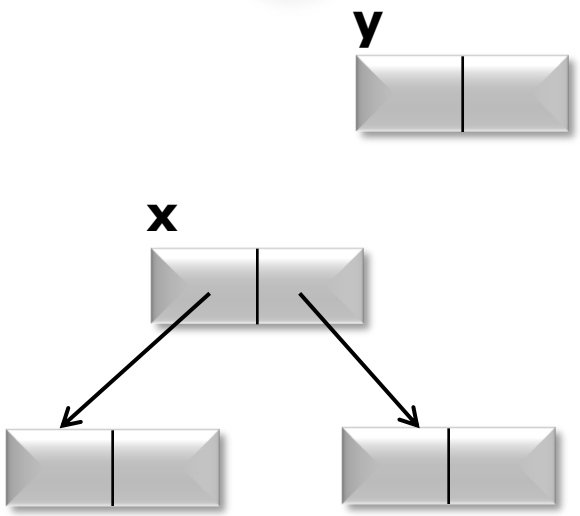
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    y.2 := malloc;
    x := y
}
```

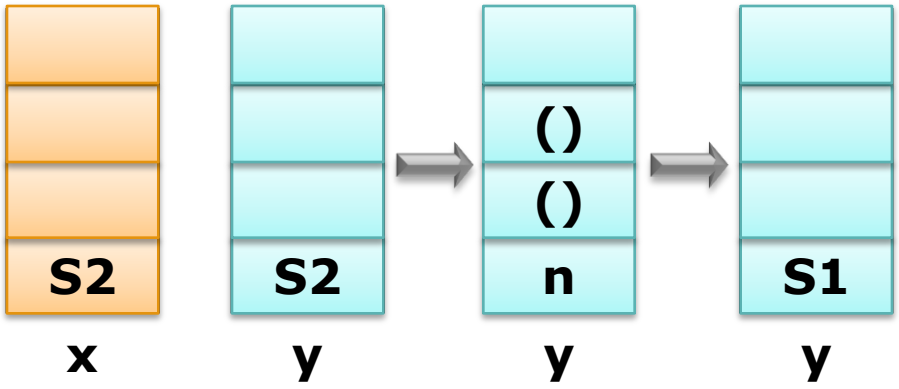




Case 1 (cont.)

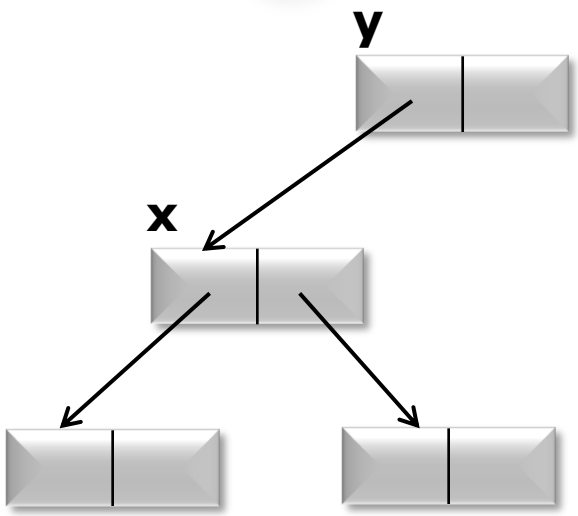
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    y.2 := malloc;
    x := y
}
```

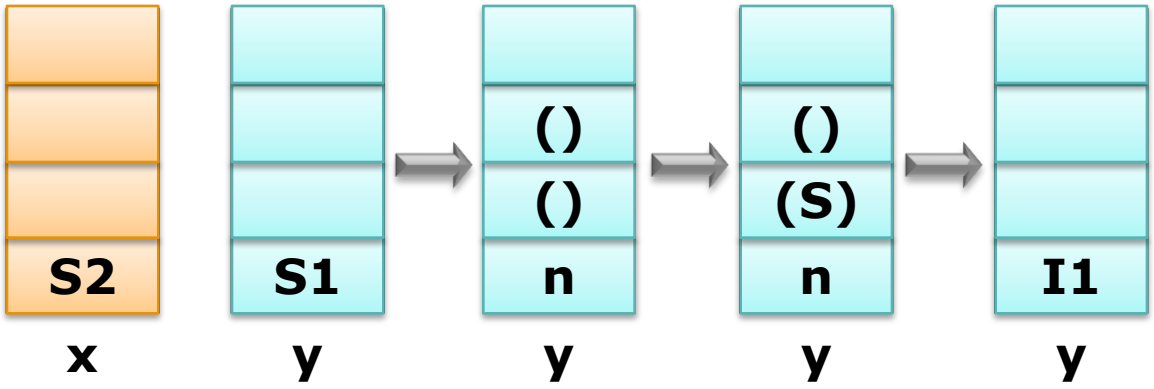




Case 1 (cont.)

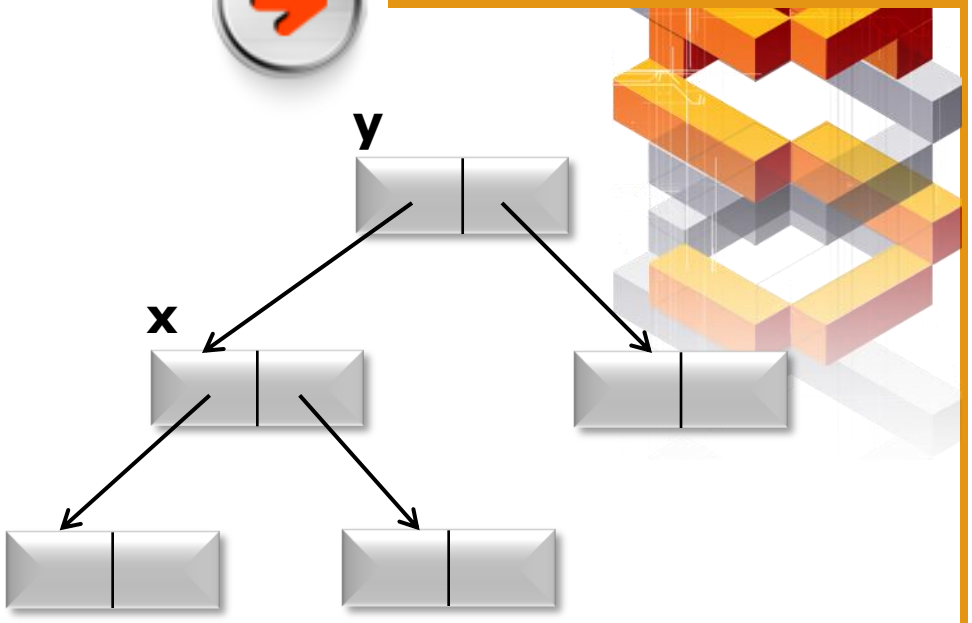
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    y.2 := malloc;
    x := y
}
```

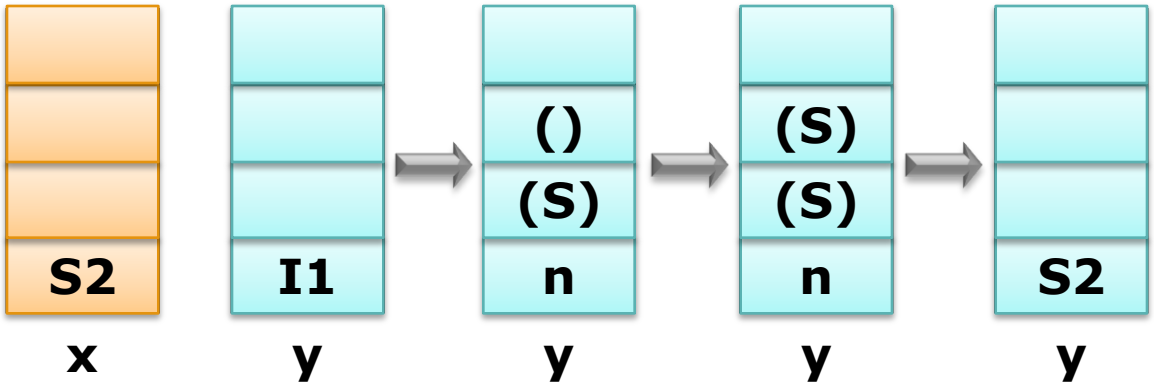




Case 1 (cont.)

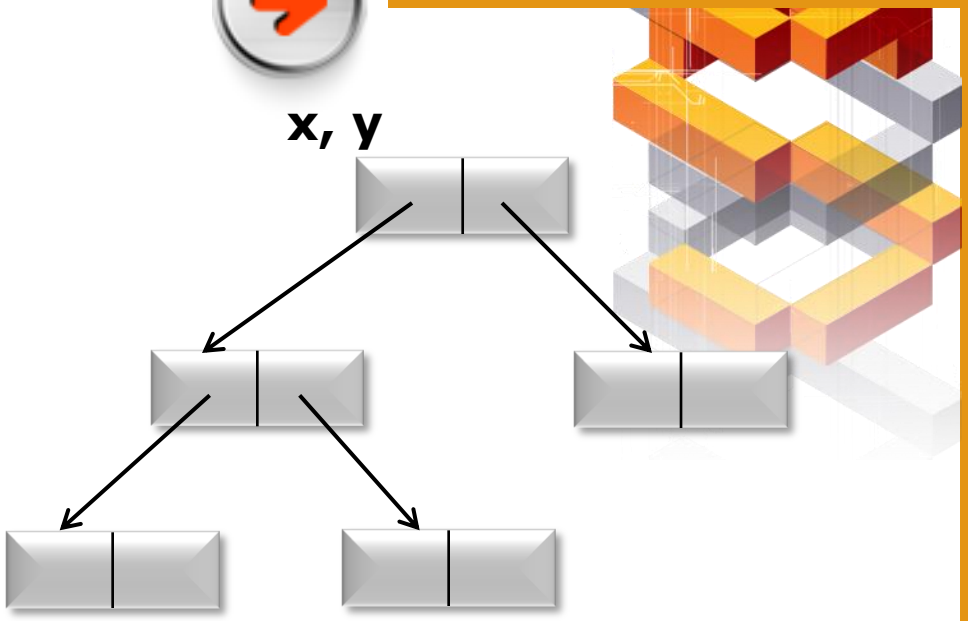
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    y.2 := malloc;
    x := y
}
```

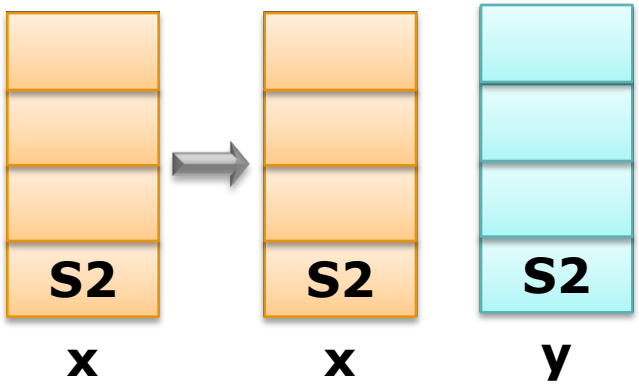




Case 1 (cont.)

```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    y.2 := malloc;
    x := y
}
```

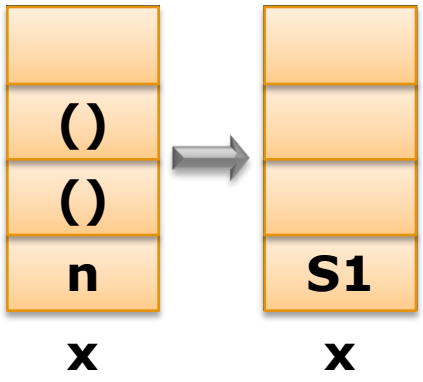




Reached fix point !!

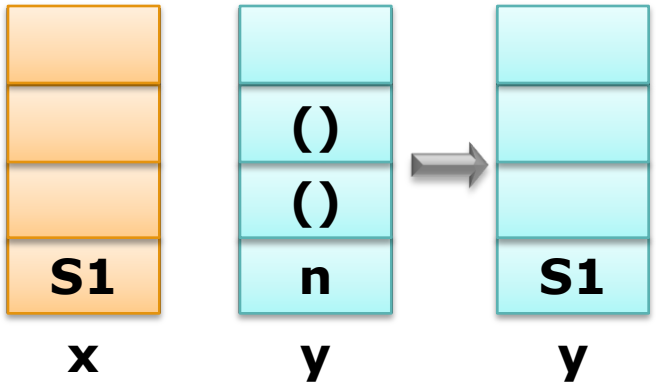
Case 2

```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    x = y
}
```



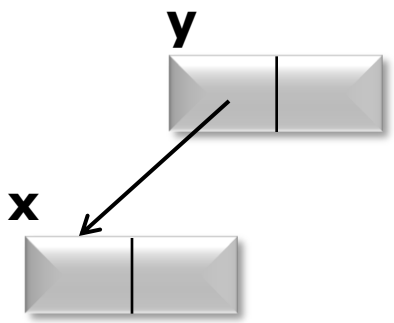
Case 2 (cont.)

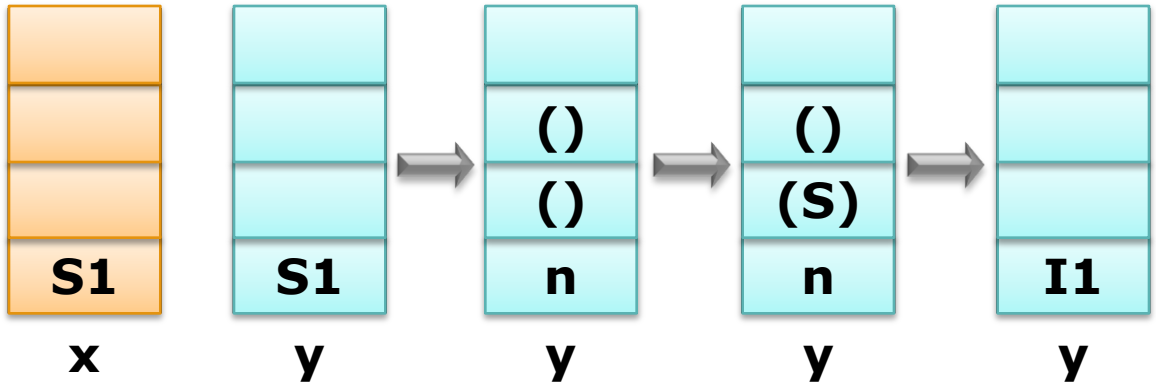
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    x = y
}
```



Case 2 (cont.)

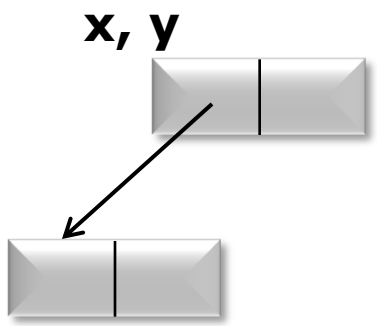
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    x = y
}
```

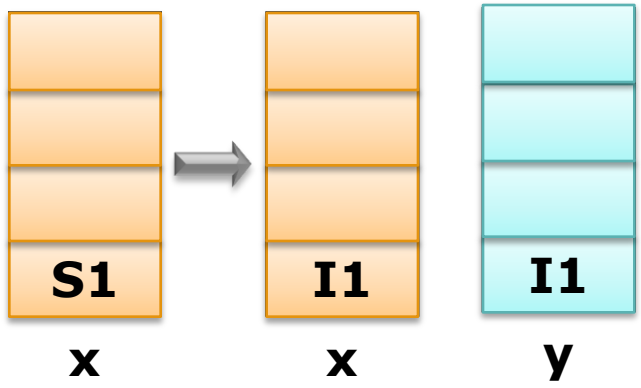




Case 2 (cont.)

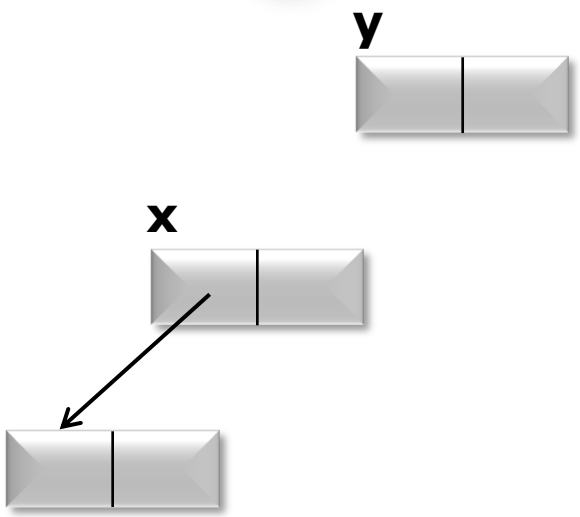
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    x = y
}
```

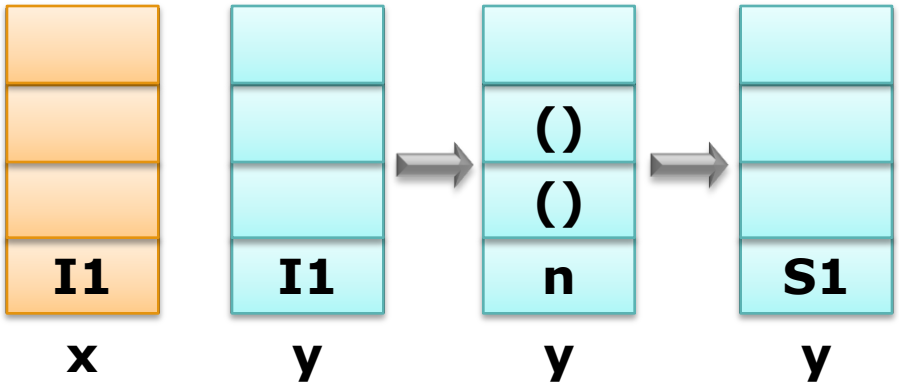




Case 2 (cont.)

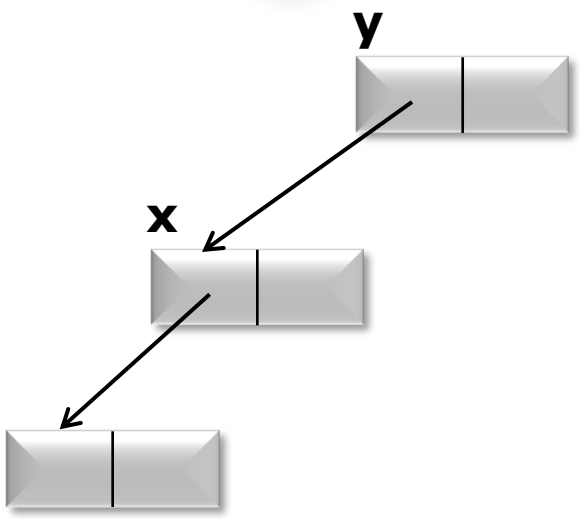
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    x = y
}
```

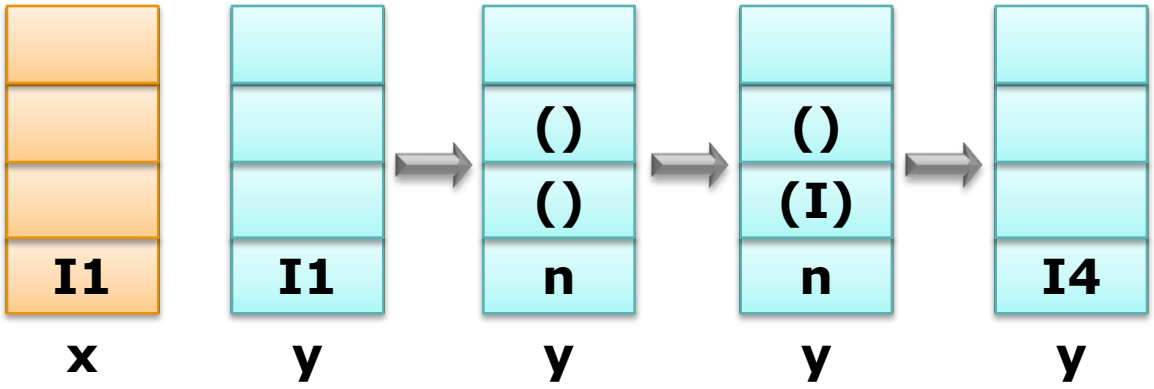




Case 2 (cont.)

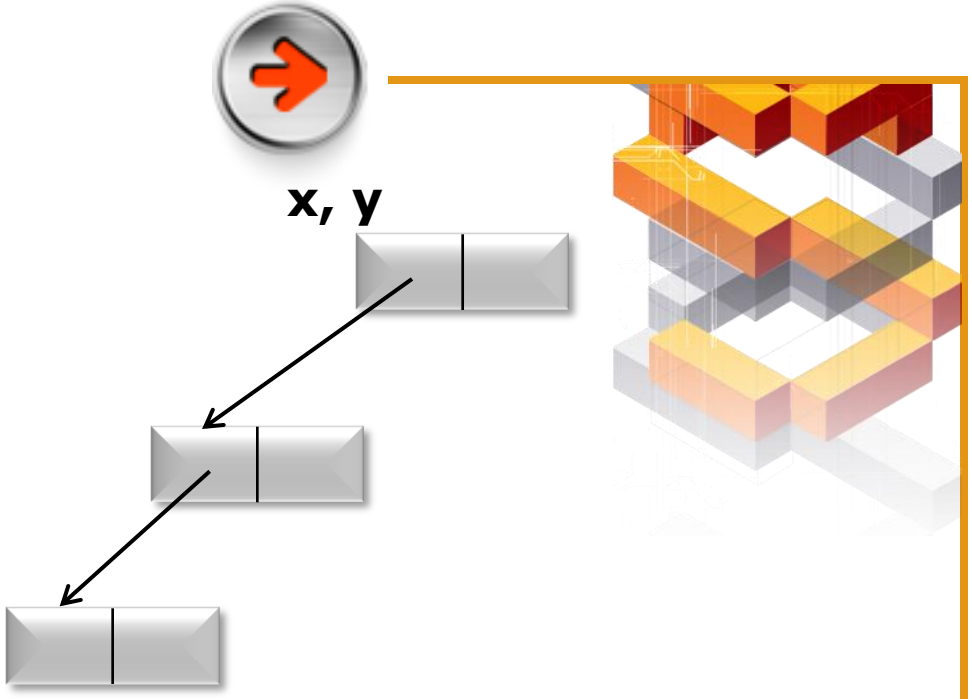
```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    x = y
}
```



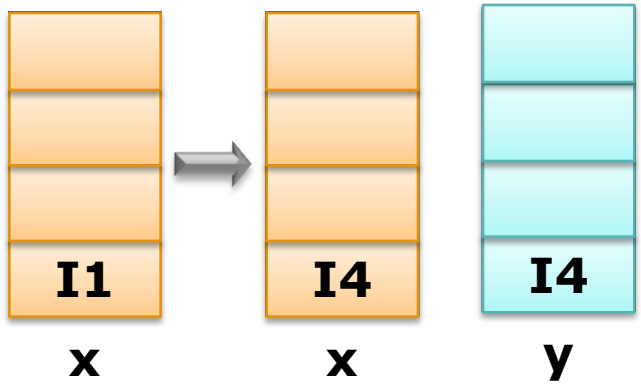


Case 2 (cont.)

```
x := malloc;
while (x)
{
    y := malloc;
    y.1 := x;
    x = y
}
```



Reached fix point !!



Future work

- **Language expansion (Make followings possible)**
  - Direct memory access, sub-tree dereferencing, cyclic update.
- **Grammar expansion**
  - Doubly linked list, linked list containing linked lists etc.


Thank You !