Tutorial-9

TA -Sinong

Quiz-8

1. What is the difference between a list and an array?

- Array is mutable

2. What do you think are the major advantages of the imperative programming style over the functional style?

- It’s easier to programmer.

3. Which rule is incorrect?

a. $$\frac{(M, e) \rightarrow (M', e')}{(M, \text{ref } e) \rightarrow (M', \text{ref } e')} \ (E{-}Ref)$$

b. $$\frac{l \notin dom(M)}{(M, \text{ref } v) \rightarrow ((M, l \mapsto v), l)} \ (E\text{-}RefV)$$

c. $$\frac{(M, e) \rightarrow (M', e')}{(M, !e) \rightarrow (M', !e')} \ (E{-}DeRef)$$

d. $$\frac{}{(M, !l) \rightarrow (M, l)} \ (E\text{-}DeRefLoc)$$

REFERENCES (OPERATIONAL SEMANTICS, CONT'D)

$$\frac{(M, e) \to (M', e')}{(M, \text{ref } e) \to (M', \text{ref } e')} \text{ (E - Ref)} \qquad \frac{l \notin \text{dom}(M)}{(M, \text{ref } v) \to ((M, l \mapsto v), l)} \text{ (E - RefV)}$$

$$\frac{(M, e) \to (M', e')}{(M, !e) \to (M', !e')} \text{ (E - DeRef)} \qquad \frac{}{(M, !l) \to (M, M(l))} \text{ (E - DeRefLoc)}$$

$$\frac{(M, e_1) \to (M', e_1')}{(M, e_1 := e_2) \to (M', e_1' := e_2)} \text{ (E - Assign1)} \qquad \frac{(M, e_2) \to (M', e_2')}{(M, v_1 := e_2) \to (M', v_1 := e_2')} \text{ (E - Assign2)}$$

$$\frac{}{(M, l := v) \to (M[l \mapsto v], ())} \text{ (E - Assign)}$$

3. Which rule is incorrect?

a. $$\frac{(M, e) \rightarrow (M', e')}{(M, \text{ref } e) \rightarrow (M', \text{ref } e')} \ (E{-}Ref)$$

b. $$\frac{l \notin \text{dom}(M)}{(M, \text{ref } v) \rightarrow ((M, l \mapsto v), l)} \ (E\text{-}RefV)$$

c. $$\frac{(M, e) \rightarrow (M', e')}{(M, !e) \rightarrow (M', !e')} \ (E{-}DeRef)$$

d. $$\frac{}{(M, !l) \rightarrow (M, l)} \ (E\text{-}DeRefLoc)$$

4.Which rule is incorrect?

a. $$\frac{(M, e_1) \rightarrow (M', e_1')}{(M, e_1 :\square\ e_2) \rightarrow (M', e_1' :\square\ e_2)}\ (E\text{-}Assign1)$$

b. $$\frac{(M, e_2) \rightarrow (M', e_2')}{(M, v_1 :\square\ e_2) \rightarrow (M', v_1 :\square\ e_2')}\ (E\text{-}Assign2)$$

c. $$\frac{}{(M, l := v) \rightarrow (M[l \mapsto v], v)}\ (E\text{-}Assign)$$

d. None of the above

REFERENCES (OPERATIONAL SEMANTICS, CONT'D)

$$\frac{(M, e) \to (M', e')}{(M, \text{ref } e) \to (M', \text{ref } e')} \text{ (E - Ref)} \qquad \frac{l \notin \text{dom}(M)}{(M, \text{ref } v) \to ((M, l \mapsto v), l)} \text{ (E - RefV)}$$

$$\frac{(M, e) \to (M', e')}{(M, !e) \to (M', !e')} \text{ (E - DeRef)} \qquad \frac{}{(M, !l) \to (M, M(l))} \text{ (E - DeRefLoc)}$$

$$\frac{(M, e_1) \to (M', e_1')}{(M, e_1 := e_2) \to (M', e_1' := e_2)} \text{ (E - Assign1)} \qquad \frac{(M, e_2) \to (M', e_2')}{(M, v_1 := e_2) \to (M', v_1 := e_2')} \text{ (E - Assign2)}$$

$$\frac{}{(M, l := v) \to (M[l \mapsto v], ())} \text{ (E - Assign)}$$

4.Which rule is incorrect?

a. $$\frac{(M, e_1) \rightarrow (M', e_1')}{(M, e_1 :\square\ e_2) \rightarrow (M', e_1' :\square\ e_2)}\ (E\text{-}Assign1)$$

b. $$\frac{(M, e_2) \rightarrow (M', e_2')}{(M, v_1 :\square\ e_2) \rightarrow (M', v_1 :\square\ e_2')}\ (E\text{-}Assign2)$$

c. $$\frac{}{(M, l := v) \rightarrow (M[l \mapsto v], v)}\ (E\text{-}Assign)$$

d. None of the above

5. Which one is incorrect?

a. $$\frac{\Sigma;\Gamma \vdash e_1 : t \quad \Sigma;\Gamma \vdash e_2 : t}{\Sigma;\Gamma \vdash e_1 := e_2 : \text{unit}} \ (\text{T-Assign})$$

b. $$\frac{\Sigma;\Gamma \mid\!- e : t\ \text{ref}}{\Sigma;\Gamma \mid\!- !e : t} \ (\text{T-Deref})$$

c. $$\frac{\Sigma;\Gamma \mid\!- e : t}{\Sigma;\Gamma \mid\!- \text{ref}\ e : t\ \text{ref}} \ (\text{T-Ref})$$

d. $$\frac{\Sigma(l) = t}{\Sigma;\Gamma \mid\!- l : t\ \text{ref}} \ (\text{T-Loc})$$

REFERENCES (TYPING)

- We define the typing relation for memory store as Σ (or Si):

  $\Sigma ::= . \mid \Sigma, l : t$ (t is the type of value stored at l)

- Our new typing judgment:

$$\Sigma; \Gamma \vdash e : t$$

- Types: t ::= .. | unit | t ref

$$\frac{}{\Sigma;\Gamma \mid\!- x : \Gamma(x)} \text{ (T - Var)} \qquad \frac{\Sigma;\Gamma, x : t_1 \mid\!- e : t_2}{\Sigma;\Gamma \mid\!- \lambda x : t_1 . e \ : \ t_1 \to t_2} \text{ (T - Abs)}$$

$$\frac{\Sigma;\Gamma \mid\!- e_1 : t_1 \to t_2 \quad \Sigma;\Gamma \mid\!- e_2 : t_1}{\Sigma;\Gamma \mid\!- e_1 \ e_2 : t_2} \text{ (T - App)} \qquad \frac{}{\Sigma;\Gamma \mid\!- () : \text{unit}} \text{ (T - Unit)}$$

$$\frac{\Sigma(l) = t}{\Sigma;\Gamma \mid\!- l : t \text{ ref}} \text{ (T - Loc)} \qquad \frac{\Sigma;\Gamma \mid\!- e : t}{\Sigma;\Gamma \mid\!- \text{ref } e : t \text{ ref}} \text{ (T - Ref)}$$

$$\frac{\Sigma;\Gamma \mid\!- e : t \text{ ref}}{\Sigma;\Gamma \mid\!- !e : t} \text{ (T - Deref)} \qquad \frac{\Sigma;\Gamma \mid\!- e_1 : t \text{ ref} \quad \Sigma;\Gamma \mid\!- e_2 : t}{\Sigma;\Gamma \mid\!- e_1 := e_2 : \text{unit}} \text{ (T - Assign)}$$

5. Which one is incorrect?

a. $$\frac{\Sigma;\Gamma \vdash e_1 : t \quad \Sigma;\Gamma \vdash e_2 : t}{\Sigma;\Gamma \vdash e_1 := e_2 : \text{unit}} \ (\text{T-Assign})$$

b. $$\frac{\Sigma;\Gamma \mid - e : t\ \text{ref}}{\Sigma;\Gamma \mid - !e : t} \ (\text{T-Deref})$$

c. $$\frac{\Sigma;\Gamma \mid - e : t}{\Sigma;\Gamma \mid - \text{ref}\ e : t\ \text{ref}} \ (\text{T-Ref})$$

d. $$\frac{\Sigma(l) \square t}{\Sigma;\Gamma \mid - l : t\ \text{ref}} \ (\text{T-Loc})$$

6. In an expression e1; e2, if e1 doesn't evaluate to (), what will happen?

- Raise error(get stuck)

7. Which one is incorrect?

a. $$\frac{(M, e_1) \rightarrow (M', e_1')}{(M, e_1; e_2) \rightarrow (M', e_1'; e_2)} \ (E\text{-}Seq1)$$

b. $$\frac{}{(M, (); e) \rightarrow (M, e)} \ (E\text{-}Seq2)$$

c. $$\frac{\Sigma; \Gamma \mid\text{-}\ e_1 : unit \quad \Sigma; \Gamma \mid\text{-}\ e_2 : t}{\Sigma; \Gamma \mid\text{-}\ e_1; e_2 : t} \ (T\text{-}Var)$$

d. All of them are correct

7. Which one is incorrect?

a. $$\frac{(M, e_1) \rightarrow (M', e_1')}{(M, e_1; e_2) \rightarrow (M', e_1'; e_2)} \ (E\text{-}Seq1)$$

b. $$\frac{}{(M, (); e) \rightarrow (M, e)} \ (E\text{-}Seq2)$$

c. $$\frac{\Sigma; \Gamma \mid\text{-}\ e_1 : unit \quad \Sigma; \Gamma \mid\text{-}\ e_2 : t}{\Sigma; \Gamma \mid\text{-}\ e_1; e_2 : t} \ (T\text{-}Var)$$

d. All of them are correct

8. Write down the evaluation steps of the following expression:

```
let y = ref 1 in
  let f = \x. x + !y in
    f (!y)
```

Example Evaluations

Program:

```
let x = ref 3 in
    let y = x in
        x := (!x) +1;
        !y
```

```
(., let x = ref 3 in
    let y = x in
    x:= (!x) + 1;
    y) →
(l 3, let x = l in
            let y = x in
            x := (!x) + 1;
            !y) →
(l 3, let y = l in
            l := (!l) + 1;
            !y) →
(l 3, l := (!l) + 1; !l) →
(l 3, l := 3 + 1; !l) →
(l 3, l := 4; !l) →
(l 4, (); !l) → (l 4, !l) → (l 4, 4)
```

8. Write down the evaluation steps of the following expression:

```
let y = ref 1 in
  let f = \x. x + !y in
    f (!y)
```

l 1, let y = l in
  let f = \x. x + !y in
    f (!y)

l 1, let f = \x. x + !l in
  f (!l)

l 1, \x.x+!l (!l)

l 1, \x.x+!l 1

l 1, 1+!l

l 1, 1+1

l 1, 2

Homework-8

Garbage-Collection

Problem-1

C has functions `malloc` and `free` that allow programmers to dynamically allocate and deallocate heap space. Research these two functions and compare their similarities and differences with the function `new` and `delete`.

Feature	new/delete	malloc/free
Memory allocated from	'Free Store'	'Heap'
Returns	Fully typed pointer	void*
On failure	Throws(never returns NULL)	Returns NULL
Required size	Calculated by compiler	Must be specified in bytes
Handling arrays	Has an explicit version	Requires manual calculations
Reallocating	Not handled intuitively	Simple (no copy constructor)
Call of reverse	Implementation defined	No
Low memory cases	Can add a new memory allocator	Not handled by user code
Overridable	Yes	No
Use of (con-)/destructor	Yes	No

Reference:
https://stackoverflow.com/questions/240212/what-is-the-difference-between-new-delete-and-malloc-free

Problem-2

Can you think of a better Mark-n-Sweep algorithm that reduces the amount of waiting time when garbage collecting?

1. Tri-color Marking Algorithm[1]

In tri-color makring, there are three sets: white, gray and black:

- White: This object is not marked.
- Gray: The object is marked but its members are not marked.
- Black: The object and its members are all marked.

There are four steps for tri-color marking.

- **Initial Marking (STW)**: When allocate a new object, we put it in white set.
- **Parallel Marking (Parallel)**: 1. Move all objects in roots from white set to gray set. 2. Then move objects from gray set to black set and move each white object it references to the gray set. 3. Repeat 2 until gray set is empty. During this step, if the main program allocates new object or change the references, these will be recorded and passed to the next step.
- **Remarking (STW)**: Do remarking depends on the records.
- **Sweep (Parallel)**: Sweep all the objects in white set.
- **Reset (Parallel)**: Reset all objects into white set.

1:https://en.wikipedia.org/wiki/Tracing garbage collection#Tri-color marking

Problem-2

How to solve two problems:

- **allocate new object**: If the program allocates new object during parallel marking or parallel sweep, it will be move to white set and be swept. To solve this problem we can change the mark strategy. When doing parallel marking, if the program allocate a new object, it will be moved to gray set.
- **references change**: Suppose A references B. During parallel marking, when A is in gray, B is in white, C is in Black, the main program let C references to B and delete the reference between A and B. In this case B will not be moved to gray so it will be swept. To solve this problem we can record all the reference change. One solution is during remarking, we move A back to gray since it has a new reference.

Compare to the original mark-n-sweep algorithm, tri-color algorithm can reduce the waiting time because some of it's steps are parallel.

Problem-2

2. Lazy sweep
During sweep, we can only sweep until there is enough space in the freelist to satisfy the allocation and the sweep process can be done parallelly.

3. Generational Collection algorithm
Divide the heap into 3 spaces: Yong, Old and Full according to the survival age.

We use 3 individual garbage collection algorithms for there 3 spaces.

In practical, most garbage collection happens in Yong so that in the most time we only doing garbage collection over Yong instead of the whole heap, which reduce the waiting time.

4. Other Solutions
.....

□

Problem-3

- Show the configuration of memory cells under reference counting.(After the final step)
- Suppose we use mark-and-sweep GC on the same program, and the maximum heap size is 20 memory cells (one cell for one object). When GC action happens, and after GC, what is the configuration of the memory?

```
Object[] p = new Object[2];
Object[] q = new Object[2];
Object[] s = new Object[2];
Object[] t = new Object[2];
p[0] = q;
p[1] = s;
q[0] = p;
q[1] = t;
s[0] = t;
s[1] = null;
t[0] = s;
t[1] = null;
for (int i = 0; i < 3; i++) {
s = new Object[2];
t = new Object[2];
s[0] = t;
s[1] = p;
t[0] = s;
t[1] = q;
p[1] = null;
q[1] = null;
p = s;
q = t;
}
```

Problem-3

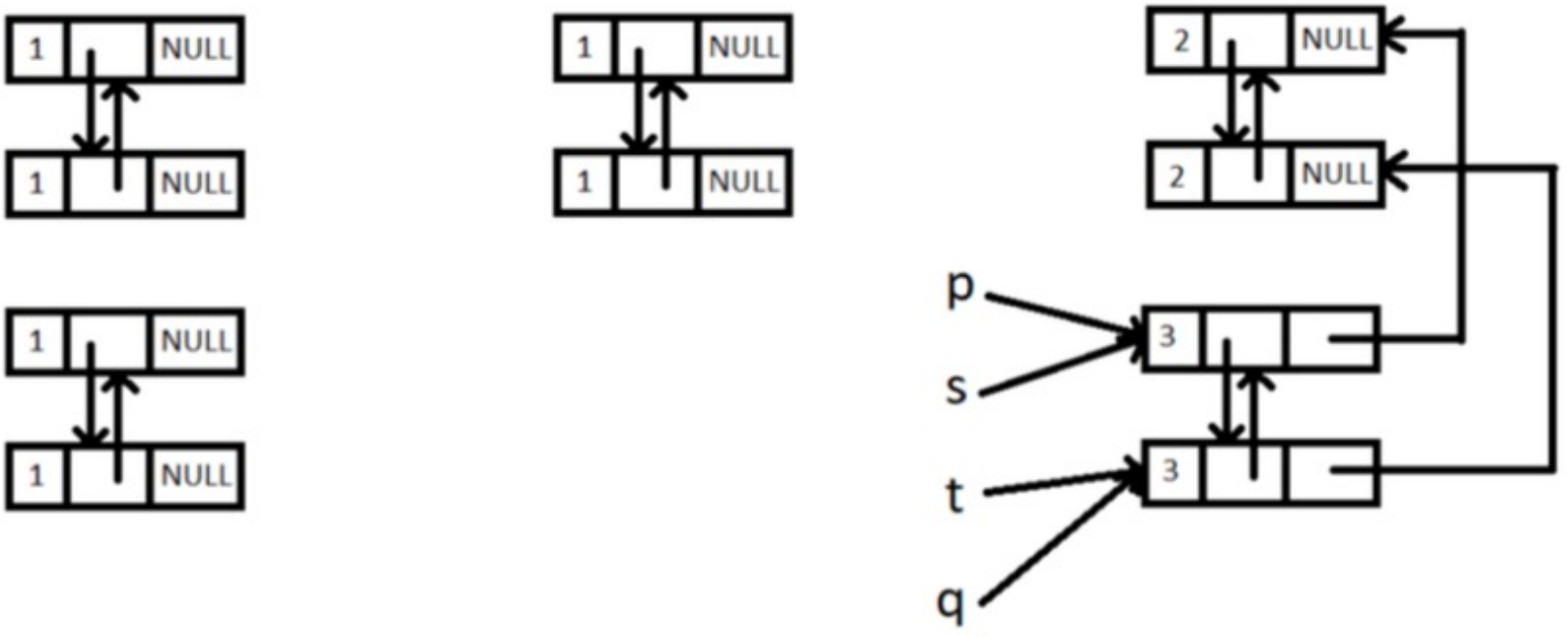

Figure 1: a

Problem-3

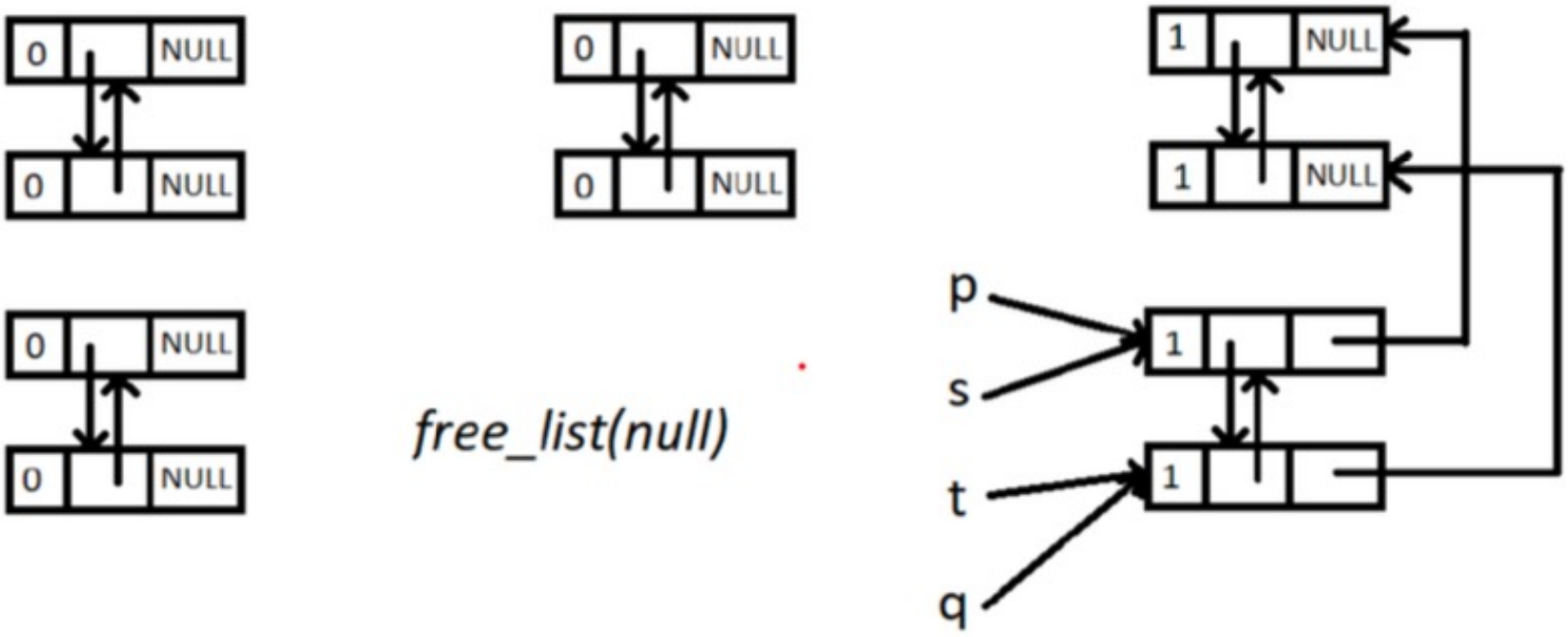

Figure 2: b_begin

Problem-3

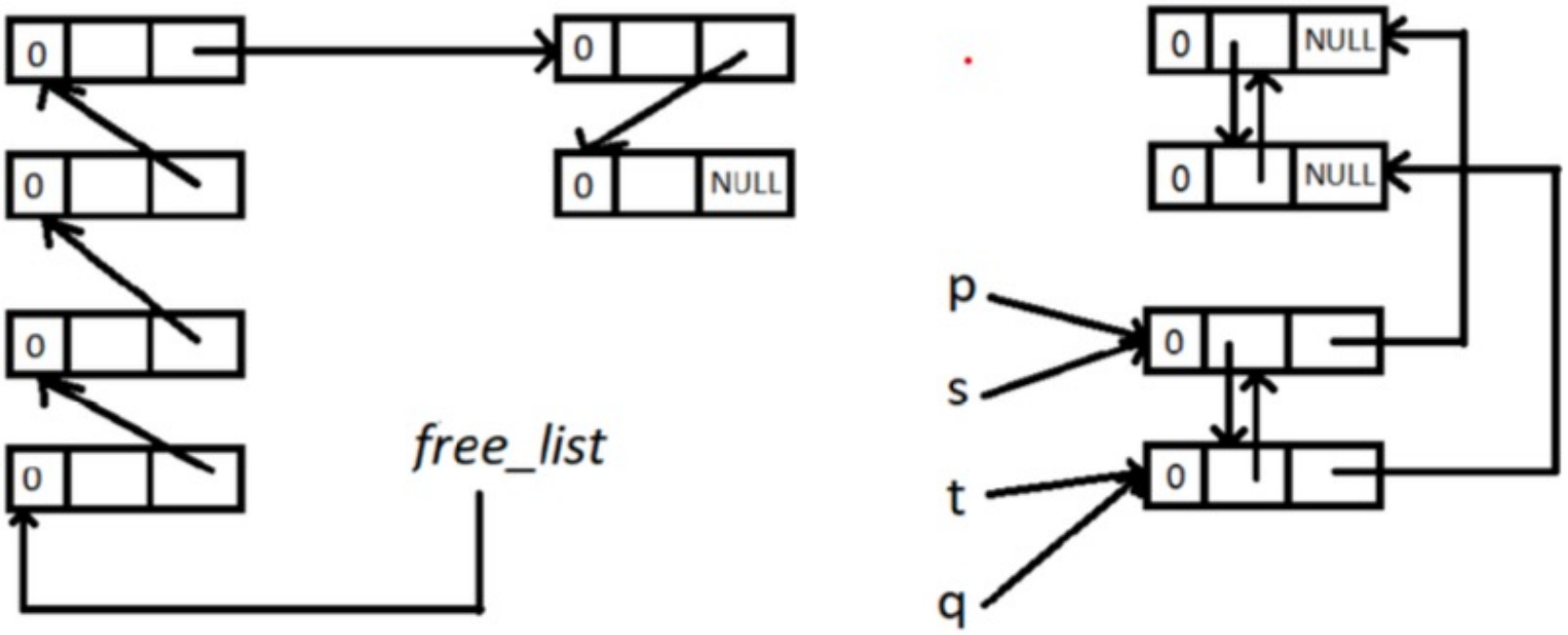

Figure 3: b_after