

# Exam Questions

---

TEACHING ASSISTANT: DAVID TRABISH

# Question

The program contains a string constant.

compiler construction

compiling

linking

runtime

# Question

The program contains a string constant.

compiler construction

compiling

linking

runtime

# Question

The function **printf** from the standard library is not found.

compiler construction

compiling

linking

runtime

# Question

The function **printf** from the standard library is not found.

compiler construction

compiling

linking

runtime

# Question

Every time the command  $x = y + 42$  is executed, the value of  $y$  is 0.

compiler construction

compiling

linking

runtime

# Question

Every time the command  $x = y + 42$  is executed, the value of  $y$  is 0.

compiler construction

compiling

linking

runtime

# Question

All the variable names start with x.

compiler construction

compiling

linking

runtime



# Question

All the variable names start with x.

compiler construction

compiling

linking

runtime

# Question

The program contains a nested loop.

compiler construction

compiling

linking

runtime

# Question

The program contains a nested loop.

compiler construction

compiling

linking

runtime

# Question

The program contains a string constant with at least 200 characters.

compiler construction

compiling

linking

runtime

# Question

The program contains a string constant with at least 200 characters.

compiler construction

compiling

linking

runtime

# Question

The size of the stack exceeds 400 bytes.

compiler construction

compiling

linking

runtime

# Question

The size of the stack exceeds 400 bytes.

compiler construction

compiling

linking

runtime

# Question

The code contains only invocations of methods which located at offset 4 in the virtual table.

compiler construction

compiling

linking

runtime



# Question

The code contains only invocations of methods which located at offset 4 in the virtual table.

compiler construction

compiling

linking

runtime

# Question

There is an addition operation which is never executed in any run.

compiler construction

compiling

linking

runtime

# Question

There is an addition operation which is never executed in any run.

compiler construction

compiling

linking

runtime

# LR(0) Item

An LR(0) item with the dot at the end is called **reduce** item:

- $N \rightarrow \alpha\beta.$

Otherwise, it's a **shift** item:

- $N \rightarrow .\alpha\beta$
- $N \rightarrow \alpha.\beta$

# LR(0) Item Closure Set

The LR(0) closure set of an LR(0) item  $i$  is a set  $S$  such that:

- $i \in S$
- If  $A \rightarrow \alpha.N\beta \in S$  then for each rule  $N \rightarrow \gamma$ :
  - $N \rightarrow.\gamma \in S$

# SLR(1)

- Same push-down automaton as in LR(0)
- But reduce items has a look-ahead set
  - $A \rightarrow \alpha.t_1, t_2, \dots$
  - where  $Follow(A) = \{t_1, t_2, \dots\}$

# LR(1)

Maintain items with **more precise** look-ahead sets

An **LR(1) item** is of the form:

- $N \rightarrow \alpha.\beta \{\sigma\}$
- where  $\sigma = t_1, t_2, \dots$  (terminals)

# LR(1) Item Closure Set

The **LR(1) closure set** of an LR(1) item  $i$  is a set  $S$  such that:

- $i \in S$
- If  $A \rightarrow \alpha.N\beta \{ \sigma \} \in S$  then for each rule  $N \rightarrow \gamma$ :
  - $N \rightarrow .\gamma \{ \tau \} \in S$ , where  $\tau = First(\beta, \{ \sigma \})$

Definition for  $First(\beta, \{ \sigma \})$ :

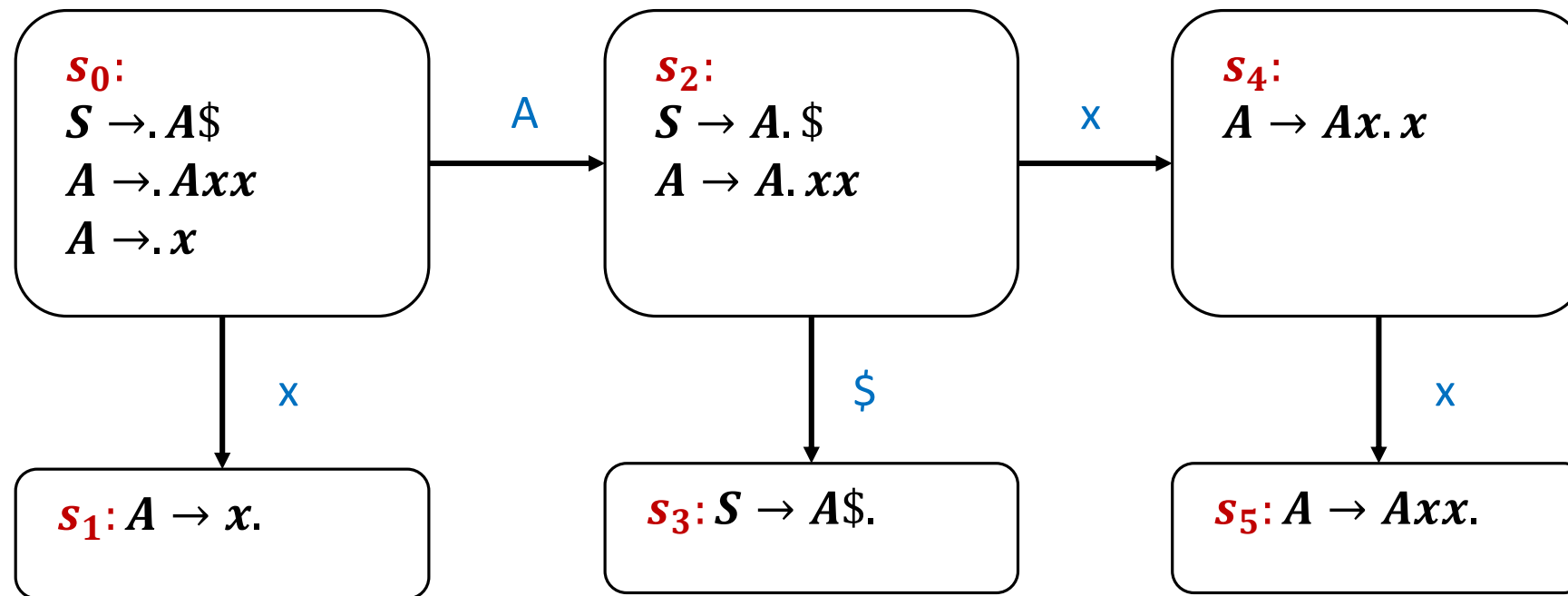
- If  $\beta$  is not nullable:
  - $First(\beta)$
- Otherwise:
  - $(First(\beta) \cup \{ \sigma \}) \setminus \{ \epsilon \}$



# Question

Is the following CFG LR(0) / SLR(1) / LR(1)?

- $S \rightarrow A\$$
- $A \rightarrow Axx$
- $A \rightarrow x$

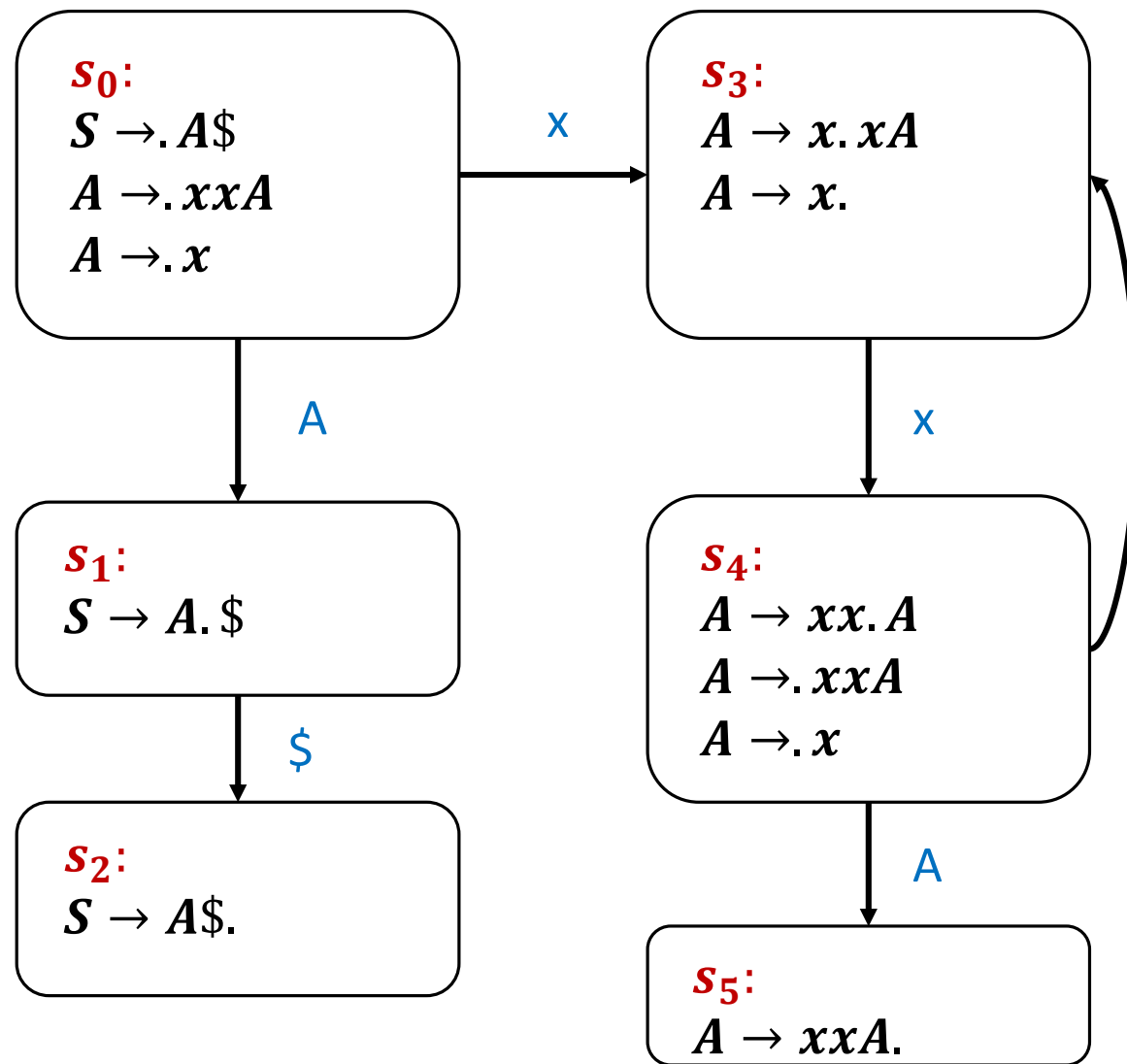


$S \rightarrow A \$$   
 $A \rightarrow A x x$   
 $A \rightarrow x$

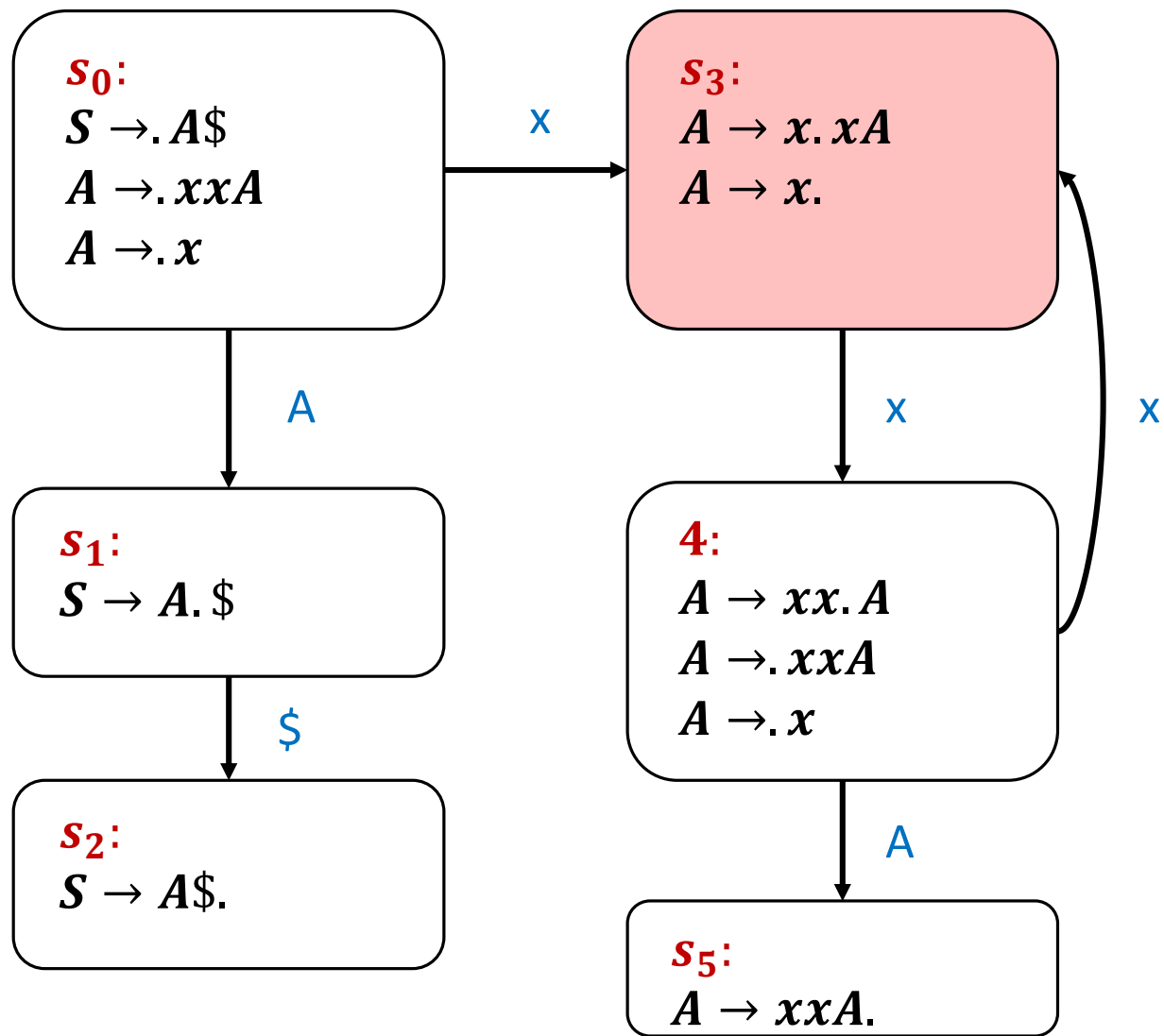
# Question

Is the following CFG LR(0) / SLR(1) / LR(1)?

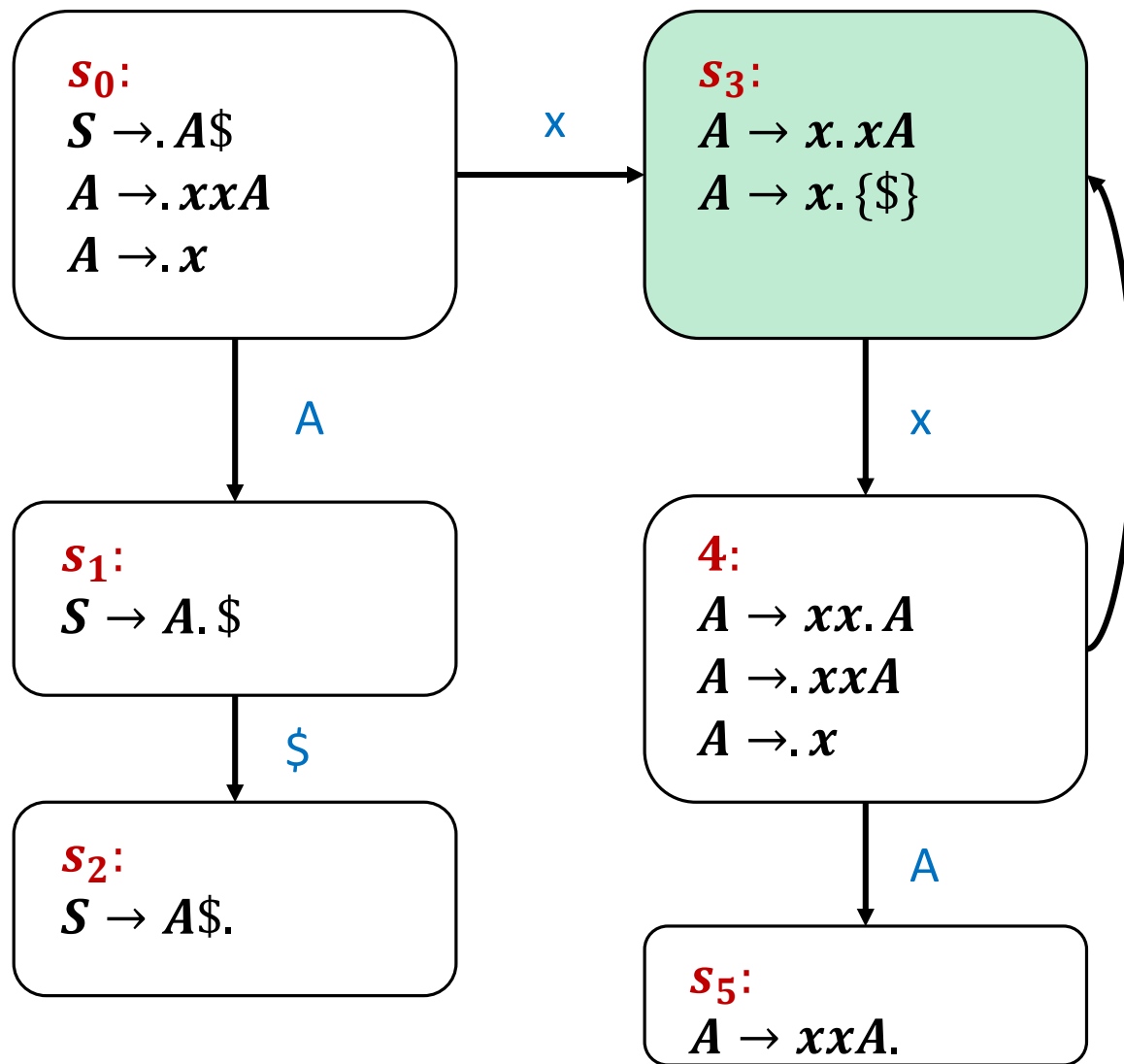
- $S \rightarrow A\$$
- $A \rightarrow xxA$
- $A \rightarrow x$



$S \rightarrow A \$$   
 $A \rightarrow x x A$   
 $A \rightarrow x$



$S \rightarrow A \$$   
 $A \rightarrow x x A$   
 $A \rightarrow x$



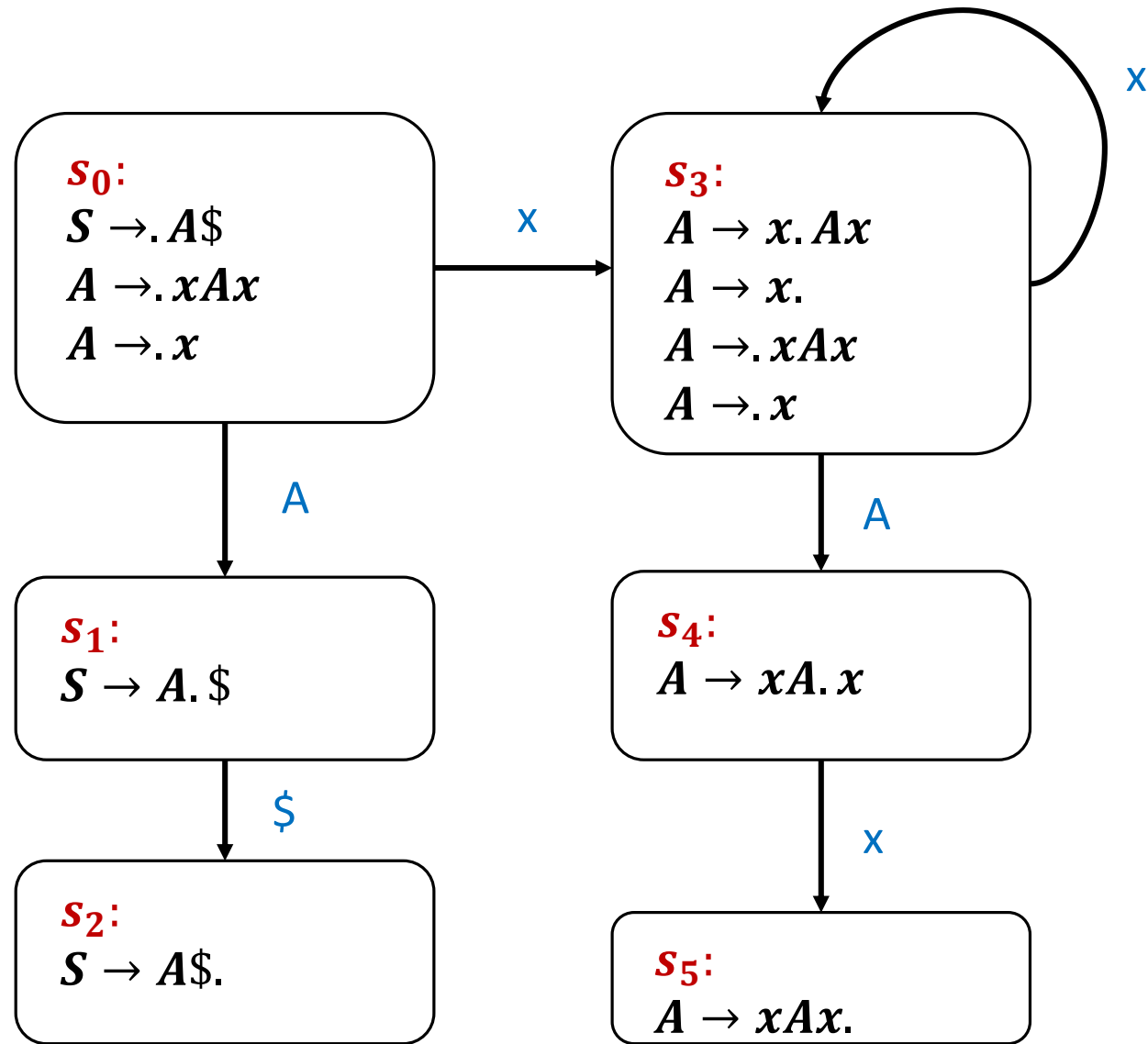
$Follow(A) = \{ \$ \}$

$S \rightarrow A\$$   
 $A \rightarrow xxA$   
 $A \rightarrow x$

# Question

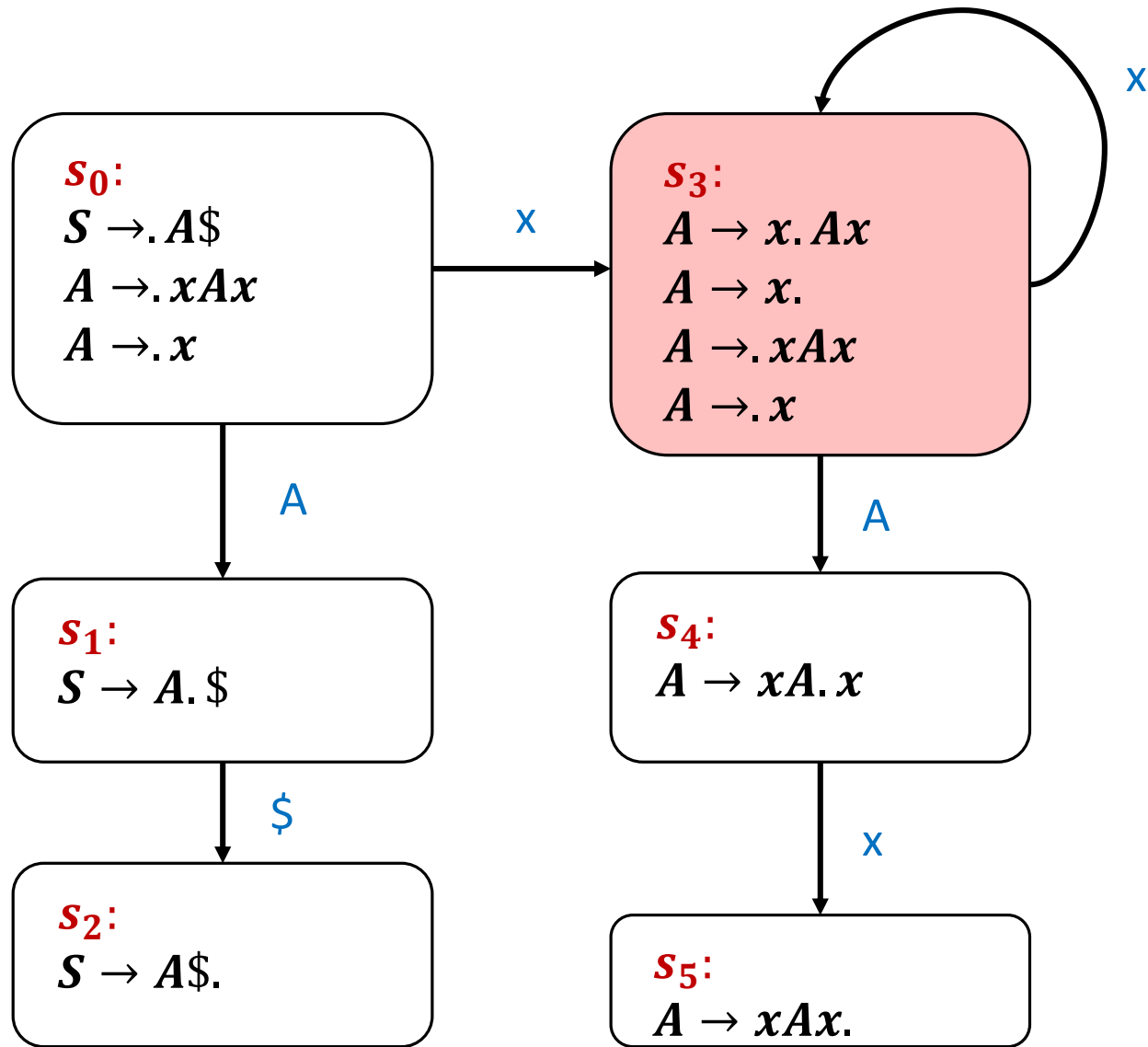
Is the following CFG LR(0) / SLR(1) / LR(1)?

- $S \rightarrow A\$$
- $A \rightarrow xAx$
- $A \rightarrow x$

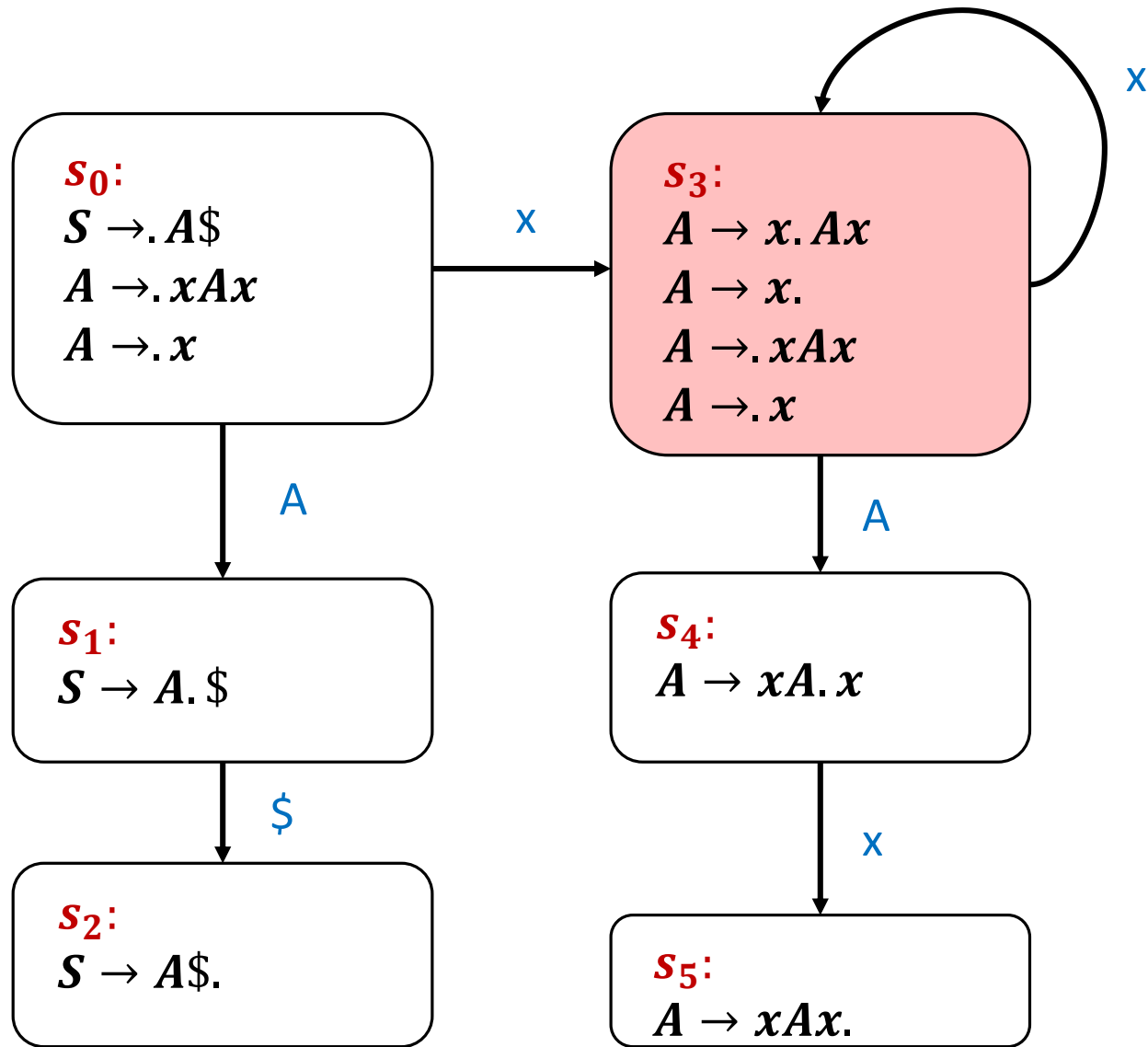


$S \rightarrow A \$$   
 $A \rightarrow x A x$   
 $A \rightarrow x$



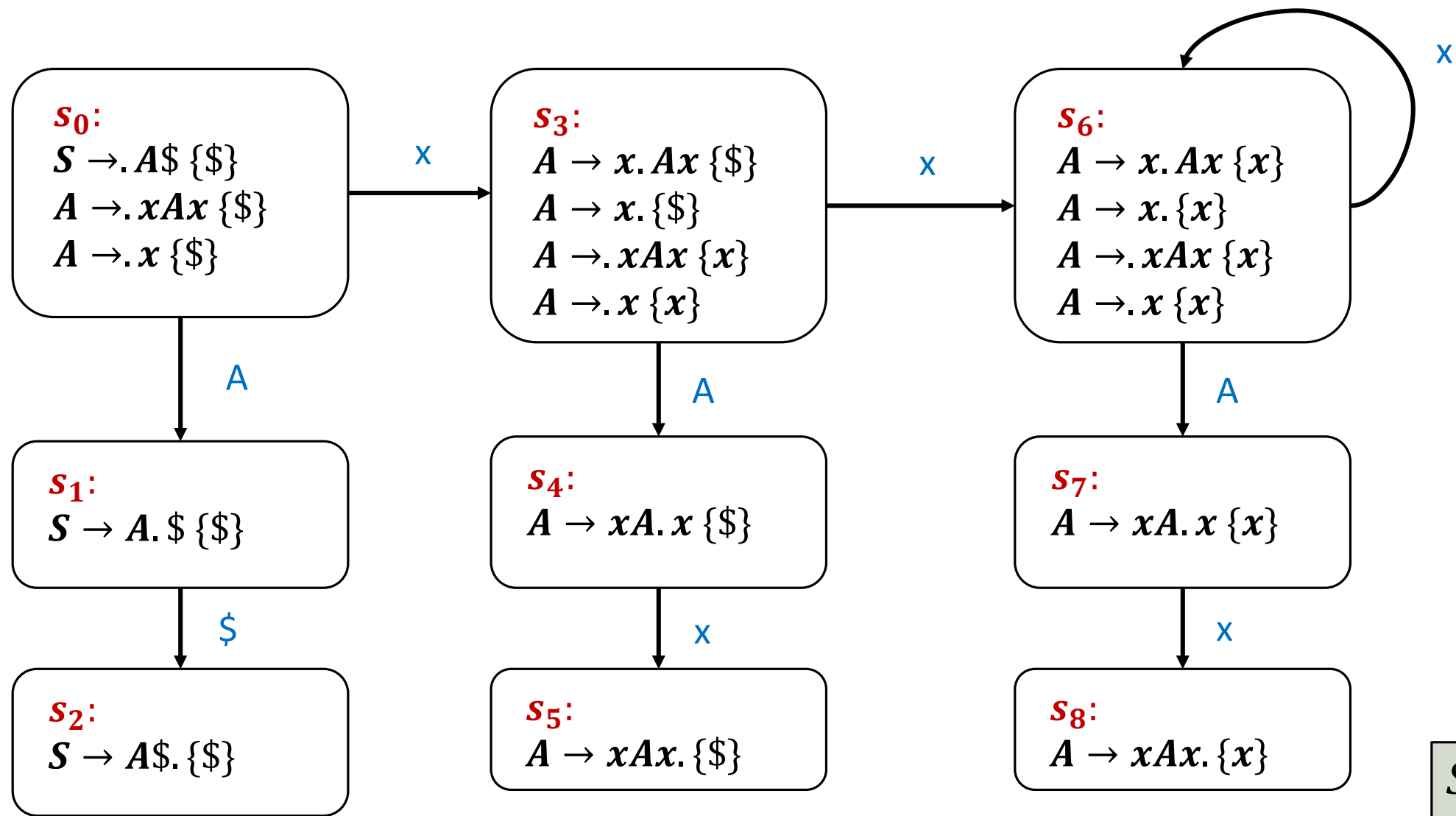


$S \rightarrow A \$$   
 $A \rightarrow x A x$   
 $A \rightarrow x$

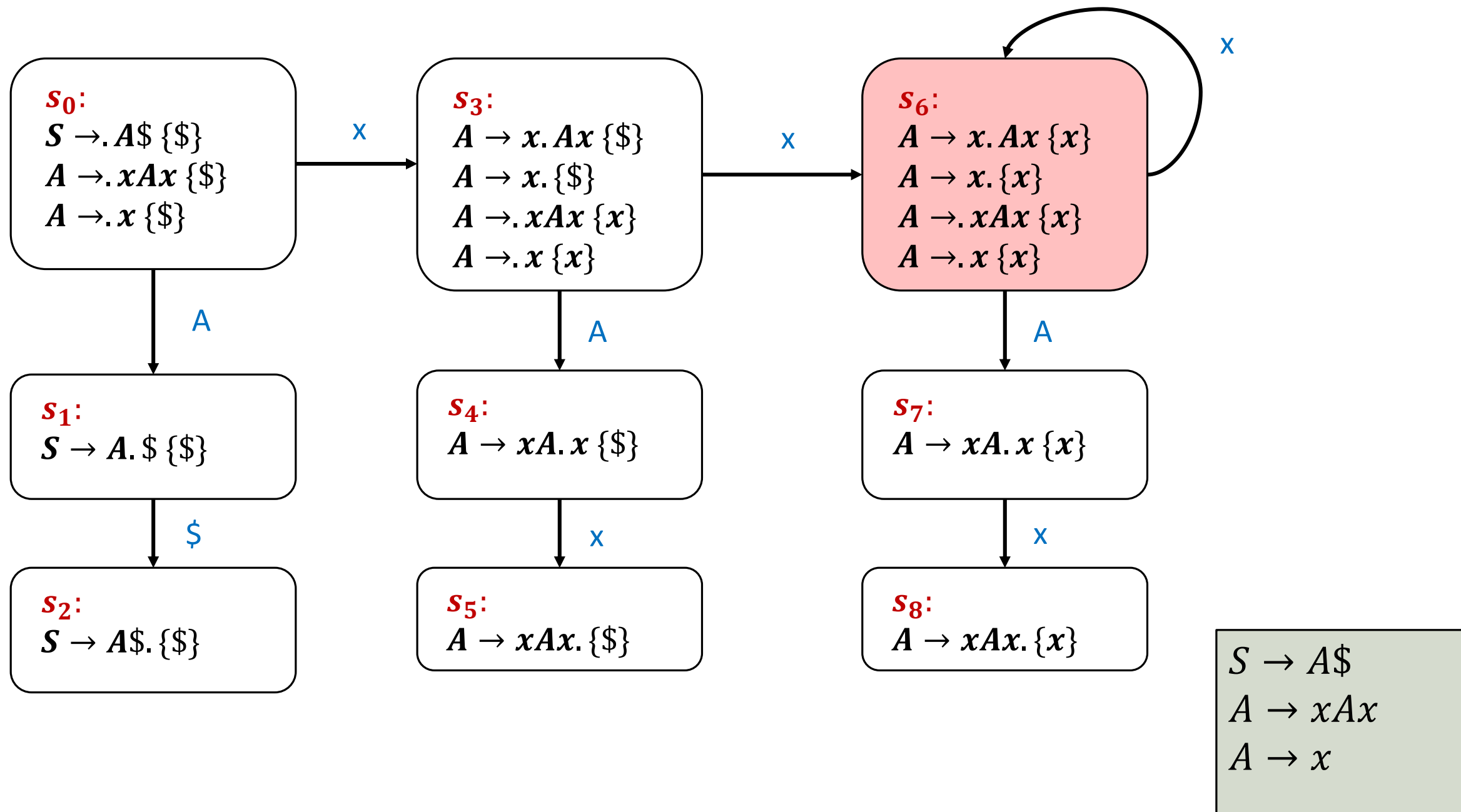


$$\text{Follow}(A) = \{\$, x\}$$

$S \rightarrow A\$$   
 $A \rightarrow xAx$   
 $A \rightarrow x$



$S \rightarrow A\$$   
 $A \rightarrow xAx$   
 $A \rightarrow x$



# Question

We extend the language with automatic type inference:

- Can use **auto** when the declaration has an initial value

Describe the changes required in:

- Lexical analysis
- Syntactic analysis
- Semantic analysis

```
auto i := 8 + 100;  
auto s := "1234";  
class A {}  
A a := new A;  
auto b := a;
```

# Question

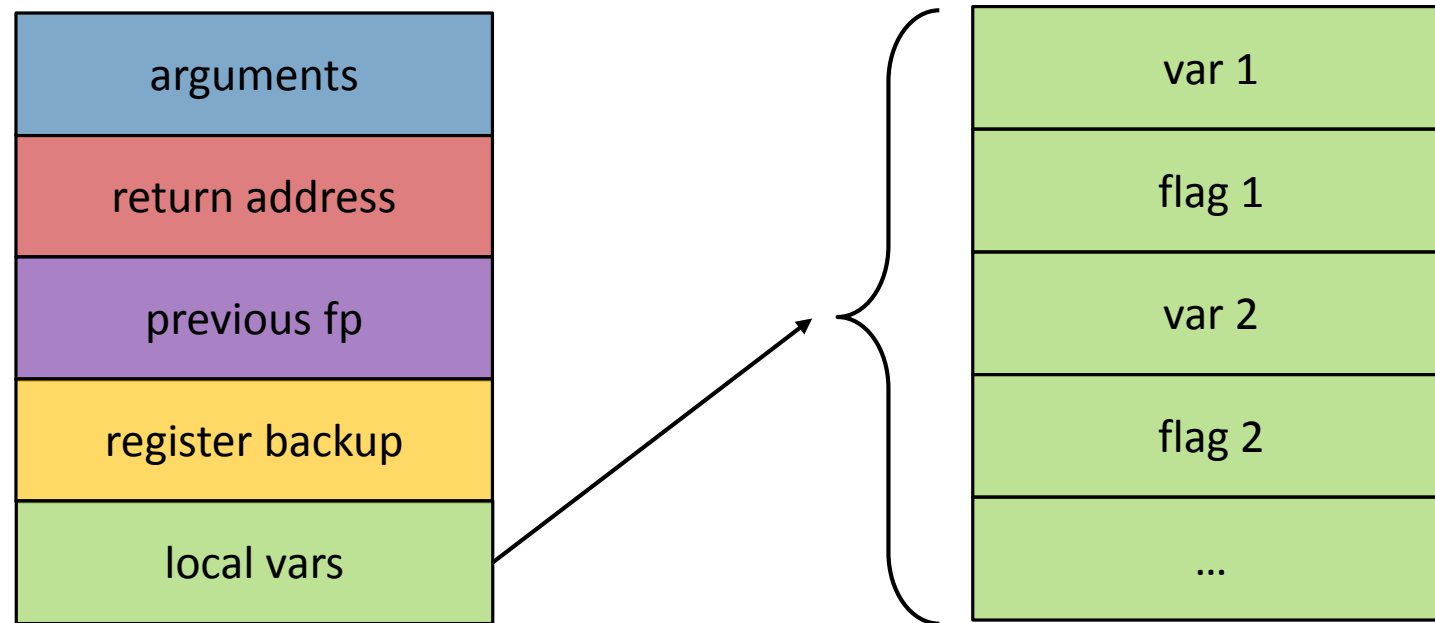
Add support for detecting accesses to uninitialized local variables.  
Describe the required changes in the code generation step.

For example:

```
void f() {  
    int x;  
    int y := 7;  
    if (y > 10) {  
        x := 100;  
    }  
    int z := x + 1;  
}
```

# Solution

High level idea:



# Solution

- Initialize each variable flag to zero on function entry
- When writing to a local variable, set its flag to 1
- When reading a local variable, check if its flag is 1

var 1
flag 1
var 2
flag 2
...



# Solution

- Initialize each variable flag to zero on function entry
- When writing to a local variable, set its flag to 1
- When reading a local variable, check if its flag is 1

prologue:

```
...  
li $s0, 0  
sw $s0, local_1_flag_offset($fp)  
li $s0, 0  
sw $s0, local_2_flag_offset($fp)  
...
```

var 1
flag 1
var 2
flag 2
...

# Solution

- Initialize each variable flag to zero on function entry
- **When writing to a local variable, set its flag to 1**
- When reading a local variable, check if its flag is 1

**x = t0**

```
sw $t0, x_offset($fp)
li $s0, 1
sw $s0, x_flag_offset($fp)
```

var 1
flag 1
var 2
flag 2
...

# Solution

- Initialize each variable flag to zero on function entry
- When writing to a local variable, set its flag to 1
- **When reading a local variable, check if its flag is 1**

`t0 = x`

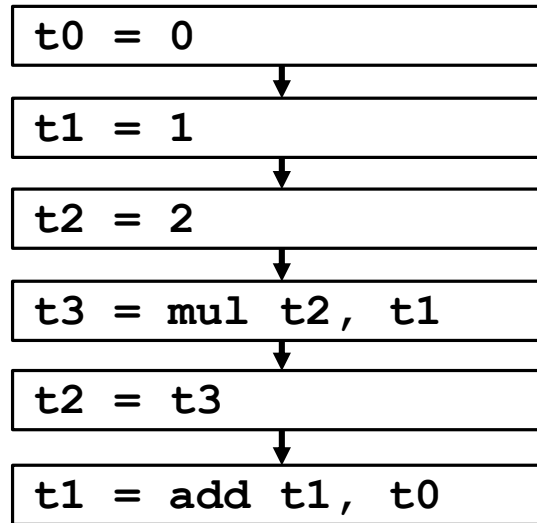
```
lw $s0, x_flag_offset($fp)
beq $s0, 0, abort
lw $t0, x_offset($fp)
```

var 1
flag 1
var 2
flag 2
...

# Question

Apply the register allocation algorithm with 3 registers (R1,R2,R3)  
R1 can't hold a result of a multiplication operation.

```
t0 = 0
t1 = 1
t2 = 2
t3 = t2 * t1
t2 = t3
t1 = t1 + t0
```



IN

{ }

{ }

{ }

{ }

{ }

{ }

OUT

{ }

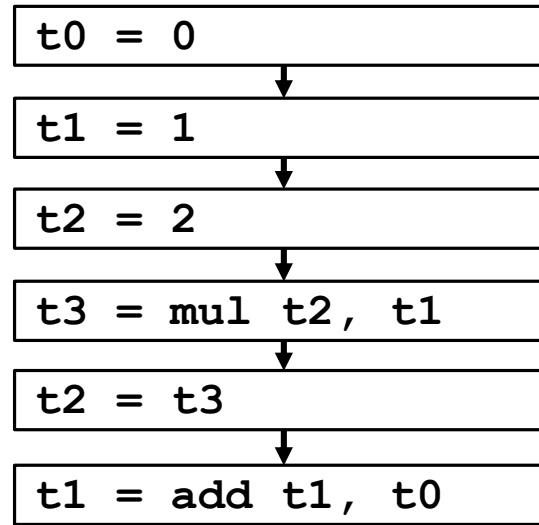
{ }

{ }

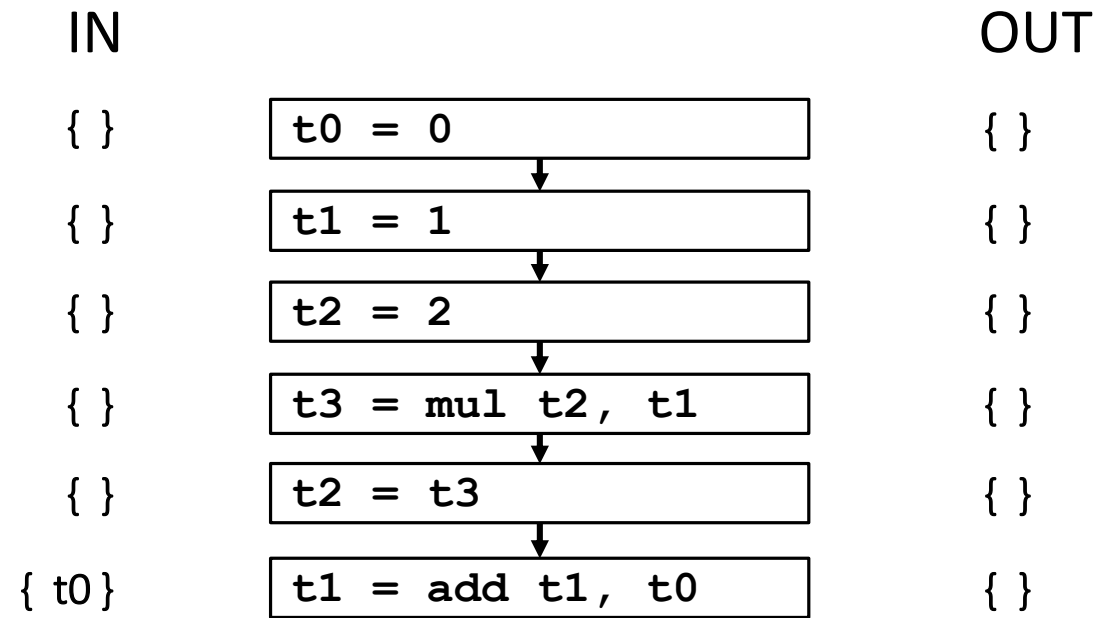
{ }

{ }

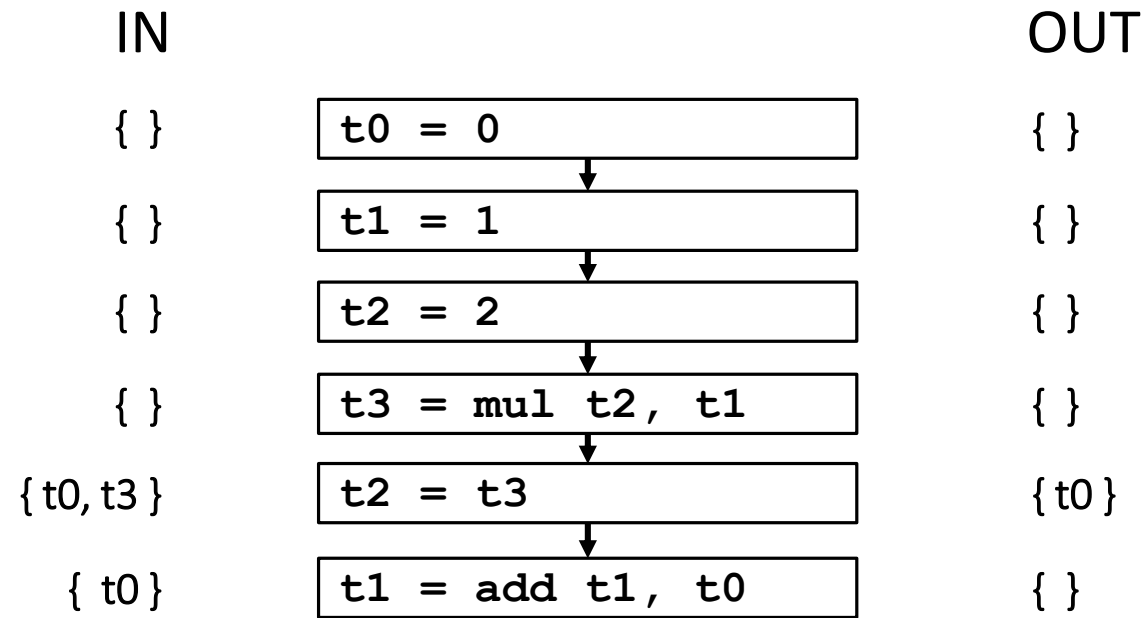
{ }



initialization

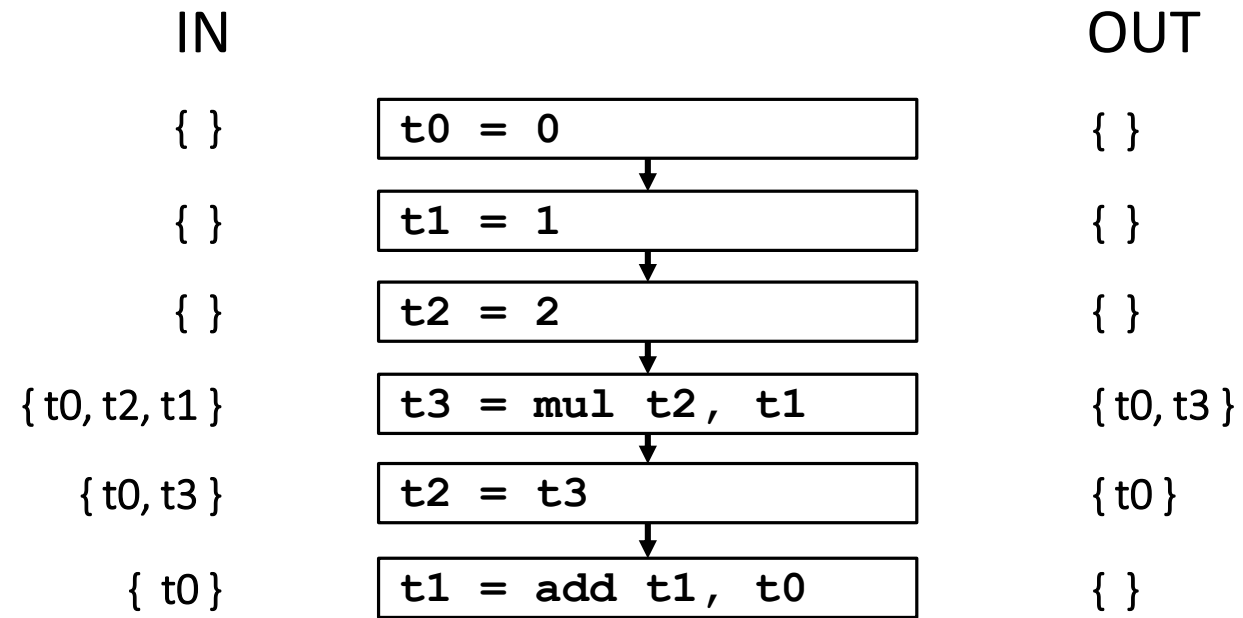


first iteration

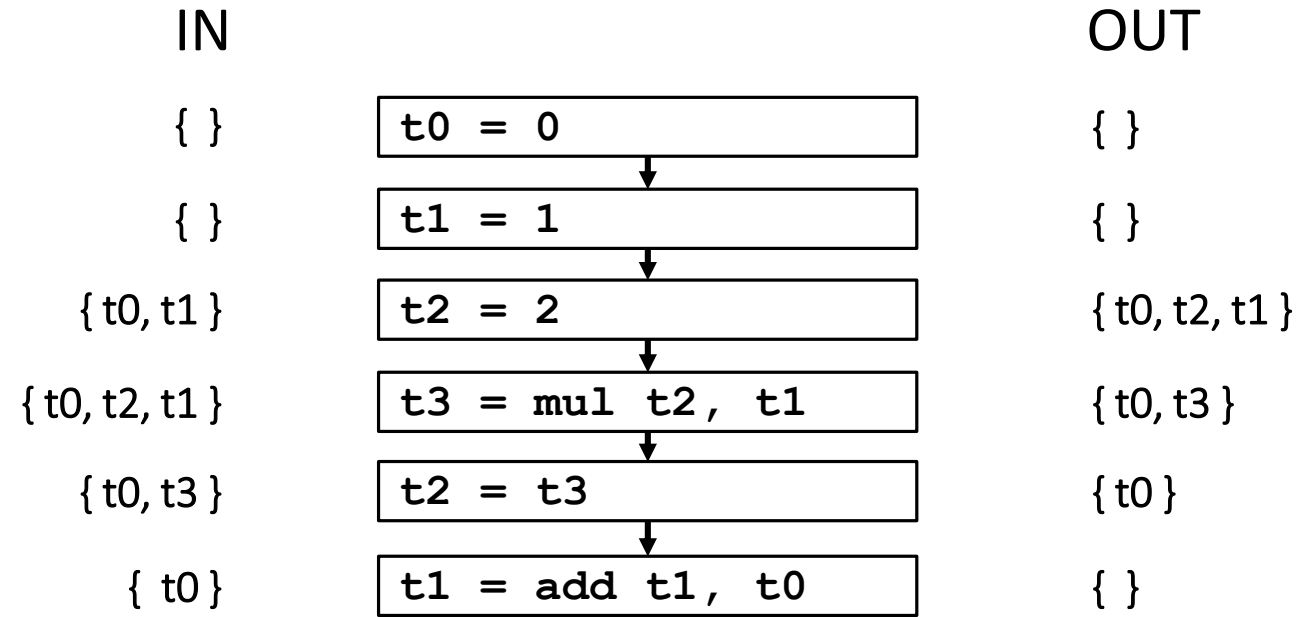


first iteration

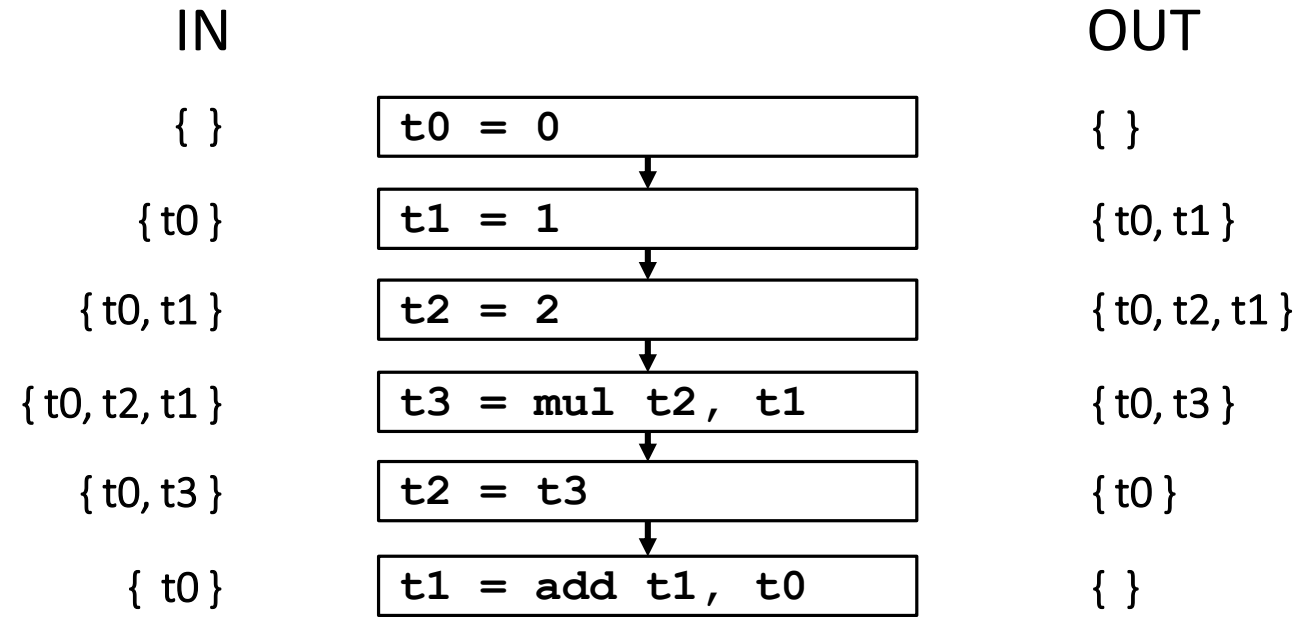




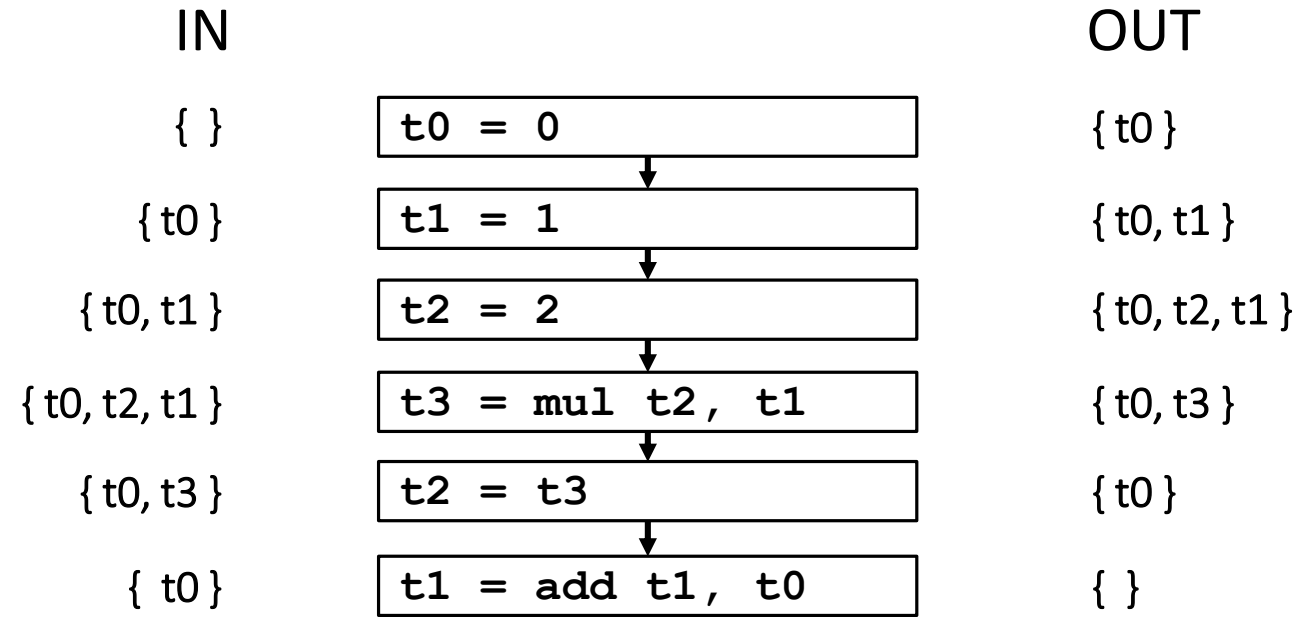
first iteration



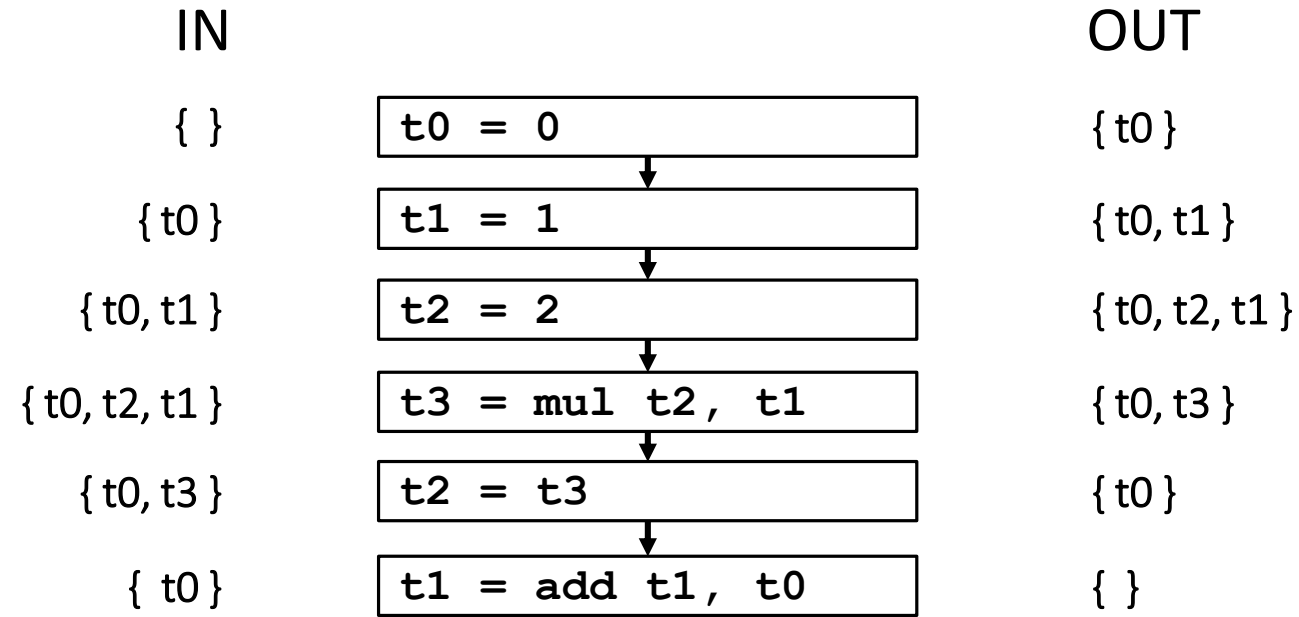
first iteration



first iteration



first iteration



second iteration...

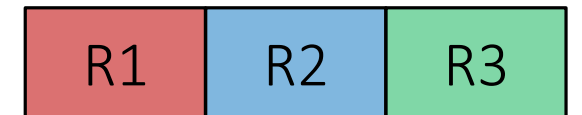
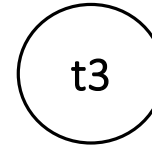
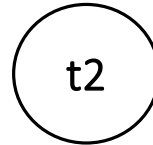
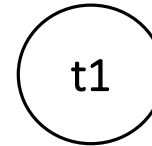
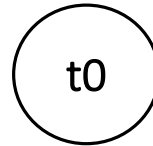
# Interference Graph

$\{t_0\}$

$\{t_0, t_1\}$

$\{t_0, t_2, t_1\}$

$\{t_0, t_3\}$



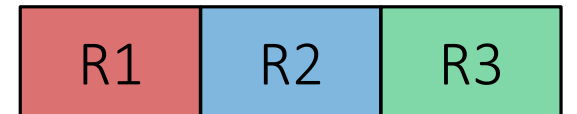
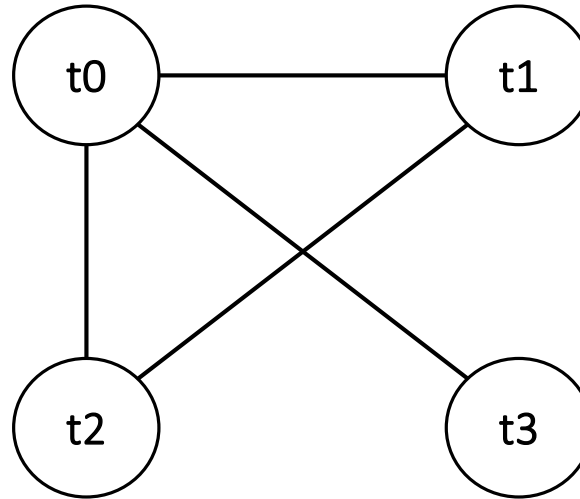
# Interference Graph

$\{t_0\}$

$\{t_0, t_1\}$

$\{t_0, t_2, t_1\}$

$\{t_0, t_3\}$



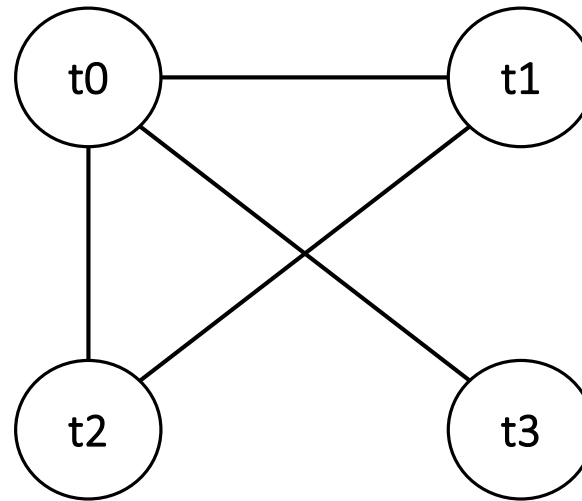
# Interference Graph

{ t0 }

{ t0, t1 }

{ t0, t2, t1 }

{ t0, t3 }



t0 = 0

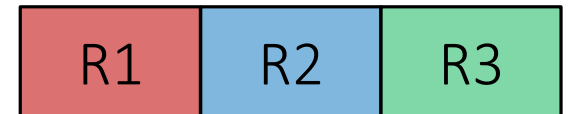
t1 = 1

t2 = 2

t3 = t2 \* t1

t2 = t3

t1 = t1 + t0





# Interference Graph

{ t0 }

{ t0, t1 }

{ t0, t2, t1 }

{ t0, t3 }

t0 = 0

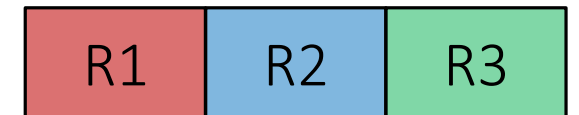
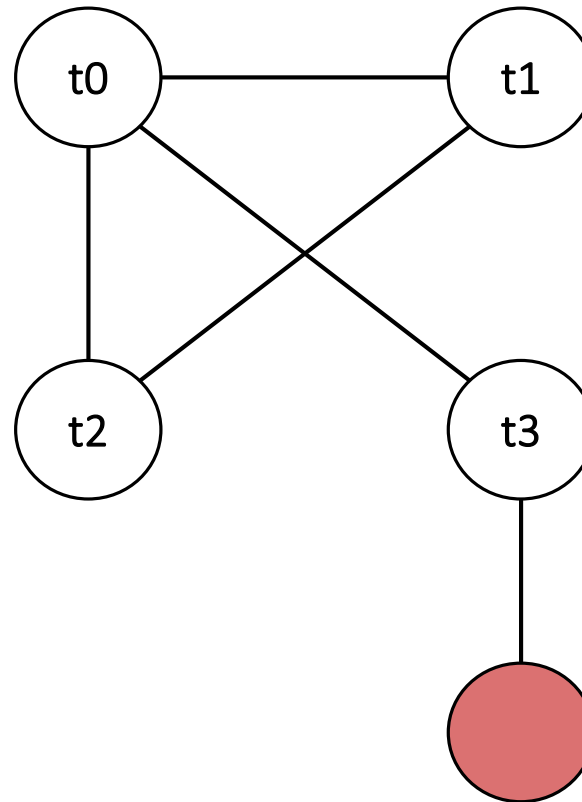
t1 = 1

t2 = 2

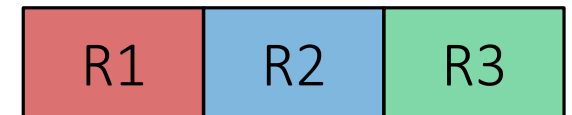
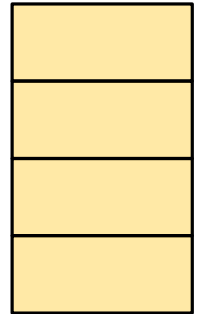
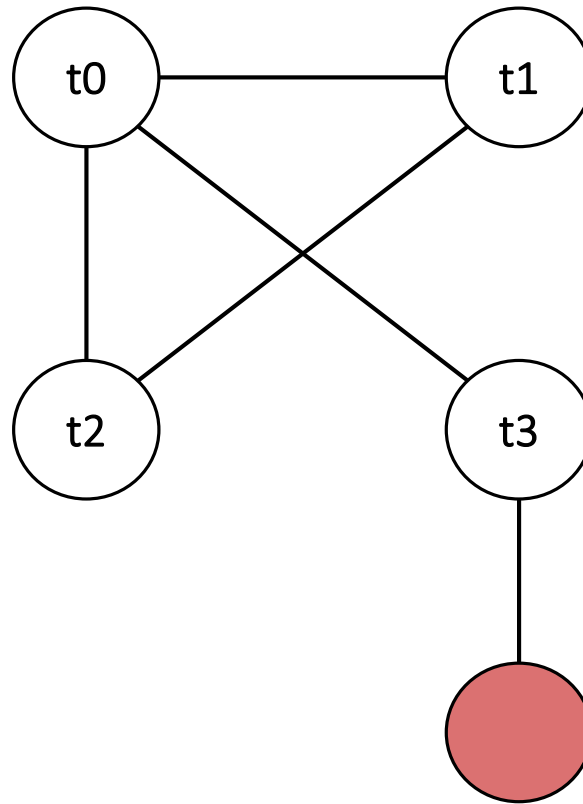
t3 = t2 \* t1

t2 = t3

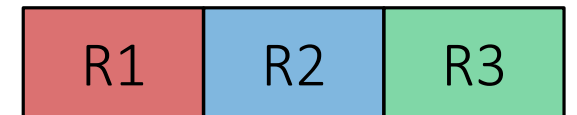
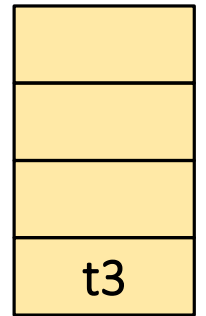
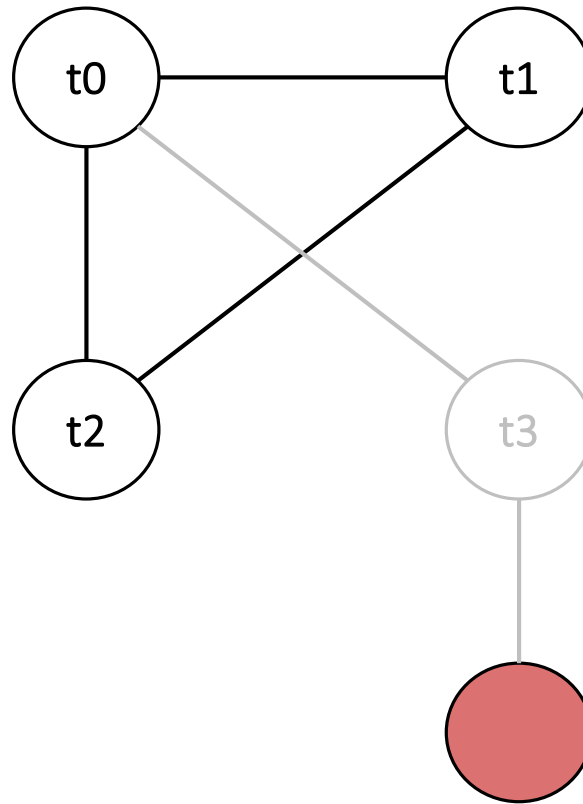
t1 = t1 + t0



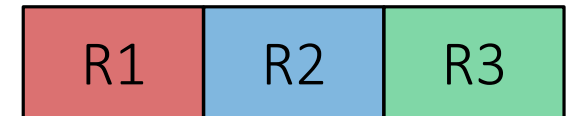
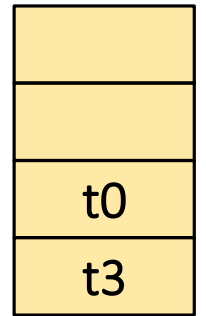
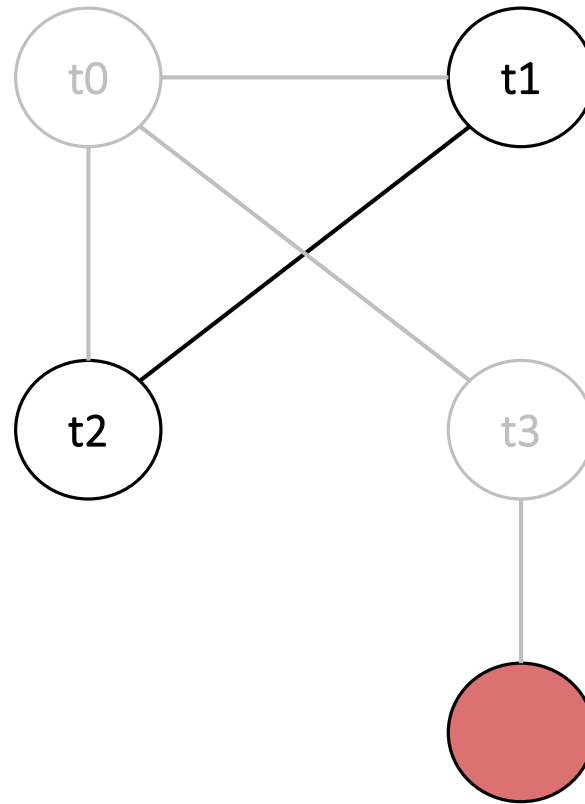
# Graph Coloring



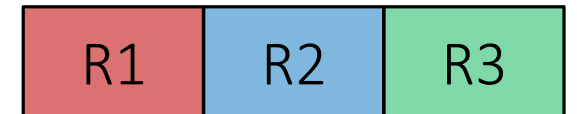
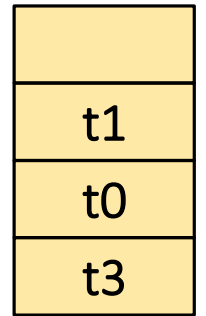
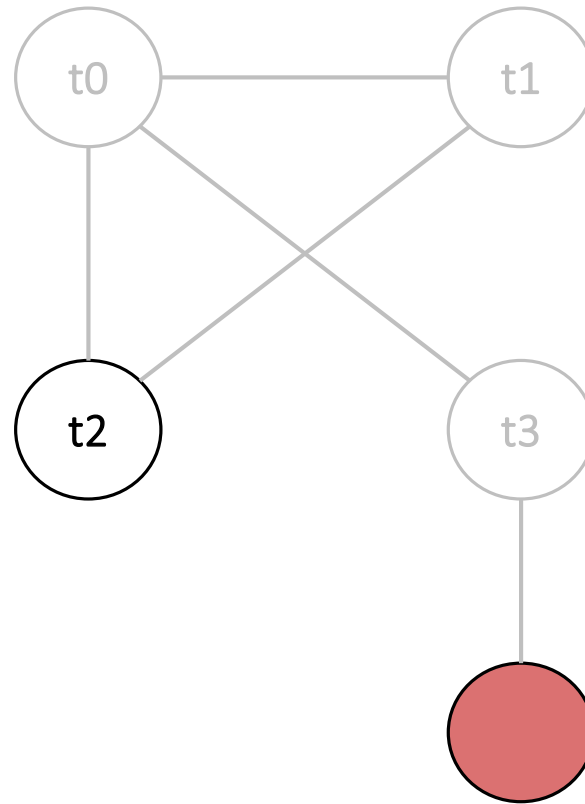
# Graph Coloring



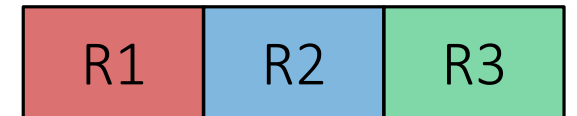
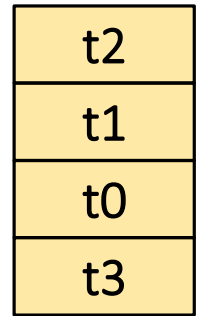
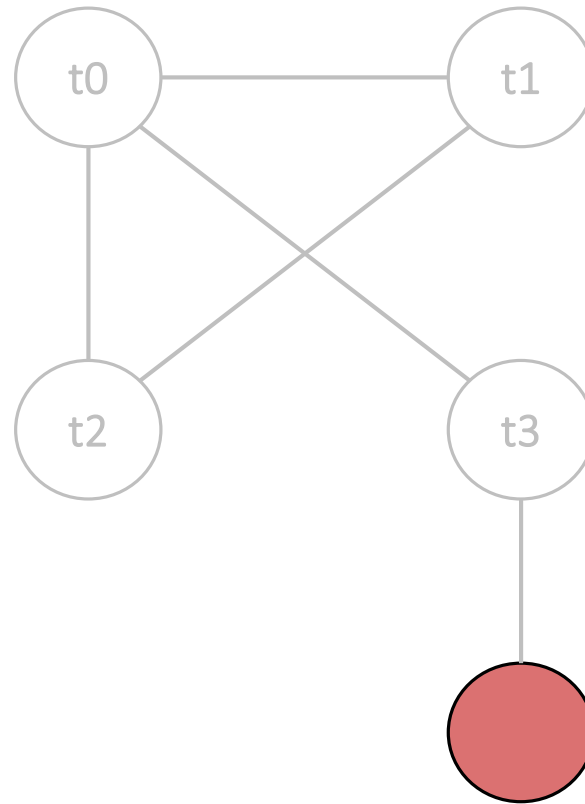
# Graph Coloring



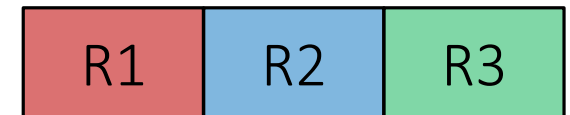
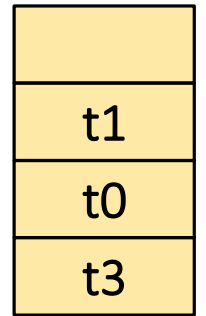
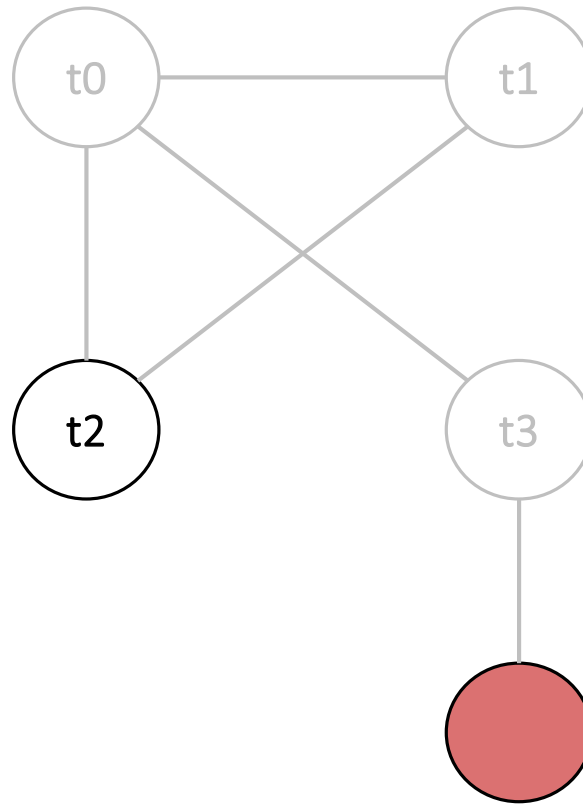
# Graph Coloring



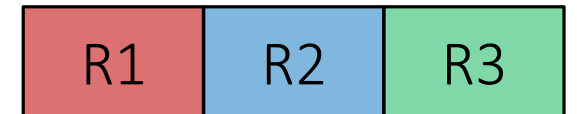
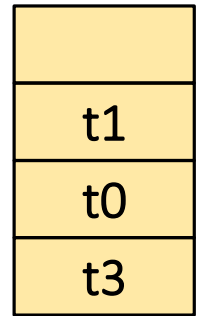
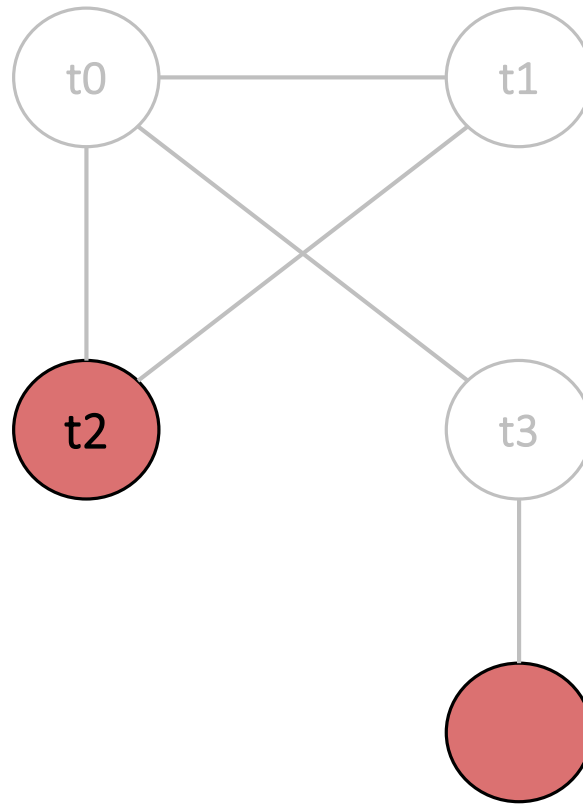
# Graph Coloring



# Graph Coloring

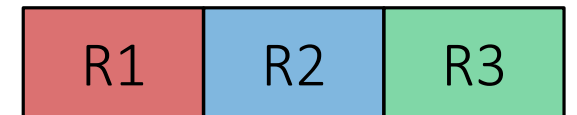
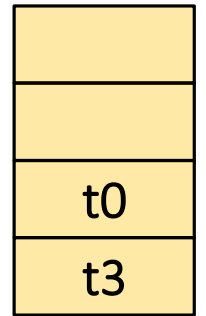
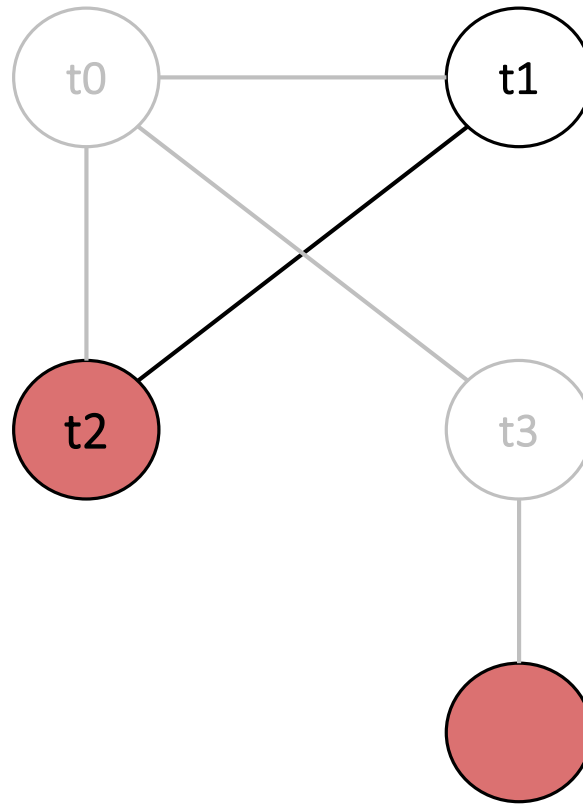


# Graph Coloring

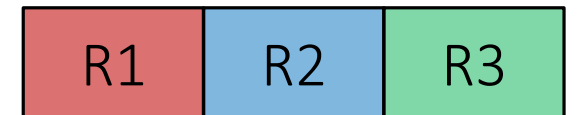
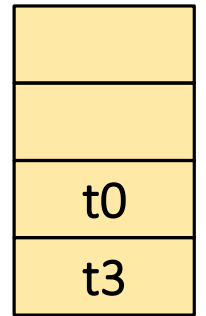
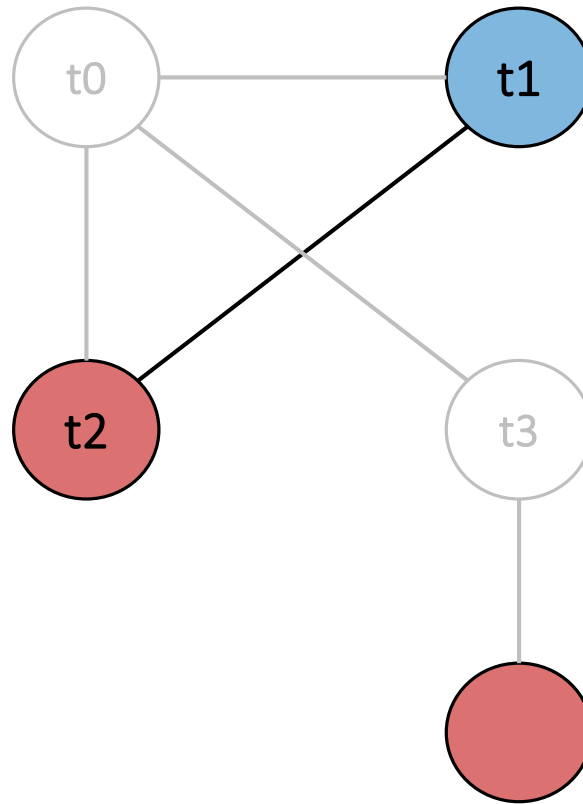




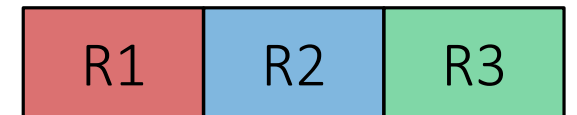
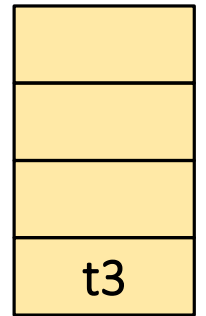
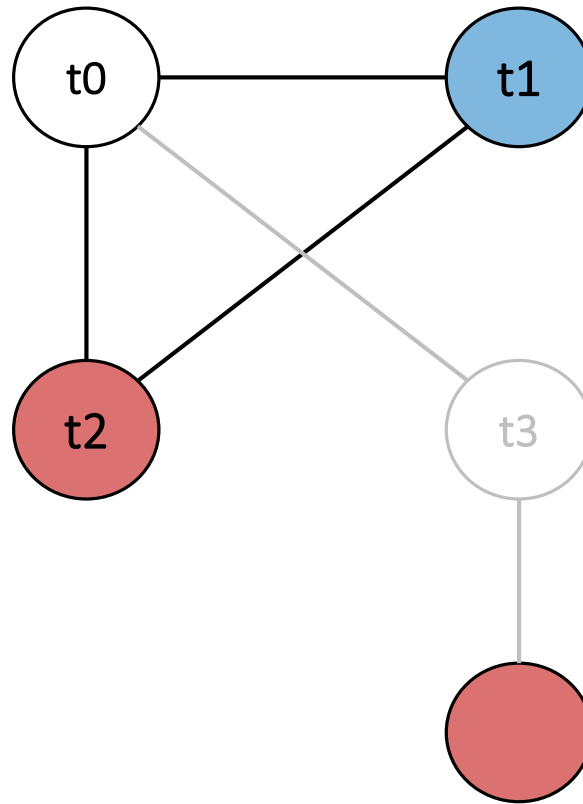
# Graph Coloring



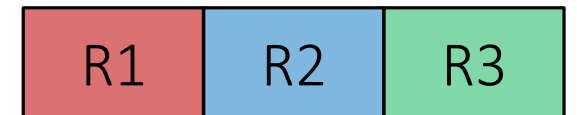
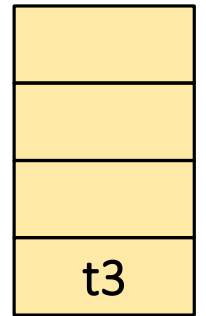
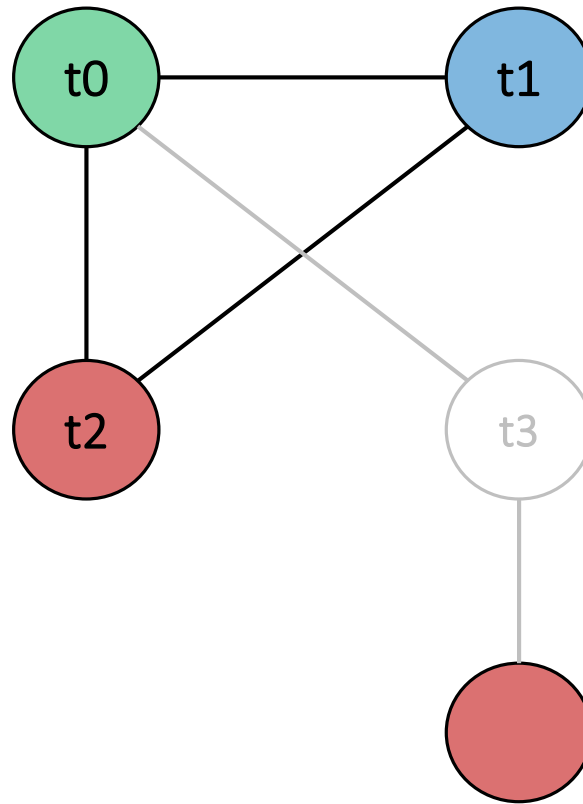
# Graph Coloring



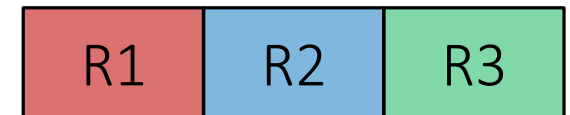
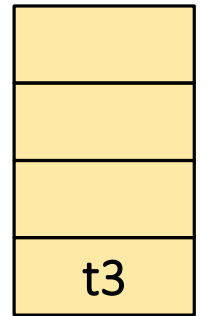
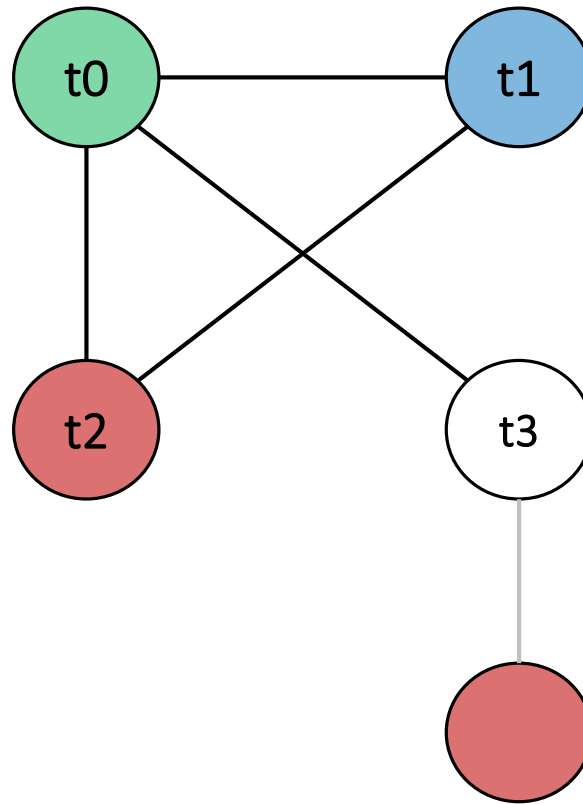
# Graph Coloring



# Graph Coloring



# Graph Coloring



# Graph Coloring

