# 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

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 only method invocations are of methods which are located at offset 4 in the virtual table.

compiler construction
compiling
linking
runtime

# Question

The only method invocations are of methods which are 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:

- $N \to \alpha\beta.$

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

- $N \to .\alpha\beta$
- $N \to \alpha.\beta$

# 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 $A \to \alpha . N\beta \in S$ then for each rule $N \to \gamma$:
  - $N \to . \gamma \in S$

# SLR(1)

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

# LR(1)

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

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

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

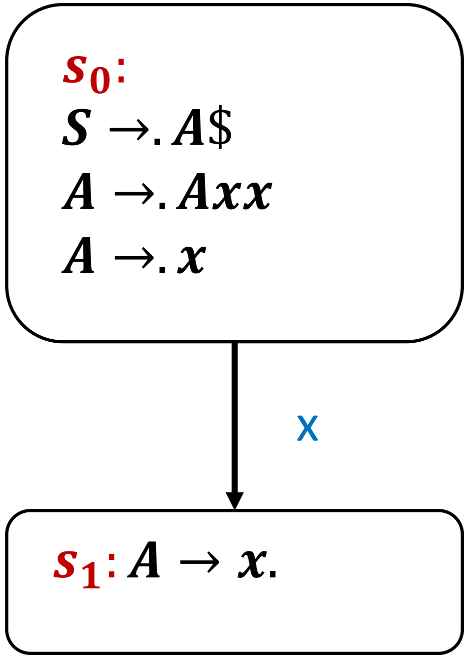
$\boldsymbol{s_0}$:

$\boldsymbol{S \rightarrow .\, A\$}$

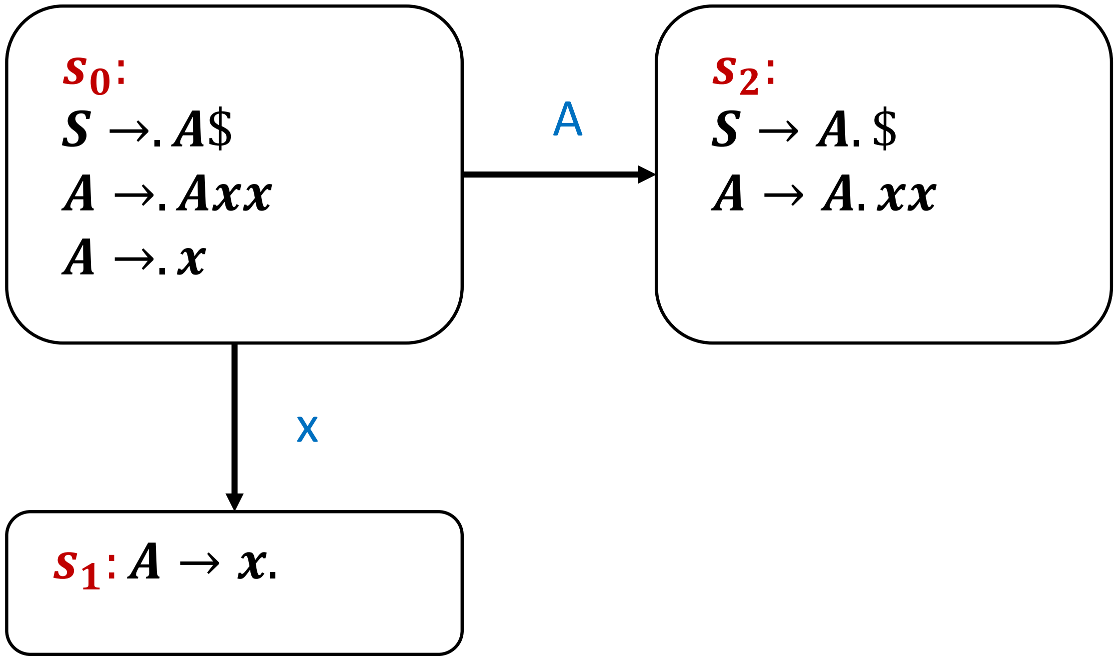
$\boldsymbol{A \rightarrow .\, Axx}$

$\boldsymbol{A \rightarrow .\, x}$

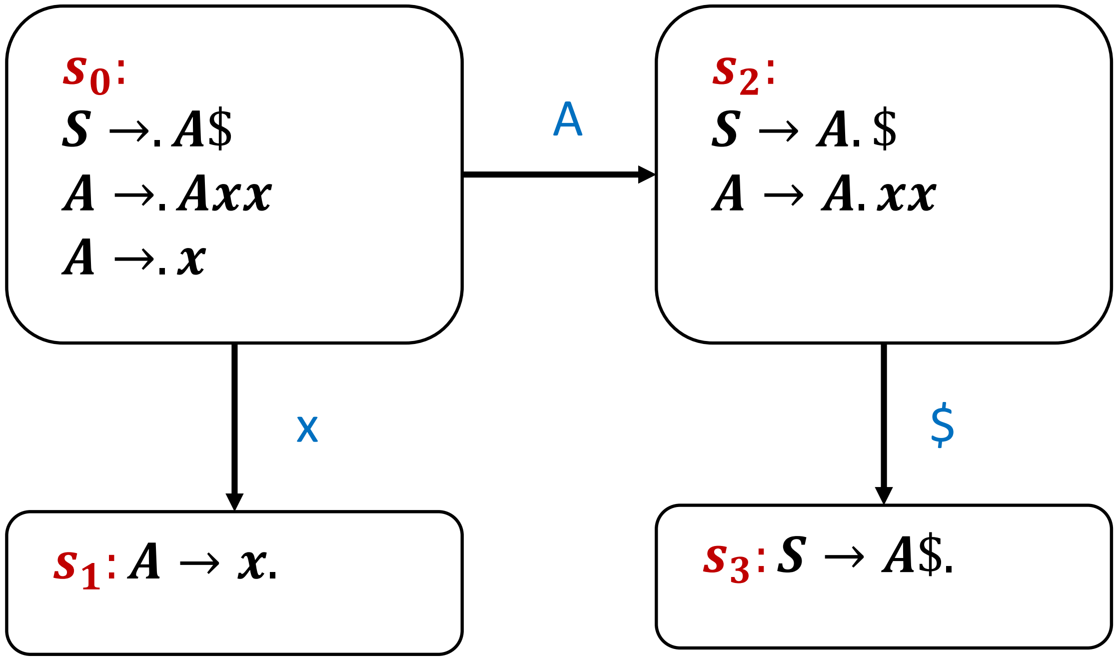
$$S \rightarrow A\$$$
$$A \rightarrow Axx$$
$$A \rightarrow x$$



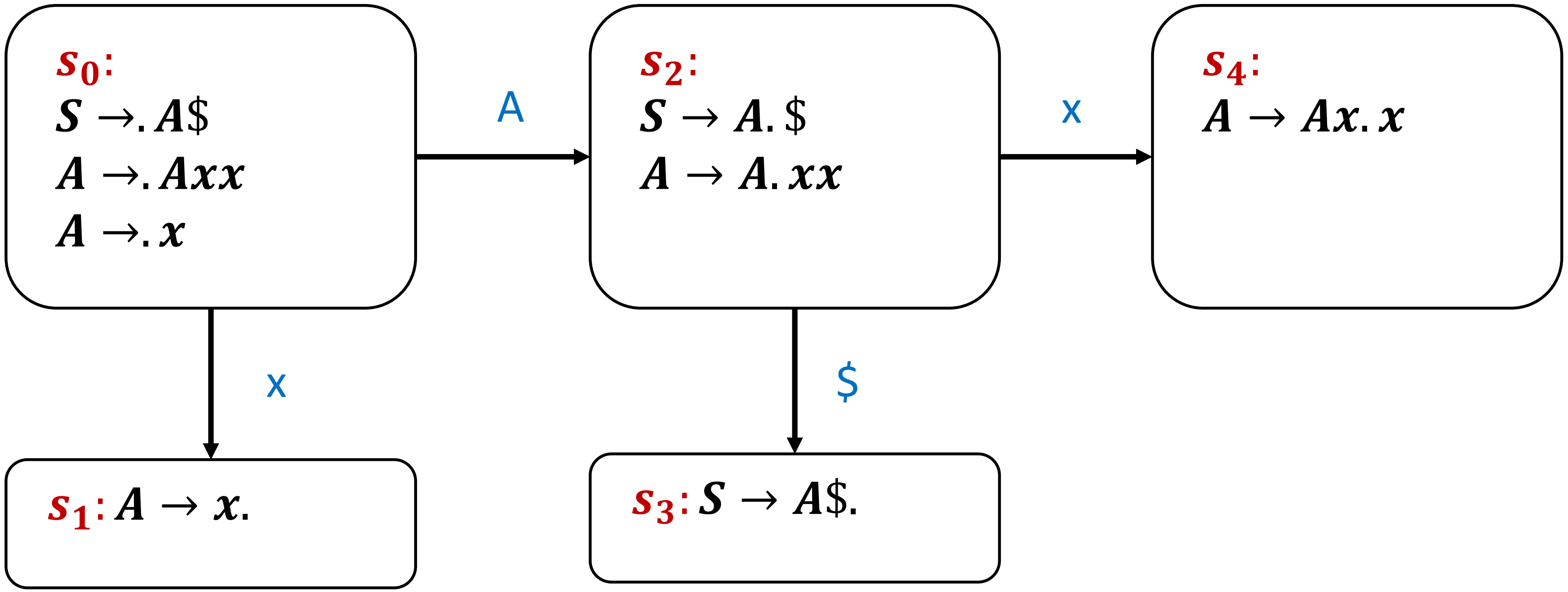

$S \to A\$$
$A \to Axx$
$A \to x$



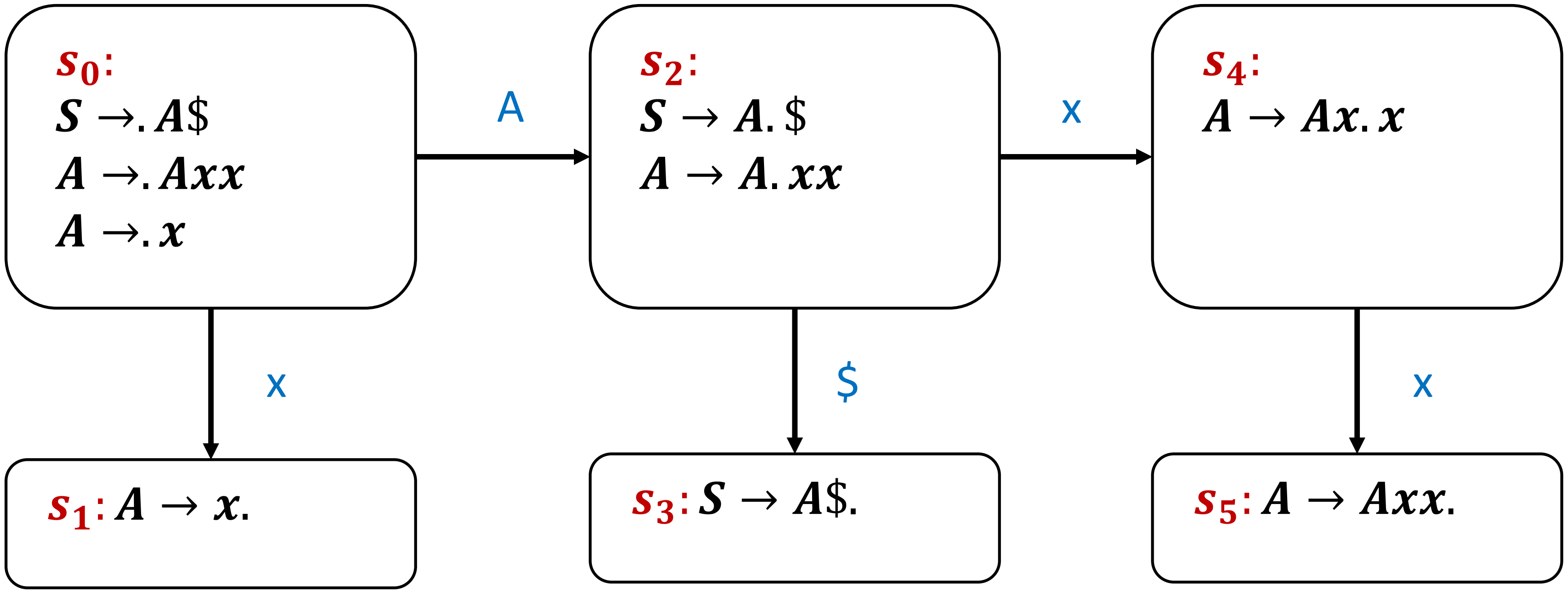

$S \rightarrow A\$$
$A \rightarrow Axx$
$A \rightarrow x$




$S \to A\$$
$A \to Axx$
$A \to x$




$S \to A\$$
$A \to Axx$
$A \to x$

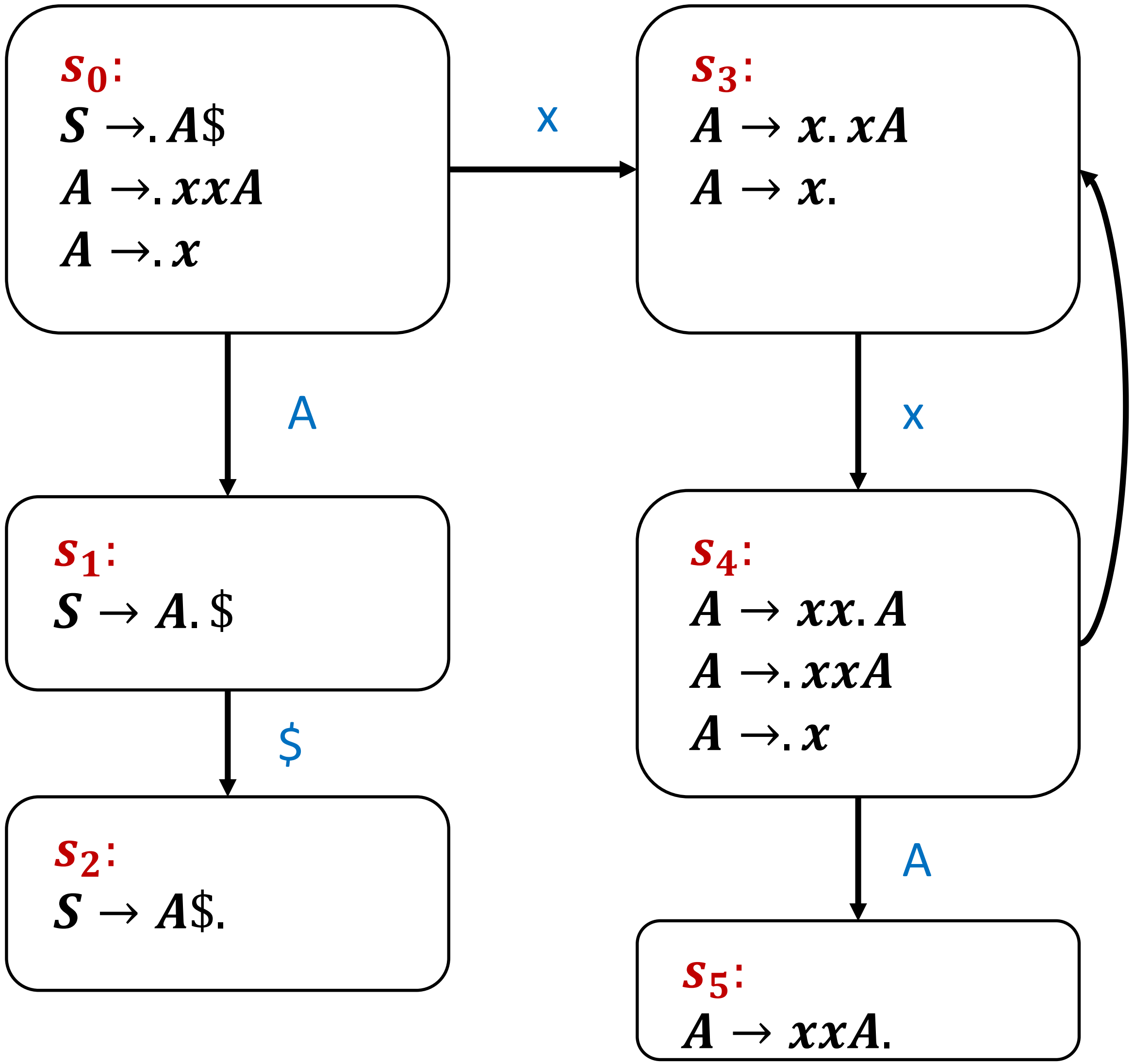



$$S \rightarrow A\$$$
$$A \rightarrow Axx$$
$$A \rightarrow x$$

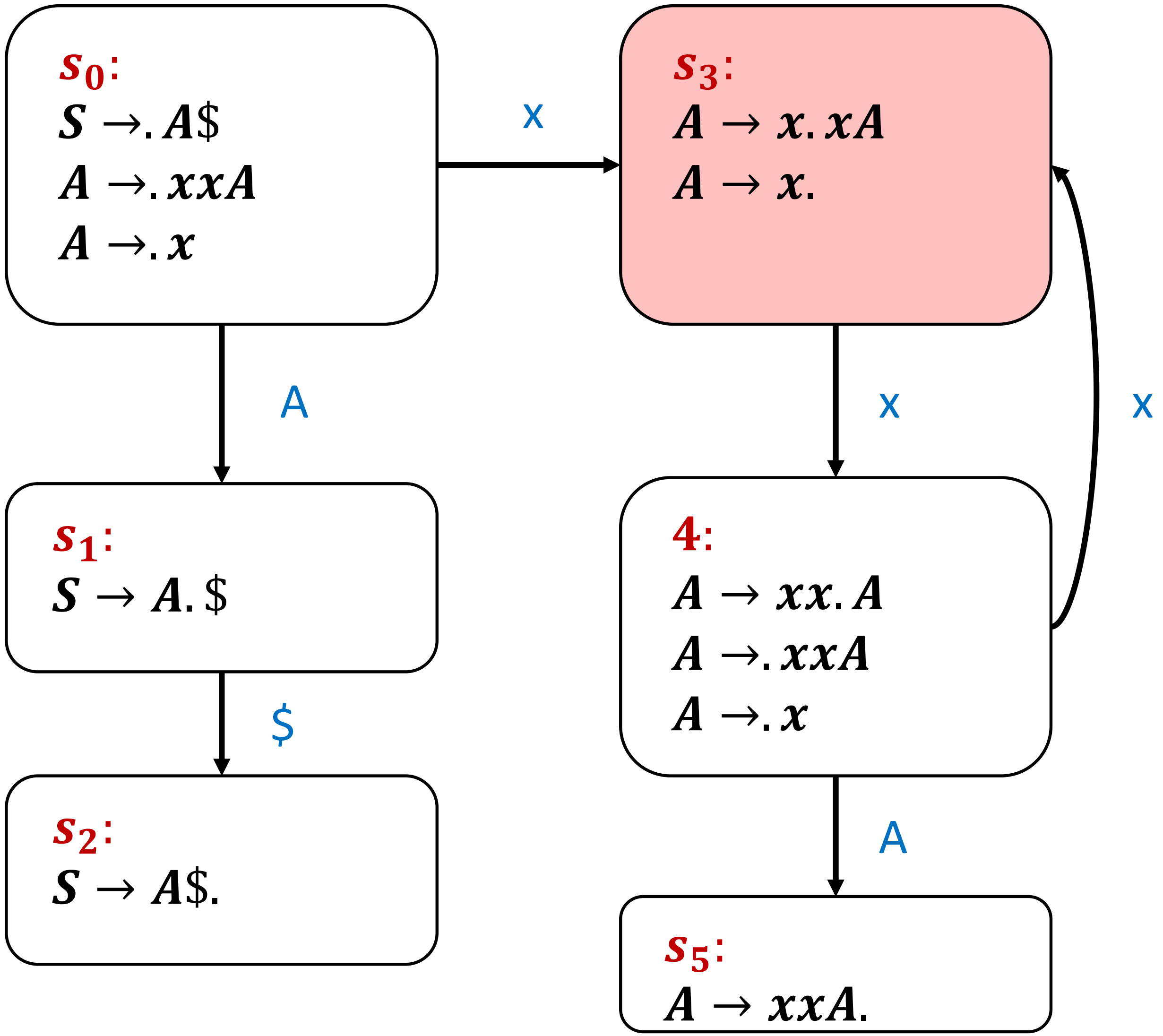
# Question

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

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



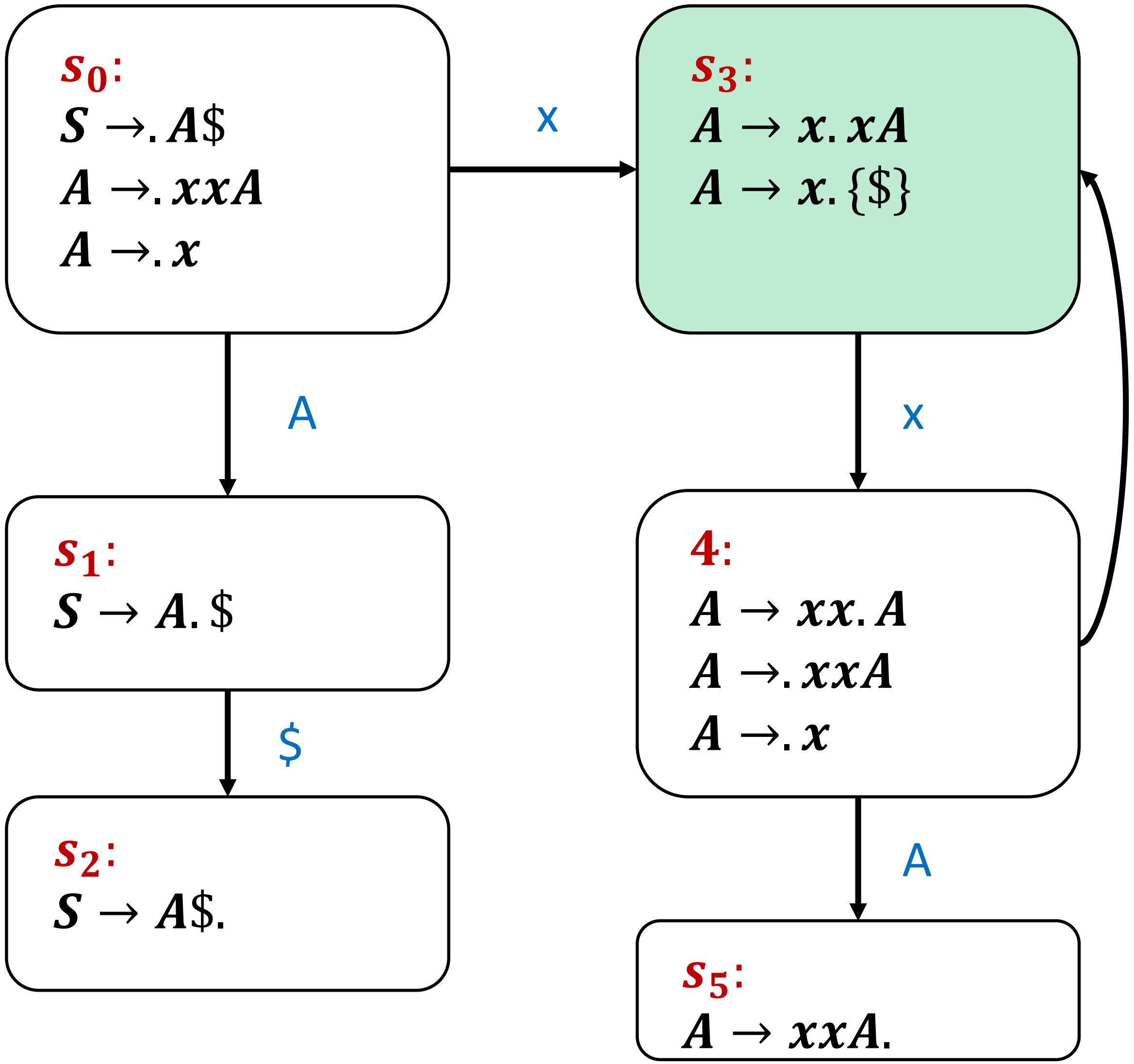

$S \rightarrow A\$$
$A \rightarrow xxA$
$A \rightarrow x$




$S \rightarrow A\$$

$A \rightarrow xxA$

$A \rightarrow x$



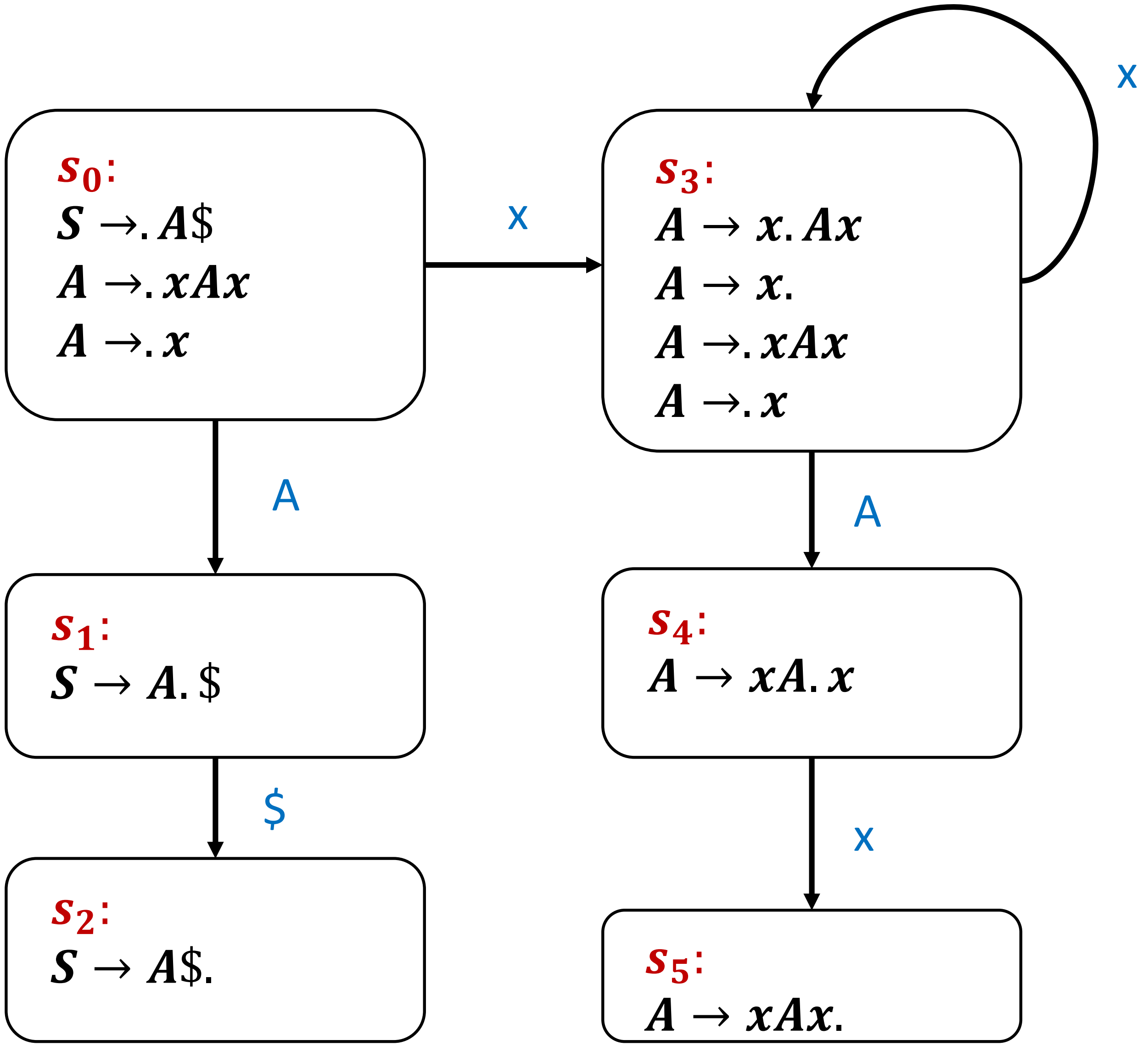

$$Follow(A) = \{\$\}$$

$S \to A\$$
$A \to xxA$
$A \to x$

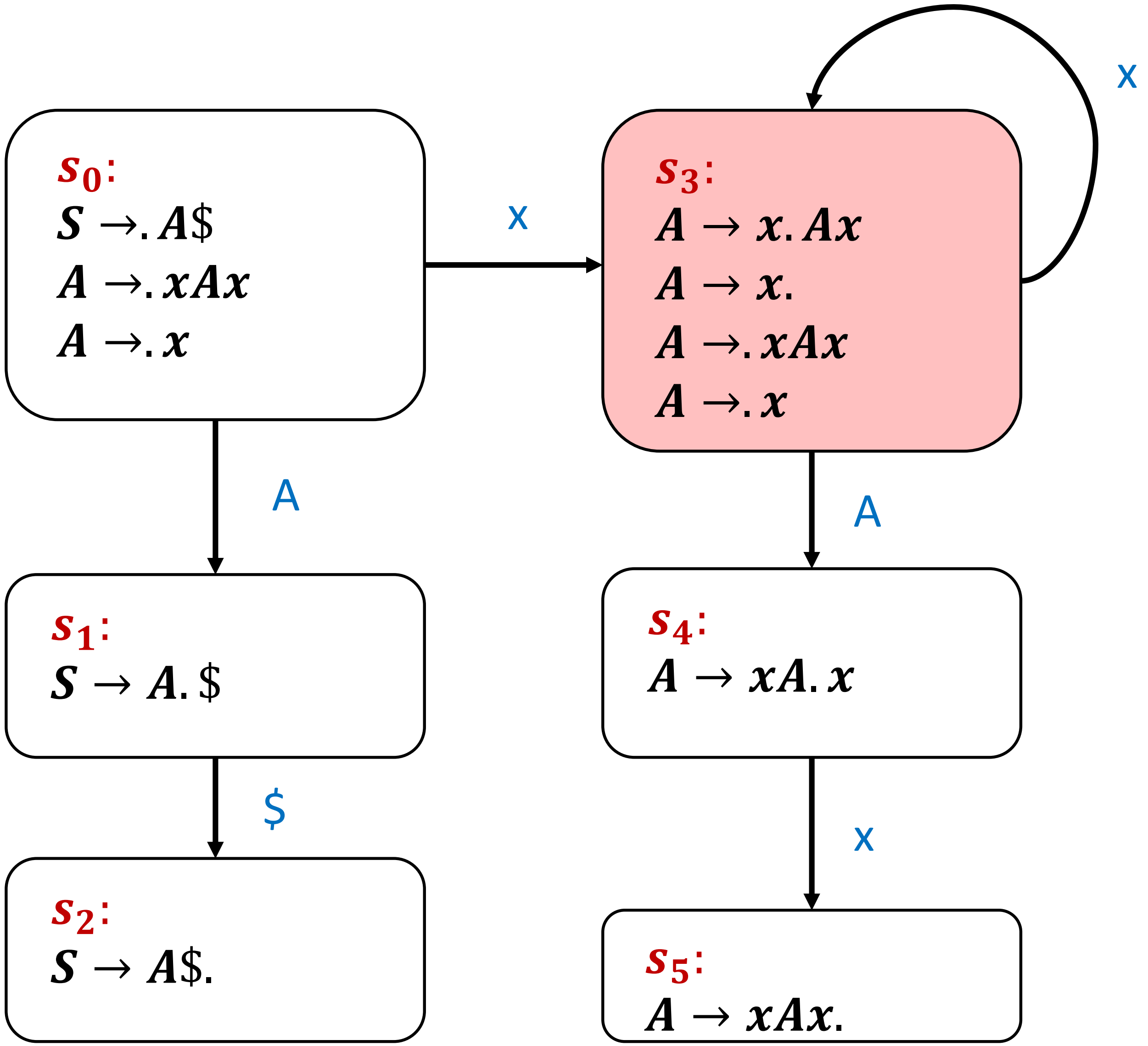
# Question

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

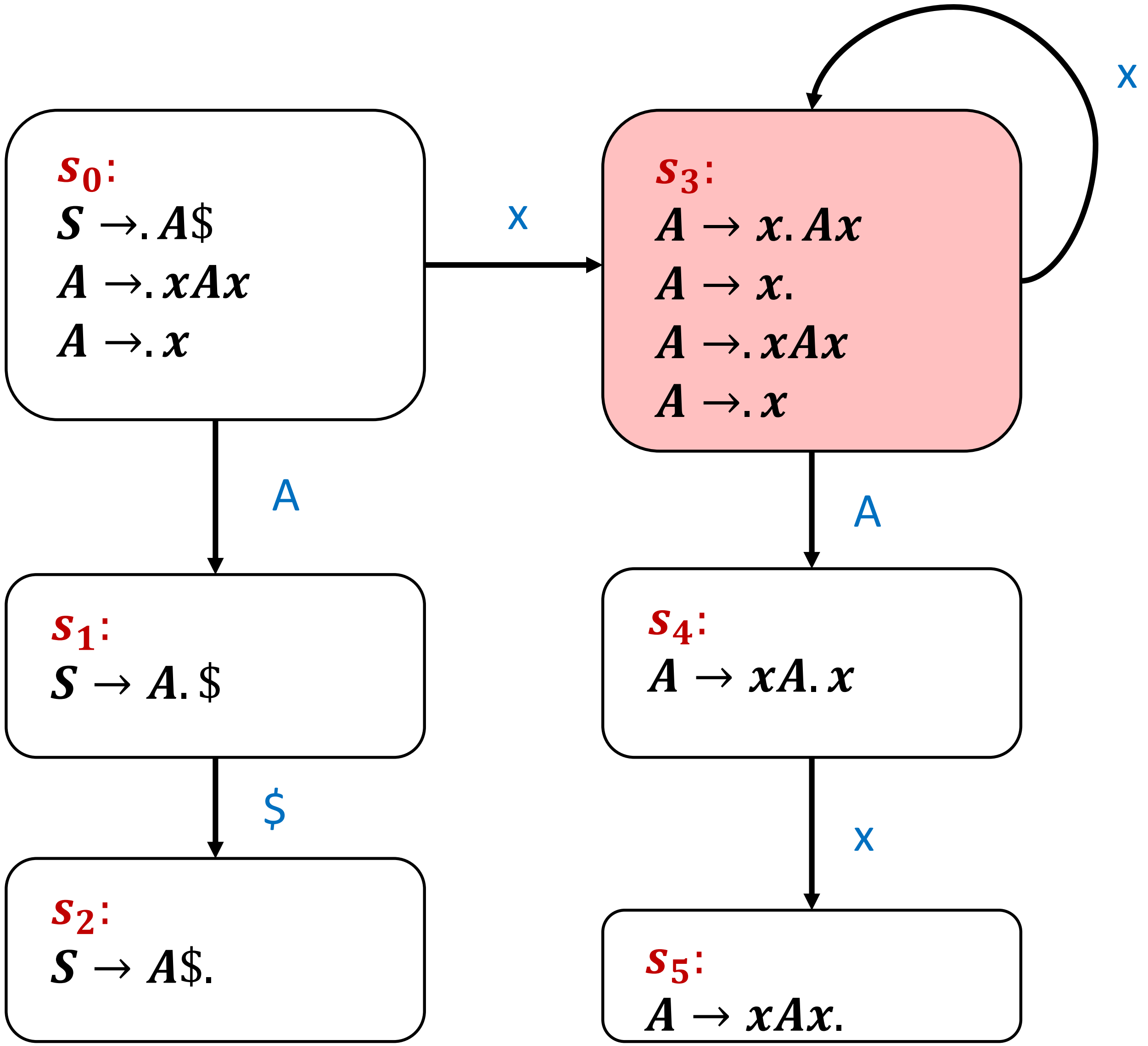
- $S \rightarrow A\$$
- $A \rightarrow xAx$
- $A \rightarrow x$




$S \to A\$$
$A \to xAx$
$A \to x$




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




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

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

$s_0$:

$S \rightarrow .A\$ \ \{\$\}$

$A \rightarrow .xAx \ \{\$\}$

$A \rightarrow .x \ \{\$\}$

$S \rightarrow A\$$

$A \rightarrow xAx$

$A \rightarrow x$

$s_0$:

$S \rightarrow .A\$ \ \{\$\}$

$A \rightarrow .xAx \ \{\$\}$

$A \rightarrow .x \ \{\$\}$

$$s_0 \xrightarrow{A} s_1$$

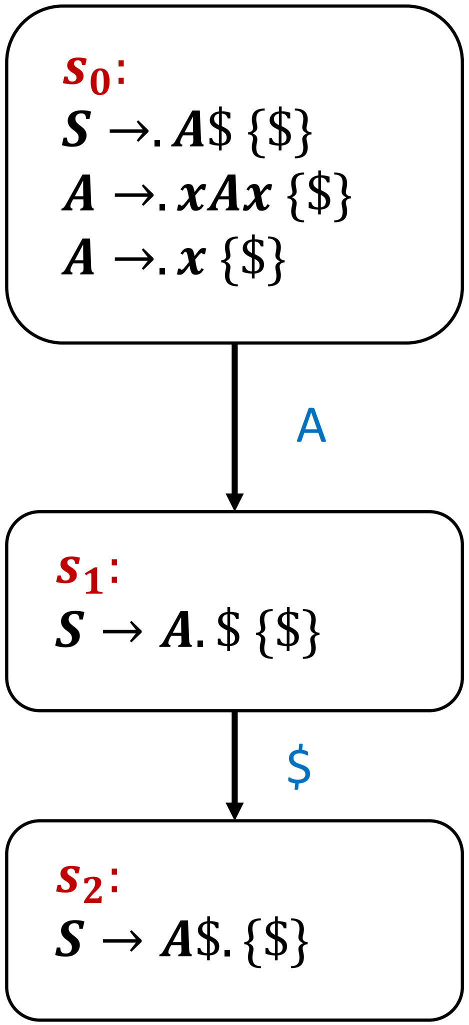
$s_1$:

$S \rightarrow A.\$ \ \{\$\}$

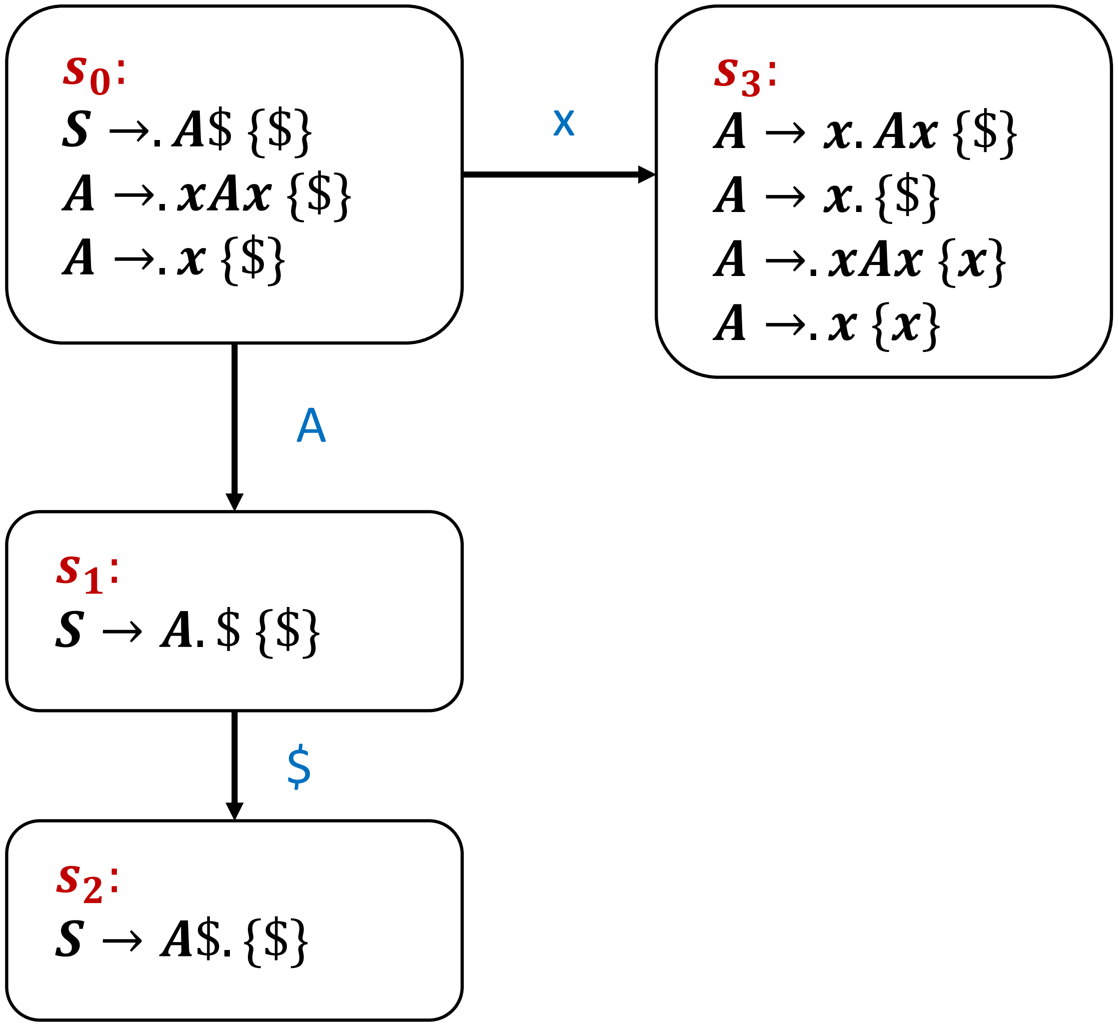
$S \rightarrow A\$$

$A \rightarrow xAx$

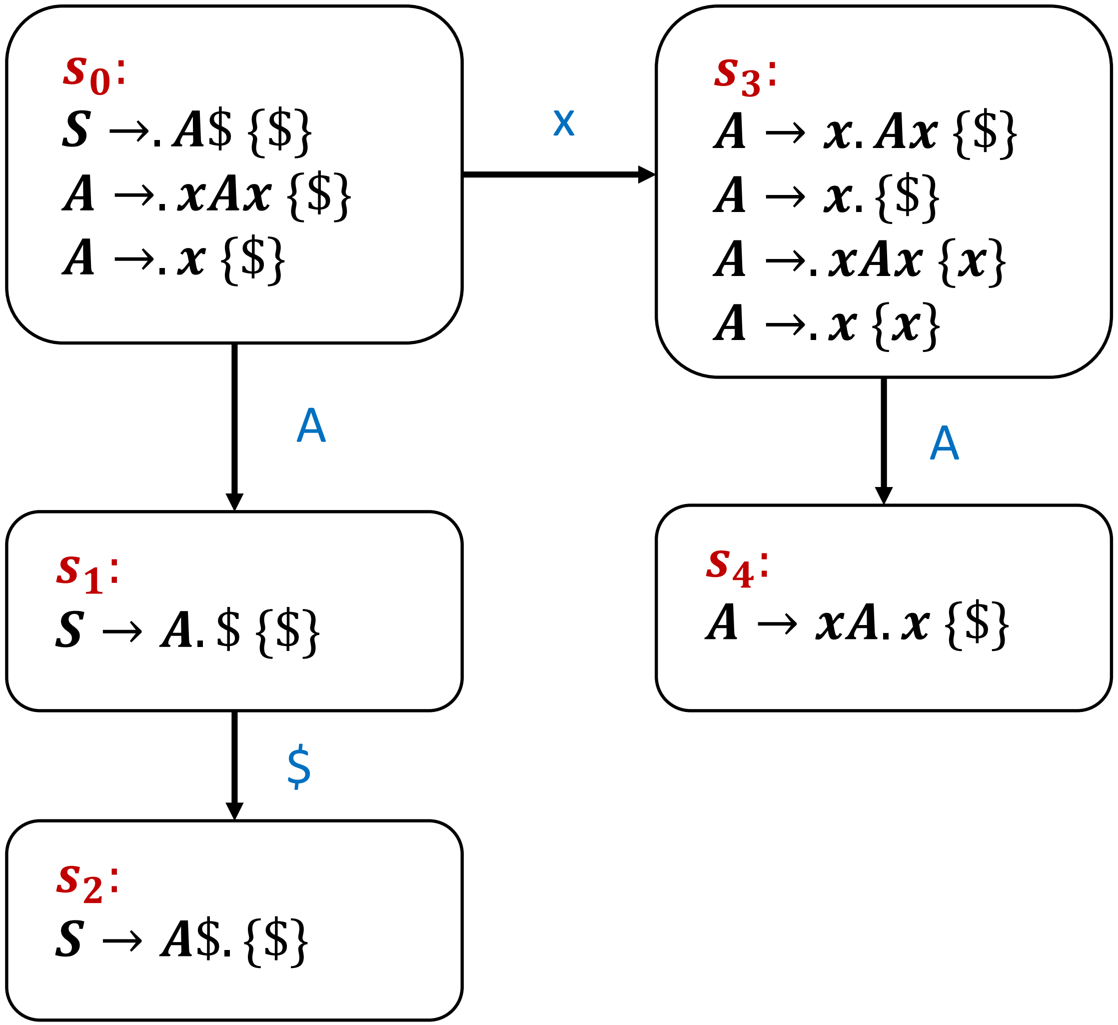
$A \rightarrow x$



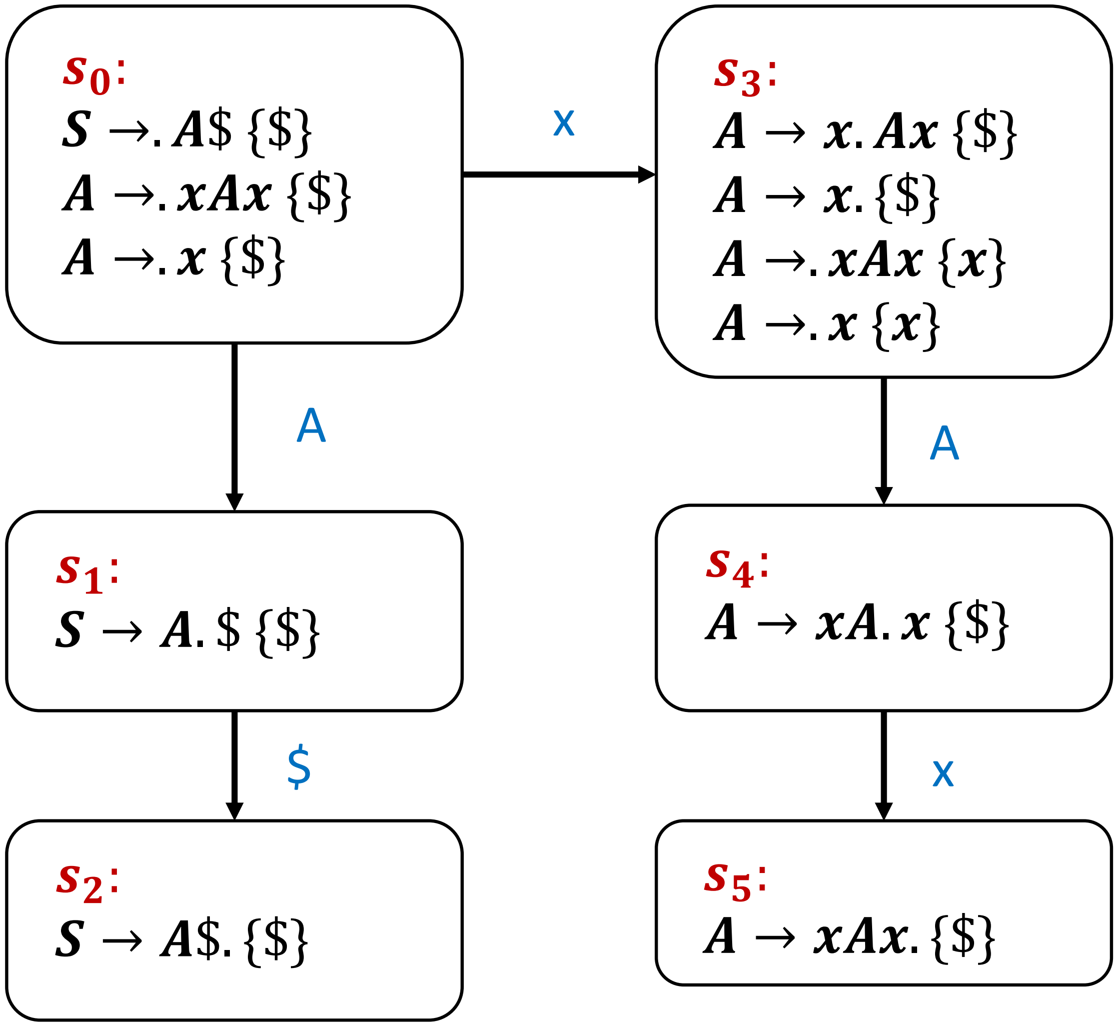

$S \rightarrow A\$$
$A \rightarrow xAx$
$A \rightarrow x$



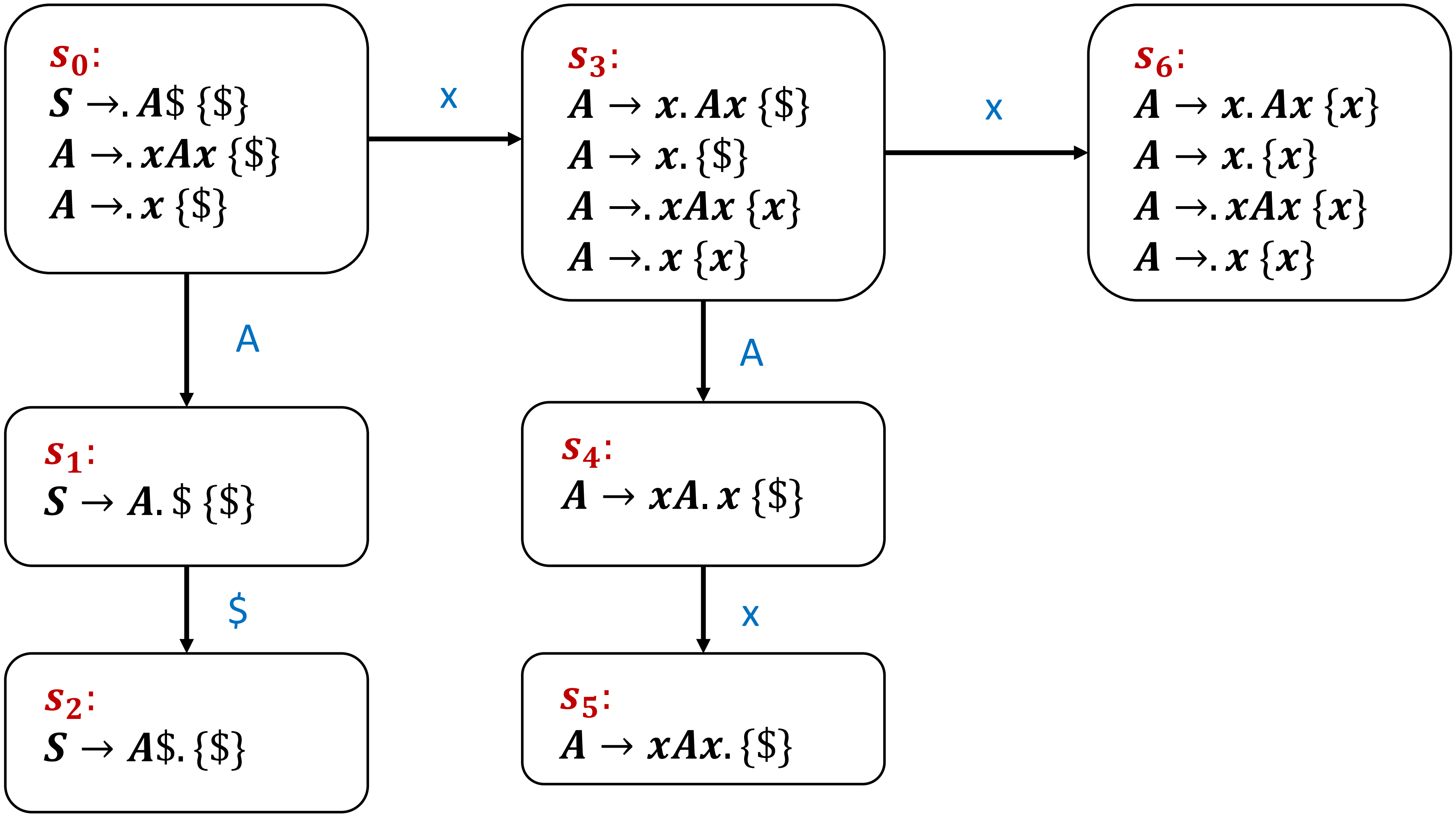

$$S \rightarrow A\$$$
$$A \rightarrow xAx$$
$$A \rightarrow x$$



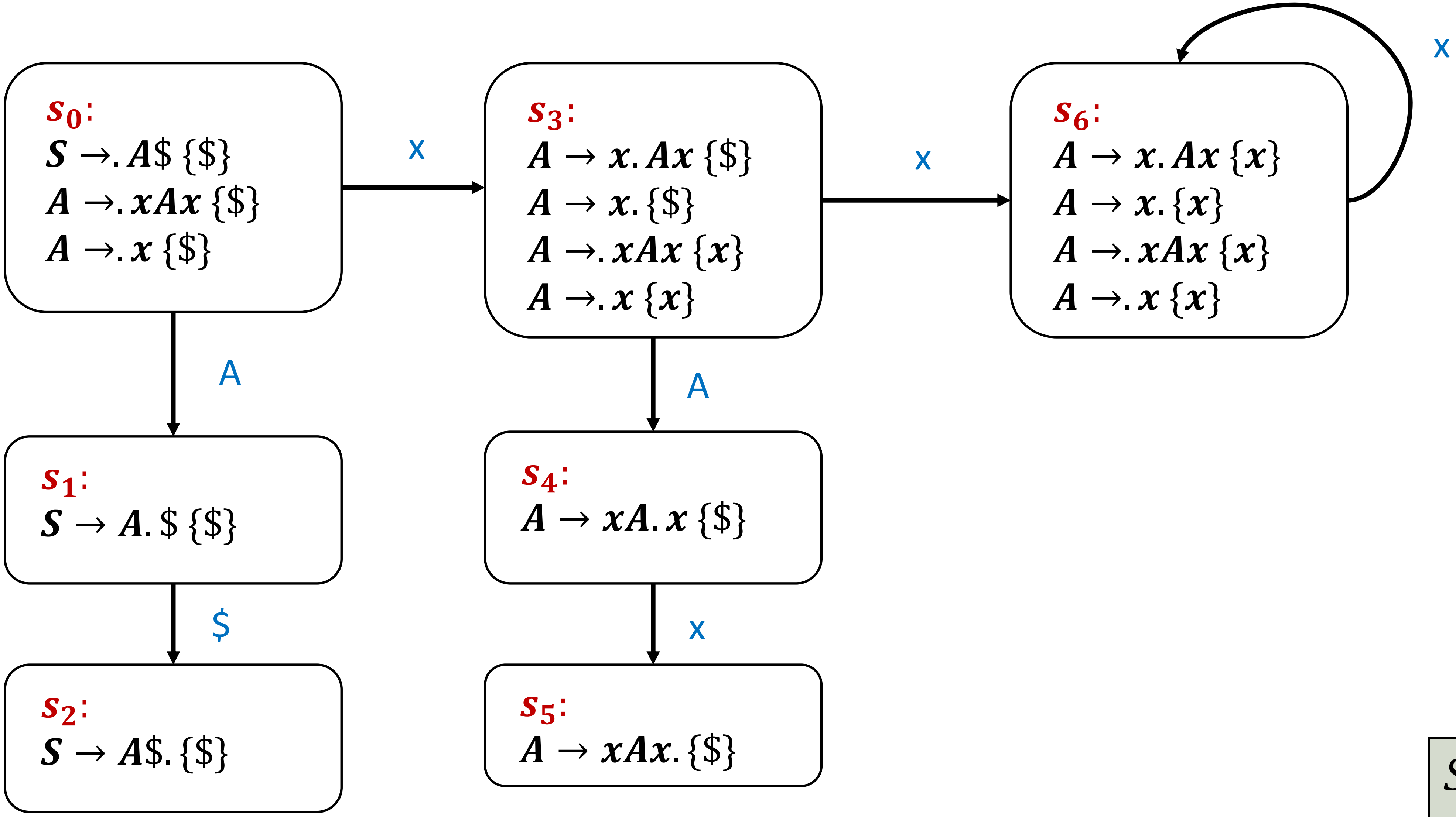

$$S \to A\$$$
$$A \to xAx$$
$$A \to x$$



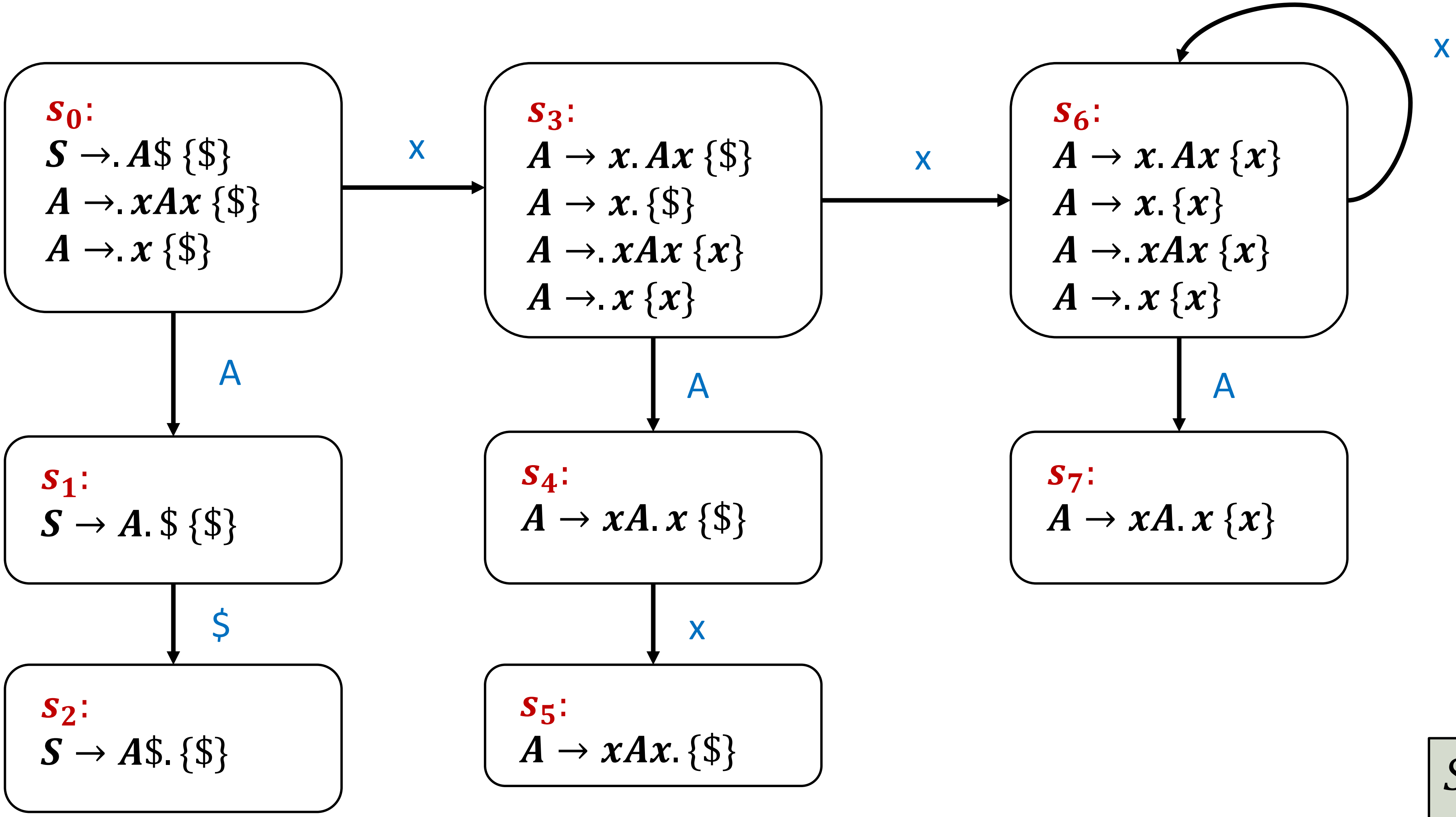

$S \rightarrow A\$$
$A \rightarrow xAx$
$A \rightarrow x$




$S \to A\$$
$A \to xAx$
$A \to x$



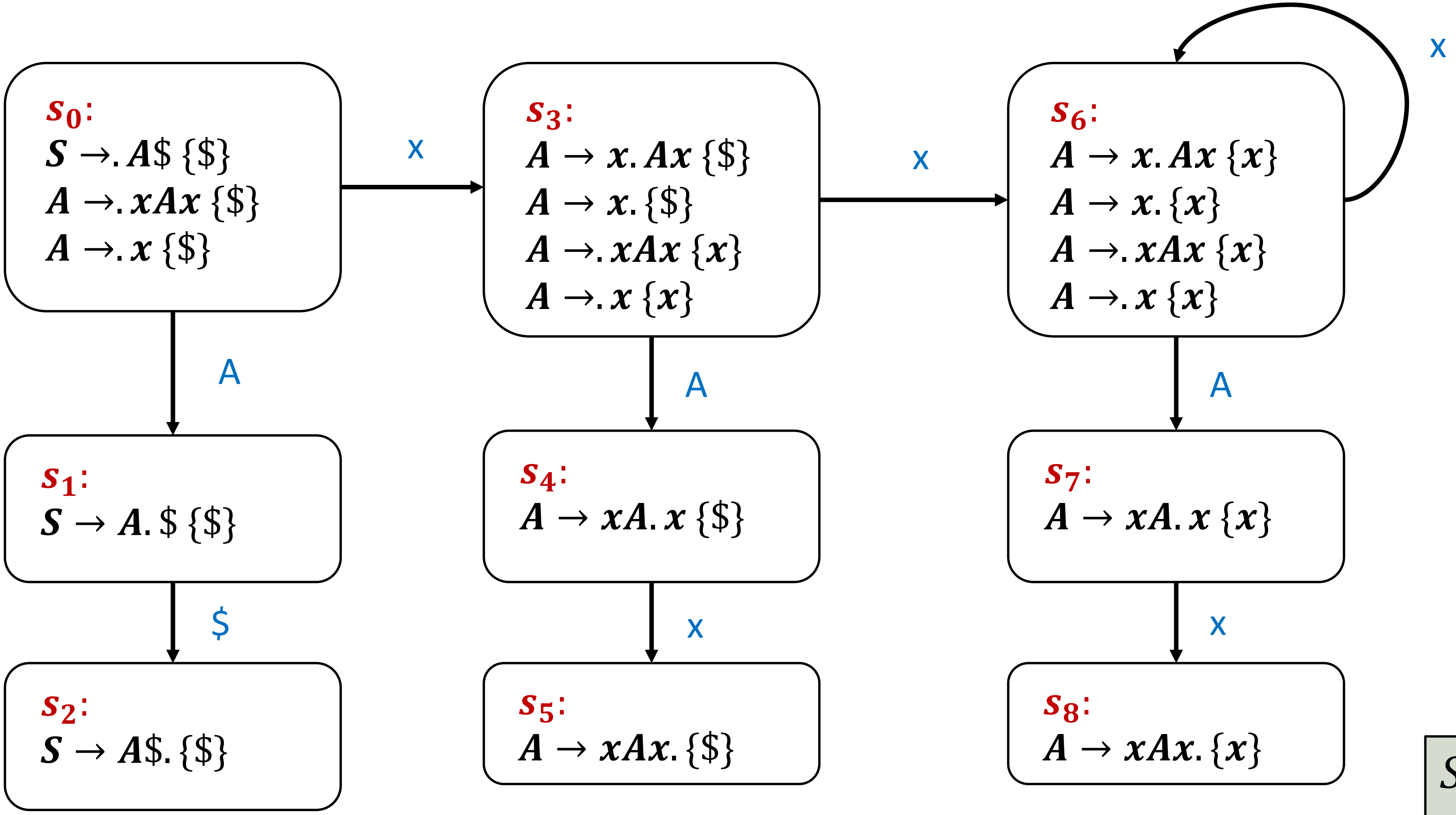

$S \to A\$$
$A \to xAx$
$A \to x$



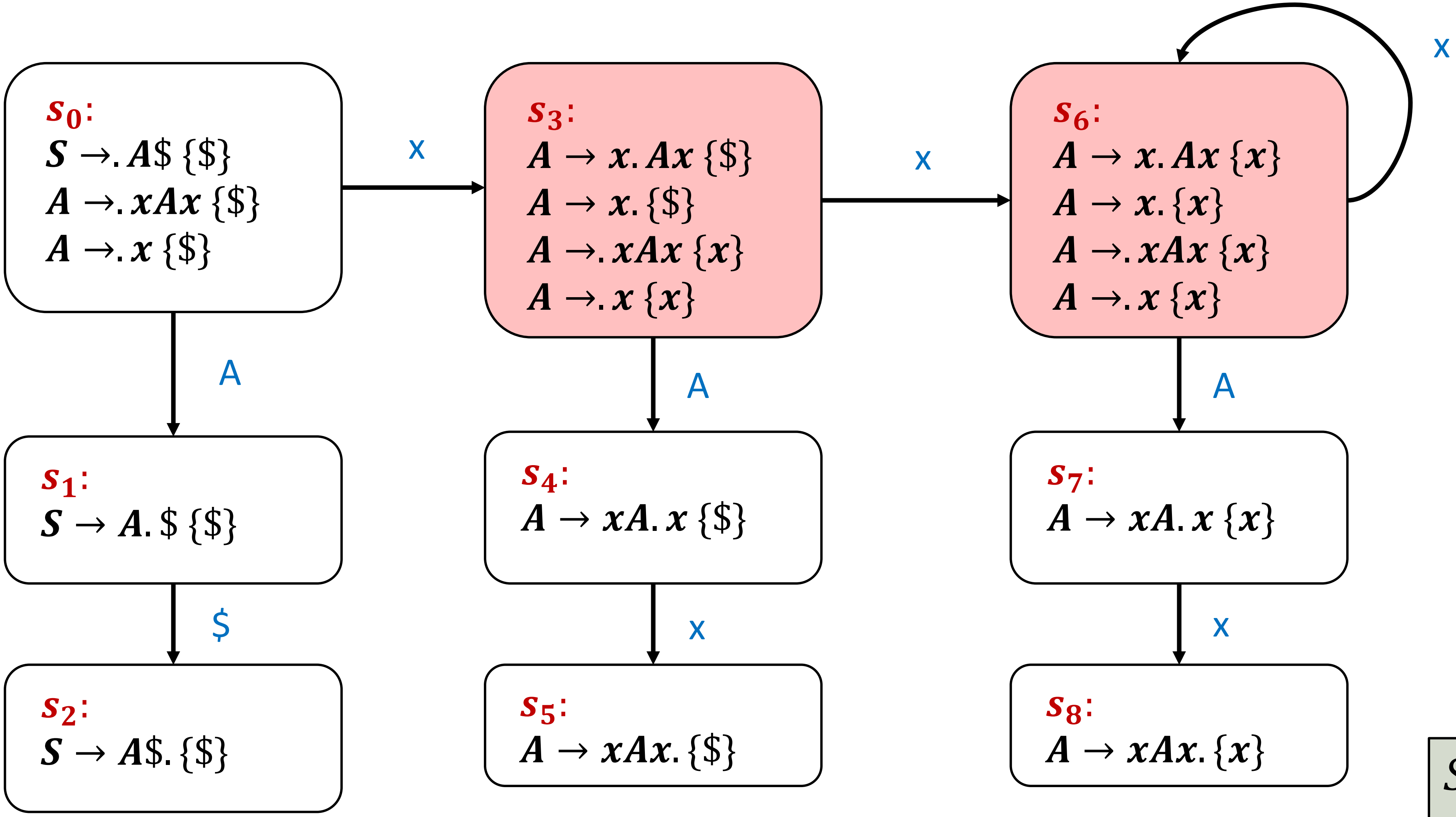

$S \to A\$$

$A \to xAx$

$A \to x$




$S \to A\$$
$A \to xAx$
$A \to x$




$$
\begin{aligned}
S &\to A\$ \\
A &\to xAx \\
A &\to x
\end{aligned}
$$

# 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

A variable *x* **depends** on a variable *y* if *y* is used (directly or indirectly) to compute *x*. For example, *c* depends on *x,y,b,z* but not on *t*.

- Define a static analysis in terms of $(D, V, \sqcup, F, I)$
- Run on the example

```
y := t;
y := z + 5;
if (x > 0) {
  x := y * 2;
} else {
  x := y + b;
}
c := x;
```

# Abstract Domain

We define $(D, V, \sqcup, F, I)$:

# Abstract Domain

We define $(D, V, \sqcup, F, I)$:

- Forward analysis

# Abstract Domain

We define $(D, V, \sqcup, F, I)$:

- Forward analysis
- $V$ contains maps of the form:
  - $Var \mapsto P(Var)$ (e.g., $\{a : \{b, c\}\}$)

# Abstract Domain

We define $(D, V, \sqcup, F, I)$:

- Forward analysis
- $V$ contains maps of the form:
  - $Var \mapsto P(Var)$ (e.g., $\{a : \{b, c\}\}$)
- Join operator:
  - $m_1 \sqcup m_2 \coloneqq \lambda v. m_1[v] \cup m_2[v]$
    - $\{a : \{b\}\} \sqcup \{a : \{c\}, d : \{e\}\} = \{a : \{b, c\}, d : \{e\}\}$

# Abstract Domain

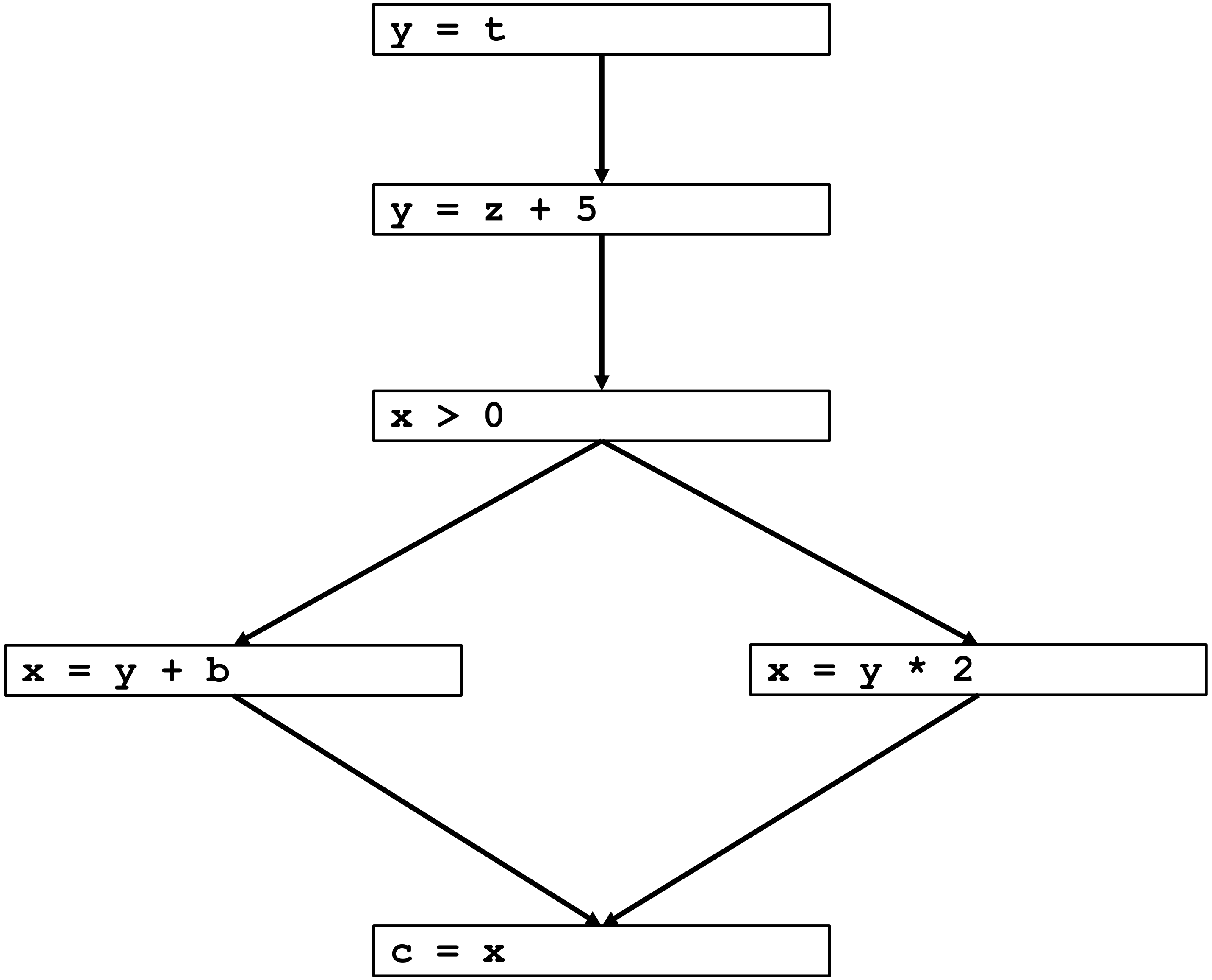
We define $(D, V, \sqcup, F, I)$:

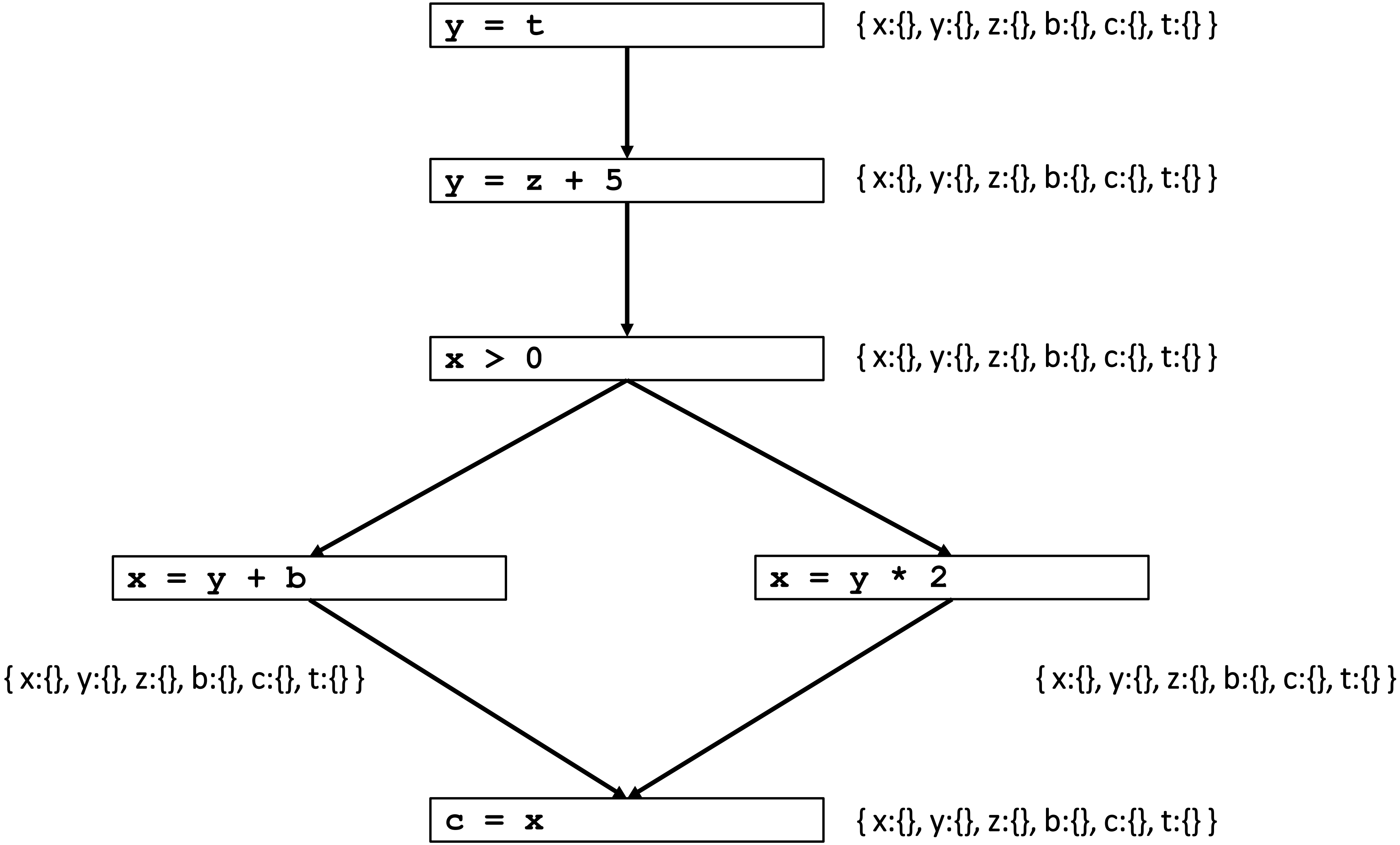
- Forward analysis
- $V$ contains maps of the form:
  - $Var \mapsto P(Var)$ (e.g., $\{a : \{b, c\}\}$)
- Join operator:
  - $m_1 \sqcup m_2 \coloneqq \lambda v. m_1[v] \cup m_2[v]$
    - $\{a : \{b\}\} \sqcup \{a : \{c\}, d : \{e\}\} = \{a : \{b, c\}, d : \{e\}\}$
- On a = b + c:
  - $\{a : s_a, b : s_b, c : s_c, \ldots\} \to \{a : \{b, c\} \cup s_b \cup s_c, b : s_b, c : s_c, \ldots\}$

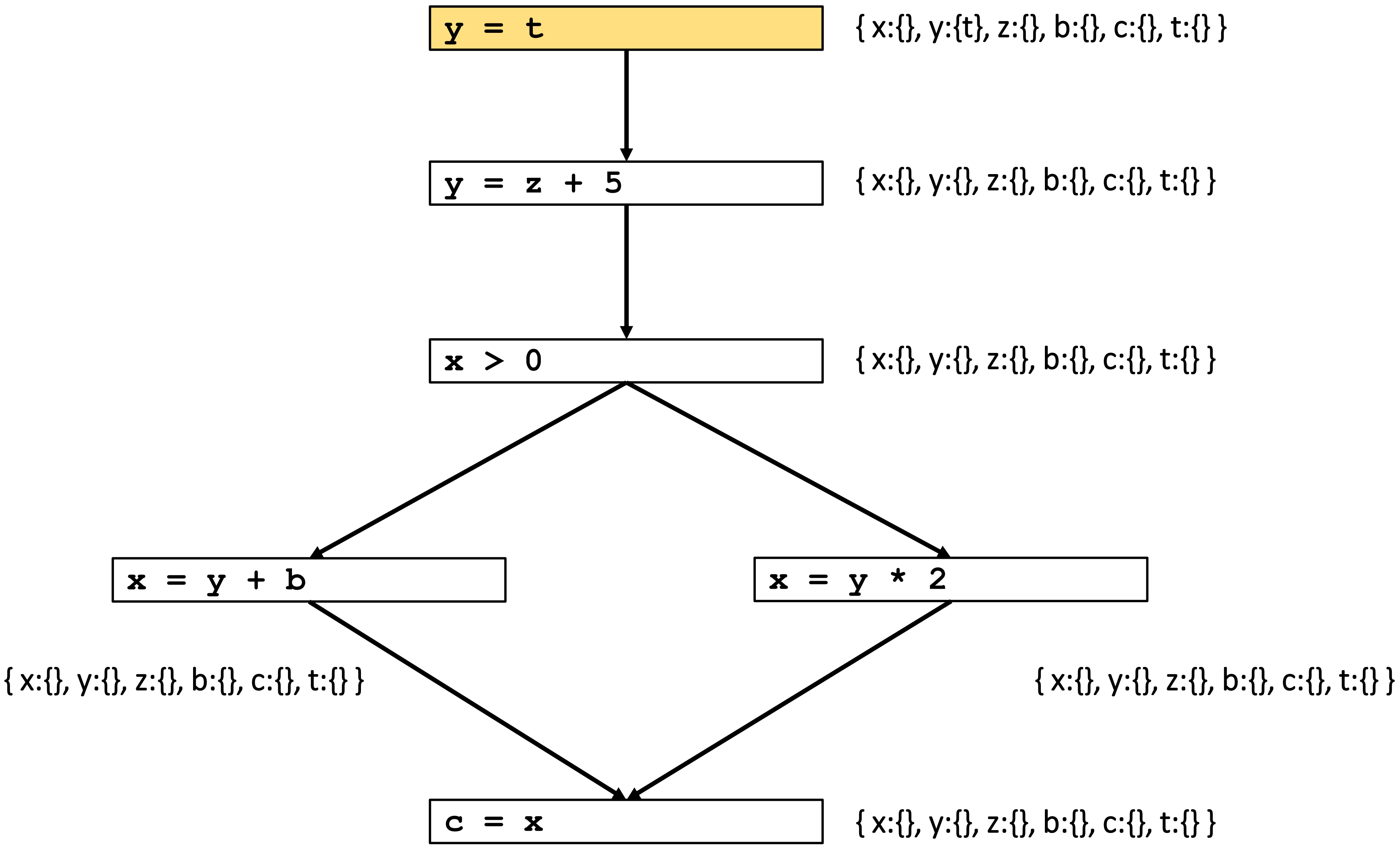
# Abstract Domain

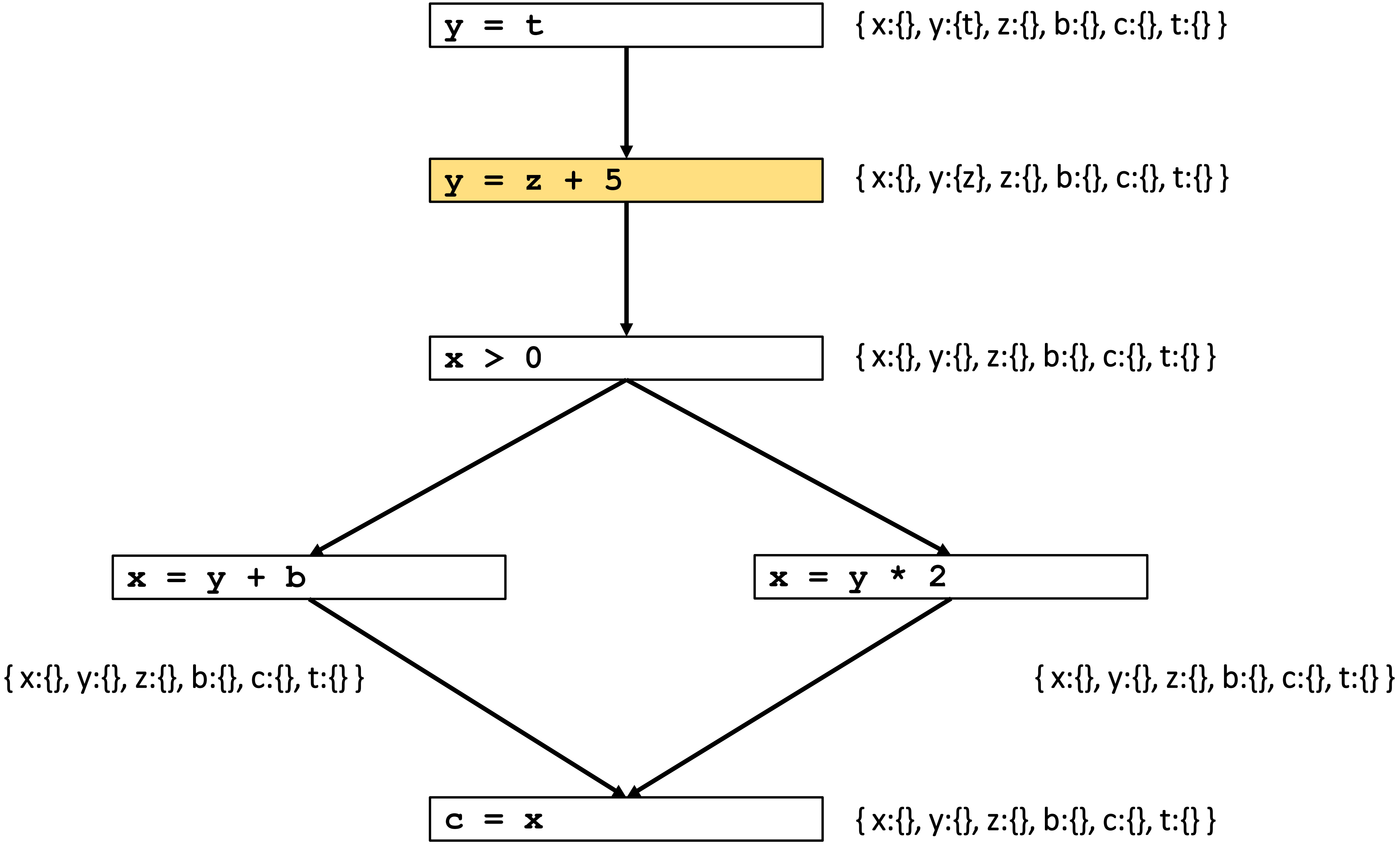
We define $(D, V, \sqcup, F, I)$:

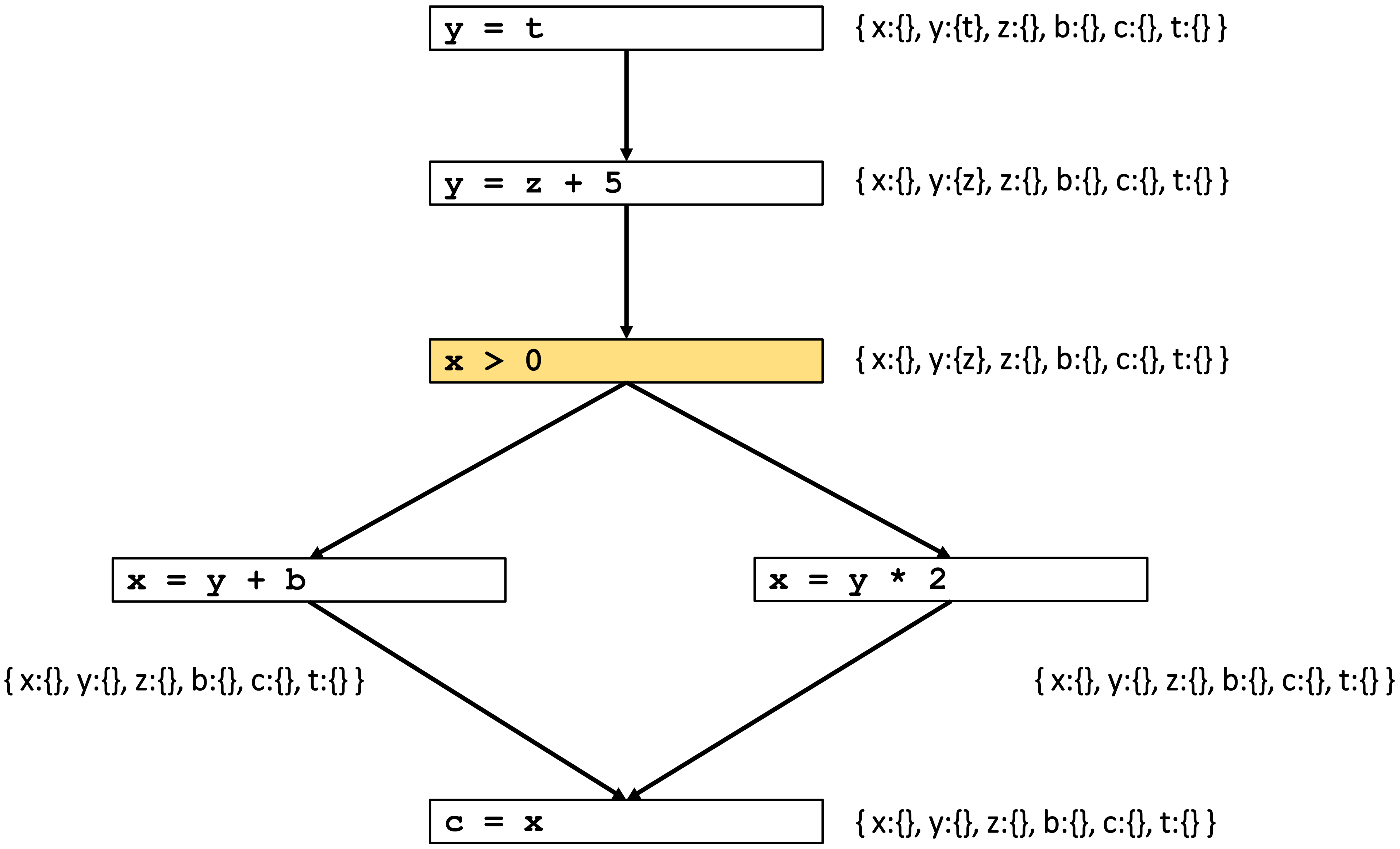
- Forward analysis
- $V$ contains maps of the form:
  - $Var \mapsto P(Var)$ (e.g., $\{a : \{b, c\}\}$)
- Join operator:
  - $m_1 \sqcup m_2 \ \coloneqq \lambda v. m_1[v] \cup m_2[v]$
    - $\{a : \{b\}\} \sqcup \{a : \{c\}, d : \{e\}\} = \{a : \{b, c\}, d : \{e\}\}$
- On a = b + c:
  - $\{a : s_a, b : s_b, c : s_c, \dots\} \rightarrow \{a : \{b, c\} \cup \ s_b \cup s_c, b : s_b, c : s_c, \dots\}$
- Initialize with:
  - $\lambda v. \{\}$

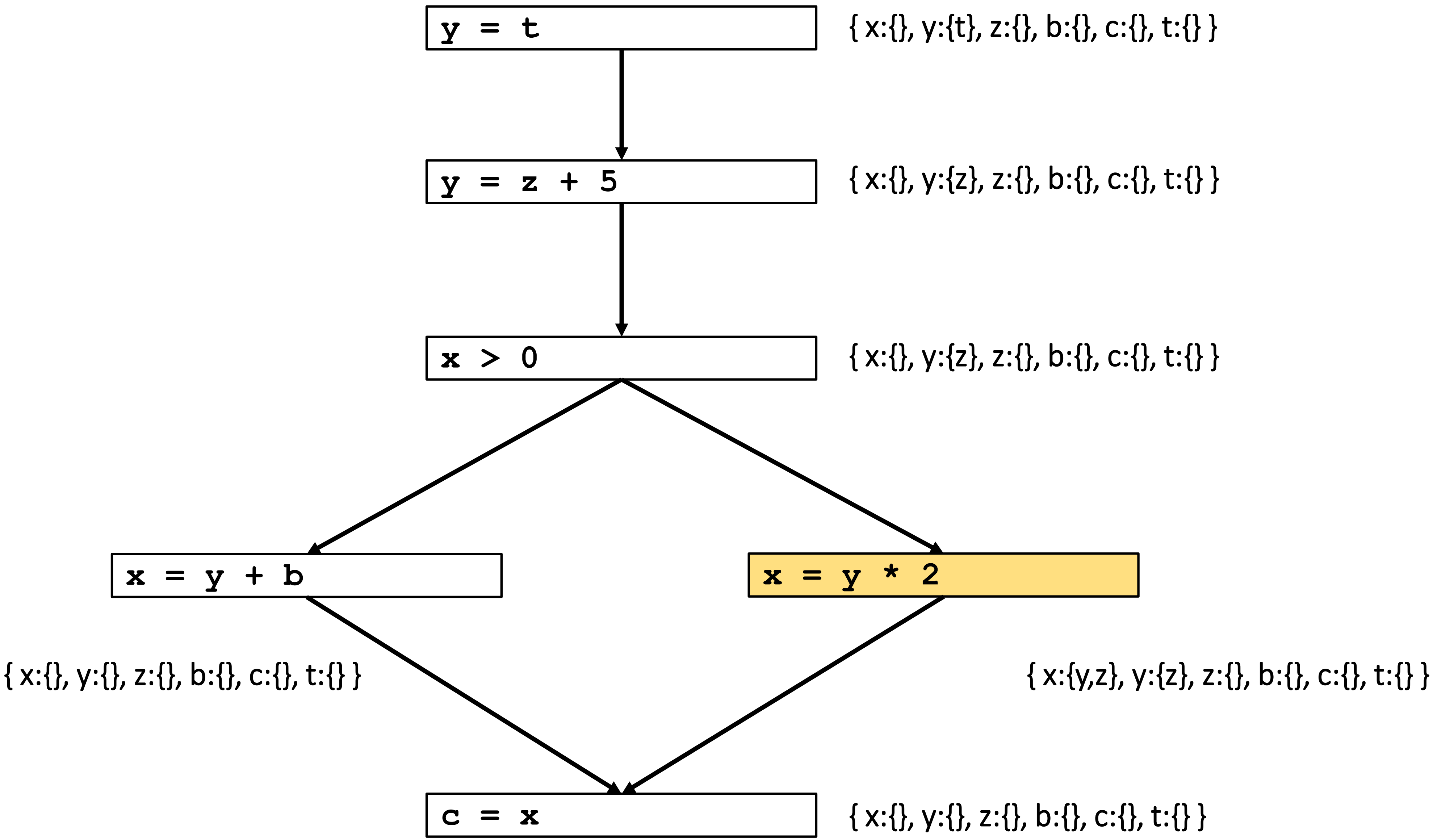

y = t
y = z + 5
x > 0
x = y + b
x = y * 2
c = x


y = t
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
y = z + 5
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
x > 0
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
x = y + b
x = y * 2
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
c = x
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }

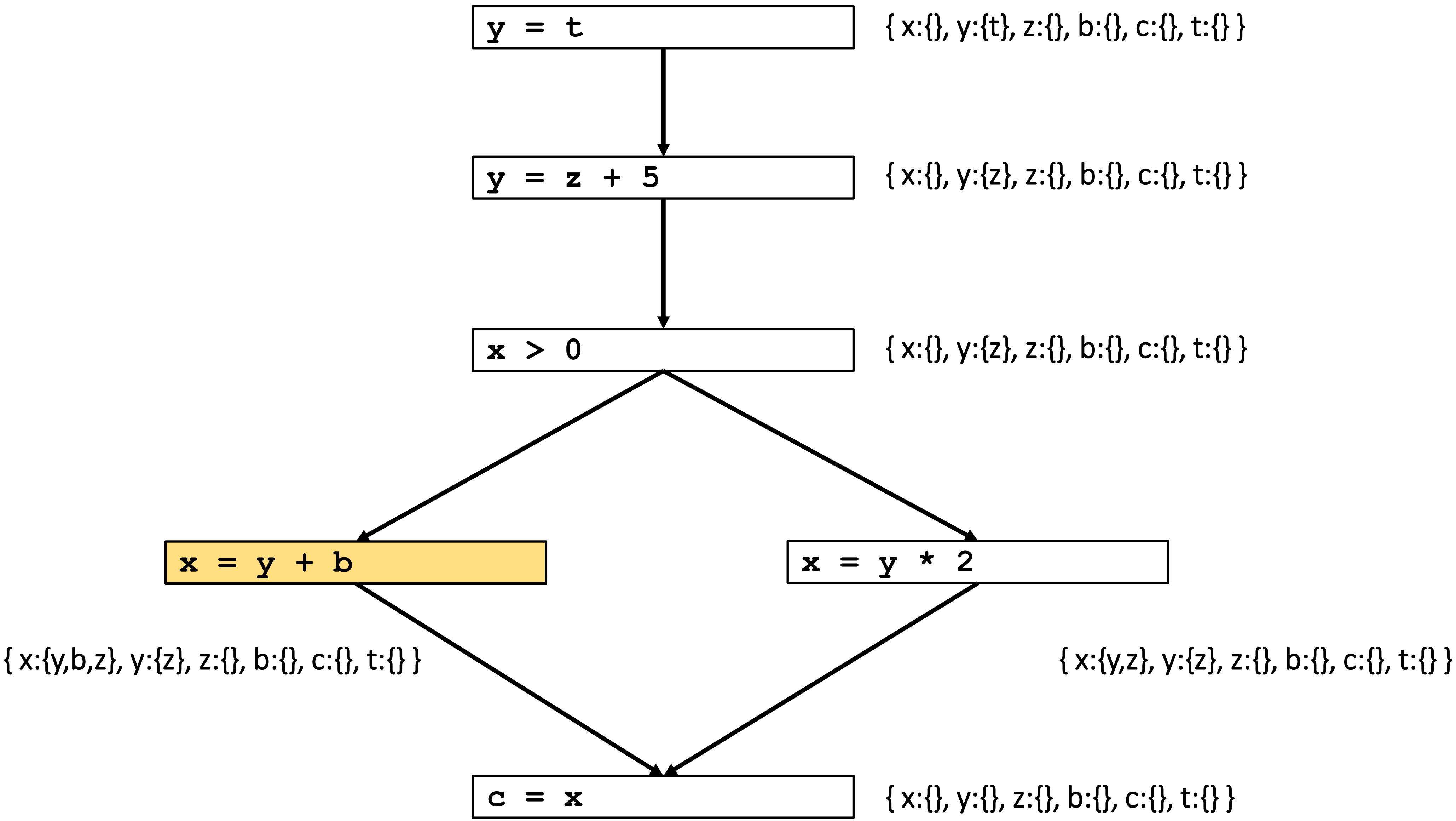

y = t
{ x:{}, y:{t}, z:{}, b:{}, c:{}, t:{} }
y = z + 5
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
x > 0
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
x = y + b
x = y * 2
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
c = x
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }


y = t
{ x:{}, y:{t}, z:{}, b:{}, c:{}, t:{} }
y = z + 5
{ x:{}, y:{z}, z:{}, b:{}, c:{}, t:{} }
x > 0
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
x = y + b
x = y * 2
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
c = x
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }

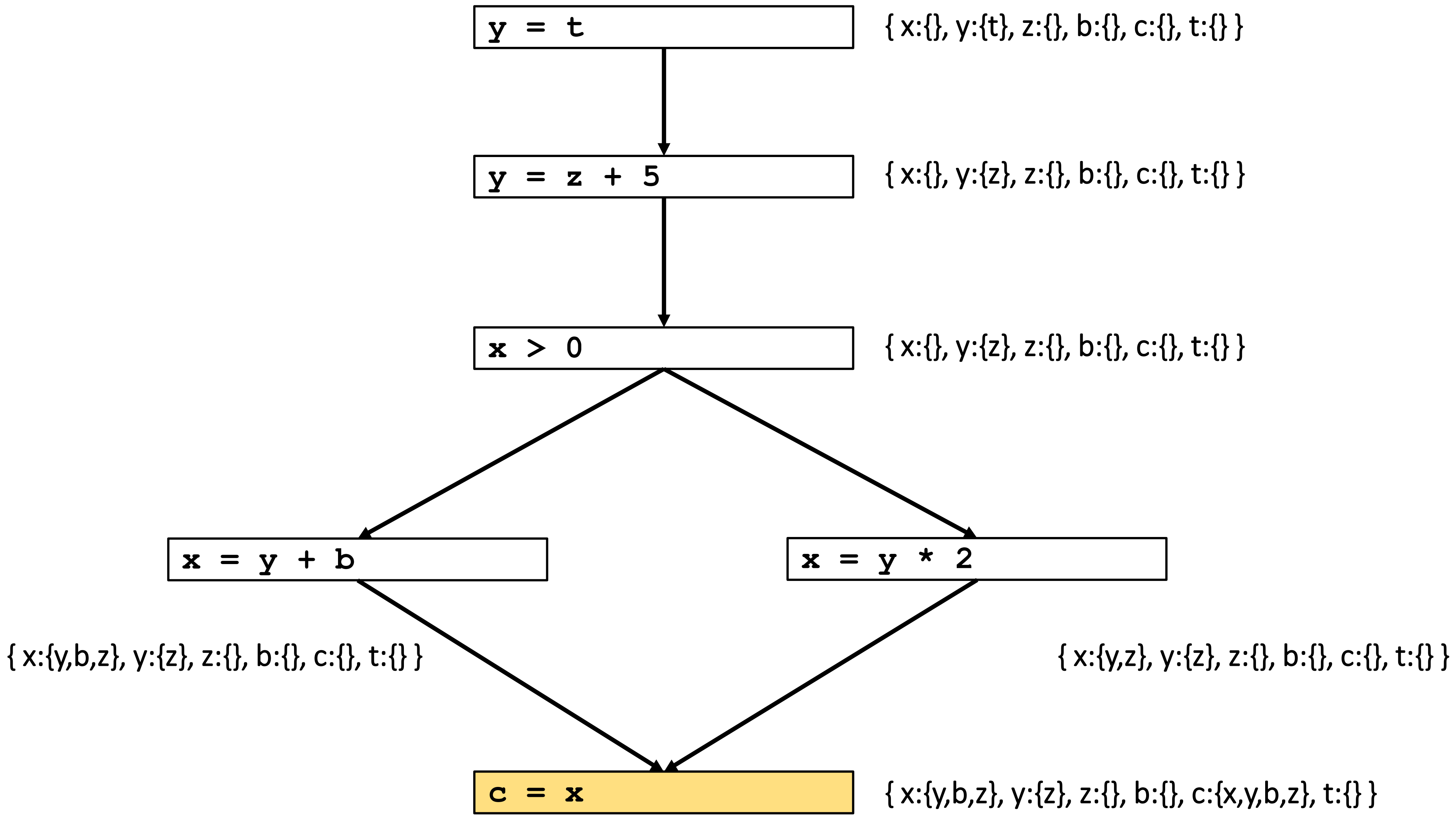

y = t
{ x:{}, y:{t}, z:{}, b:{}, c:{}, t:{} }
y = z + 5
{ x:{}, y:{z}, z:{}, b:{}, c:{}, t:{} }
x > 0
{ x:{}, y:{z}, z:{}, b:{}, c:{}, t:{} }
x = y + b
x = y * 2
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
c = x
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }



y = t
{ x:{}, y:{t}, z:{}, b:{}, c:{}, t:{} }
y = z + 5
{ x:{}, y:{z}, z:{}, b:{}, c:{}, t:{} }
x > 0
{ x:{}, y:{z}, z:{}, b:{}, c:{}, t:{} }
x = y + b
x = y * 2
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }
{ x:{y,z}, y:{z}, z:{}, b:{}, c:{}, t:{} }
c = x
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }



y = t
{ x:{}, y:{t}, z:{}, b:{}, c:{}, t:{} }
y = z + 5
{ x:{}, y:{z}, z:{}, b:{}, c:{}, t:{} }
x > 0
{ x:{}, y:{z}, z:{}, b:{}, c:{}, t:{} }
x = y + b
x = y * 2
{ x:{y,b,z}, y:{z}, z:{}, b:{}, c:{}, t:{} }
{ x:{y,z}, y:{z}, z:{}, b:{}, c:{}, t:{} }
c = x
{ x:{}, y:{}, z:{}, b:{}, c:{}, t:{} }


y = t
{ x:{}, y:{t}, z:{}, b:{}, c:{}, t:{} }
y = z + 5
{ x:{}, y:{z}, z:{}, b:{}, c:{}, t:{} }
x > 0
{ x:{}, y:{z}, z:{}, b:{}, c:{}, t:{} }
x = y + b
x = y * 2
{ x:{y,b,z}, y:{z}, z:{}, b:{}, c:{}, t:{} }
{ x:{y,z}, y:{z}, z:{}, b:{}, c:{}, t:{} }
c = x
{ x:{y,b,z}, y:{z}, z:{}, b:{}, c:{x,y,b,z}, t:{} }

# Question

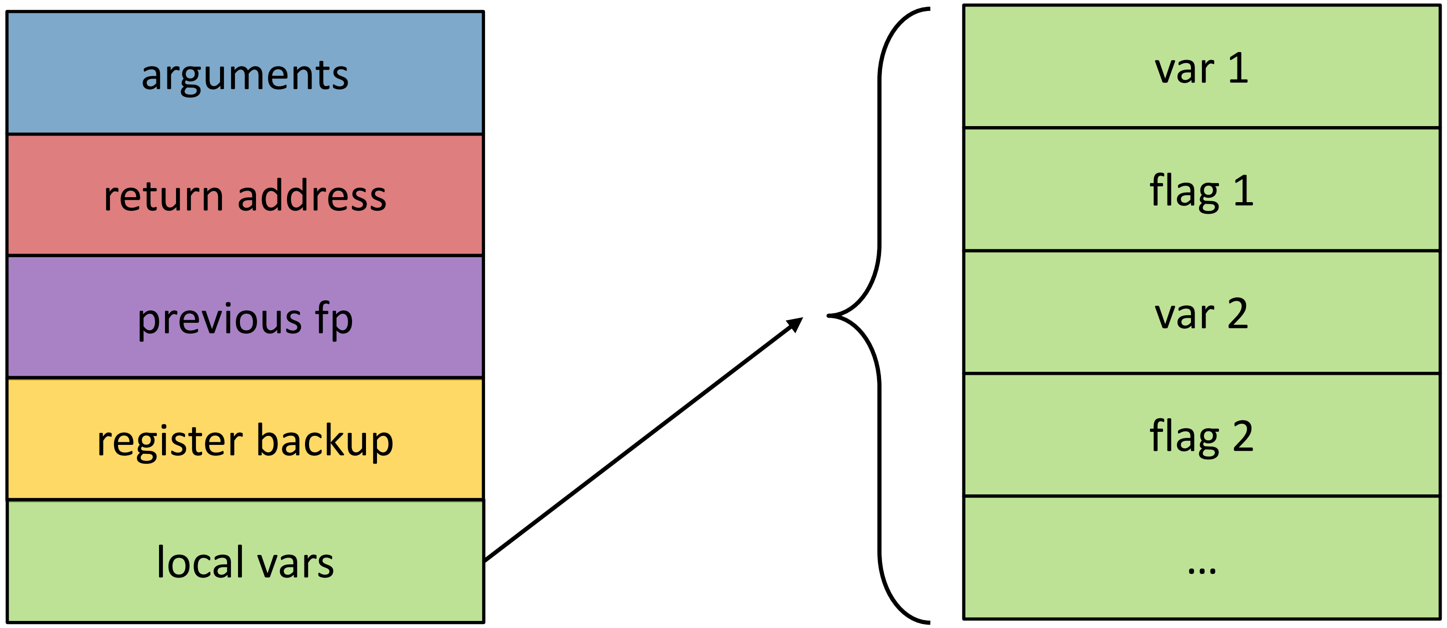
Add support for detecting accesses to uninitialized local variables.
Describe the required changes in the code generation step.

For example:

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

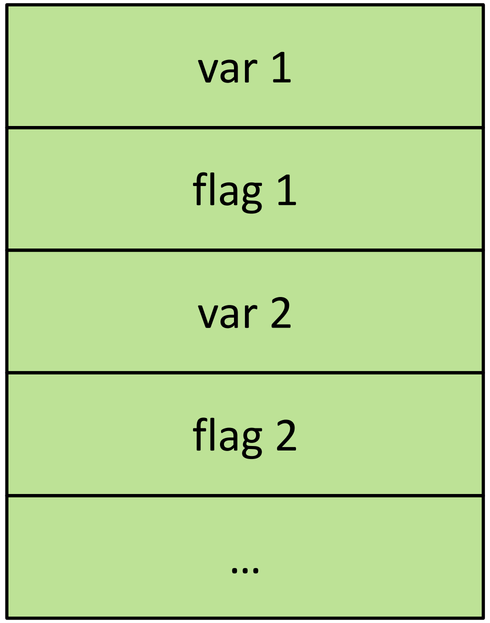
# Solution

High level idea:



# Solution

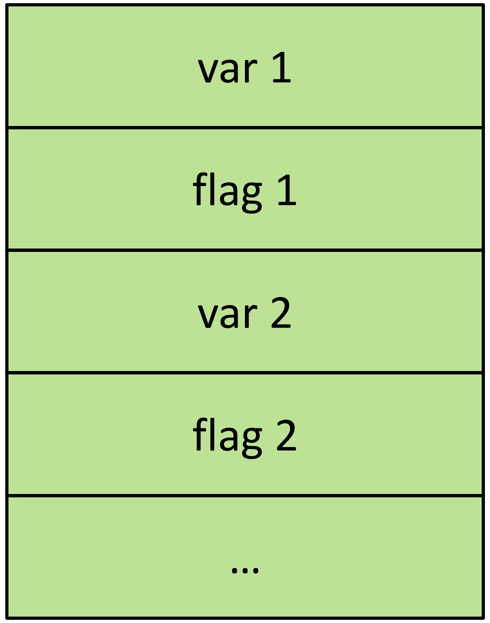
- Initialize each variable flag to zero on function entry
- When writing to a local variable, set its flag to 1
- When reading a local variable, check if its flag is 1



# 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)
...
```

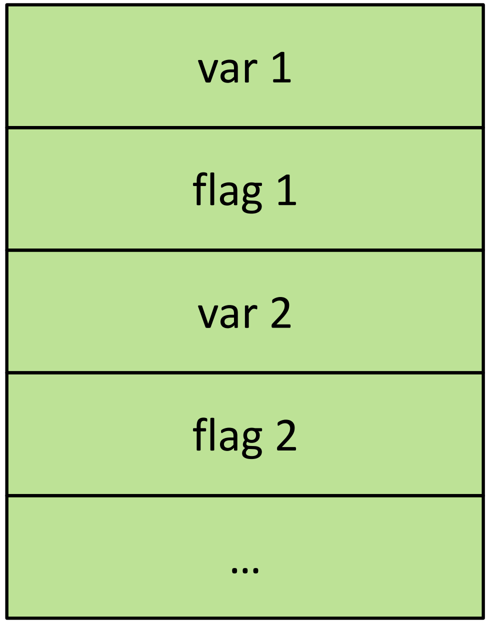


# Solution

- 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)
```

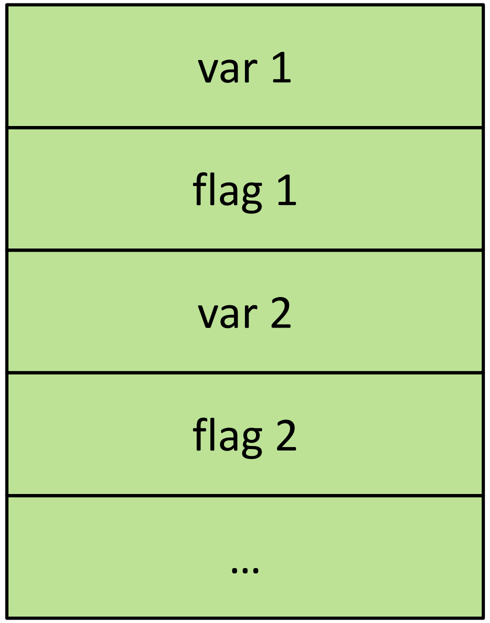


# Solution

- 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)
```



# 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
```

```
t0 = 0
t1 = 1
t2 = 2
t3 = mul t2, t1
t2 = t3
t1 = add t1, t0
```

| IN | | OUT |
|---|---|---|
| { } | `t0 = 0` | { } |
| { } | `t1 = 1` | { } |
| { } | `t2 = 2` | { } |
| { } | `t3 = mul t2, t1` | { } |
| { } | `t2 = t3` | { } |
| { } | `t1 = add t1, t0` | { } |

# initialization

| IN | | OUT |
|---|---|---|
| { } | `t0 = 0` | { } |
| { } | `t1 = 1` | { } |
| { } | `t2 = 2` | { } |
| { } | `t3 = mul t2, t1` | { } |
| { } | `t2 = t3` | { } |
| { t0, t1 } | `t1 = add t1, t0` | { } |

# first iteration

| IN | | OUT |
| --- | --- | --- |
| { } | `t0 = 0` | { } |
| { } | `t1 = 1` | { } |
| { } | `t2 = 2` | { } |
| { } | `t3 = mul t2, t1` | { } |
| { t0, t1, t3 } | `t2 = t3` | { t0, t1 } |
| { t0, t1 } | `t1 = add t1, t0` | { } |

# first iteration

| IN | | OUT |
|---|---|---|
| { } | `t0 = 0` | { } |
| { } | `t1 = 1` | { } |
| { } | `t2 = 2` | { } |
| { t2, t1, t0 } | `t3 = mul t2, t1` | { t0, t1, t3 } |
| { t0, t1, t3 } | `t2 = t3` | { t0, t1 } |
| { t0, t1 } | `t1 = add t1, t0` | { } |

# first iteration

| IN | | OUT |
|---|---|---|
| { } | `t0 = 0` | { } |
| { } | `t1 = 1` | { } |
| { t1, t0 } | `t2 = 2` | { t2, t1, t0 } |
| { t2, t1, t0 } | `t3 = mul t2, t1` | { t0, t1, t3 } |
| { t0, t1, t3 } | `t2 = t3` | { t0, t1 } |
| { t0, t1 } | `t1 = add t1, t0` | { } |

# first iteration

| IN | | OUT |
| --- | --- | --- |
| { } | `t0 = 0` | { } |
| { t0 } | `t1 = 1` | { t1, t0 } |
| { t1, t0 } | `t2 = 2` | { t2, t1, t0 } |
| { t2, t1, t0 } | `t3 = mul t2, t1` | { t0, t1, t3 } |
| { t0, t1, t3 } | `t2 = t3` | { t0, t1 } |
| { t0, t1 } | `t1 = add t1, t0` | { } |

# first iteration

| IN | | OUT |
|---|---|---|
| { } | `t0 = 0` | { t0 } |
| { t0 } | `t1 = 1` | { t1, t0 } |
| { t1, t0 } | `t2 = 2` | { t2, t1, t0 } |
| { t2, t1, t0 } | `t3 = mul t2, t1` | { t0, t1, t3 } |
| { t0, t1, t3 } | `t2 = t3` | { t0, t1 } |
| { t0, t1 } | `t1 = add t1, t0` | { } |

# first iteration

| IN | | OUT |
| --- | --- | --- |
| { } | `t0 = 0` | { t0 } |
| { t0 } | `t1 = 1` | { t1, t0 } |
| { t1, t0 } | `t2 = 2` | { t2, t1, t0 } |
| { t2, t1, t0 } | `t3 = mul t2, t1` | { t0, t1, t3 } |
| { t0, t1, t3 } | `t2 = t3` | { t0, t1 } |
| { t0, t1 } | `t1 = add t1, t0` | { } |

# second iteration…

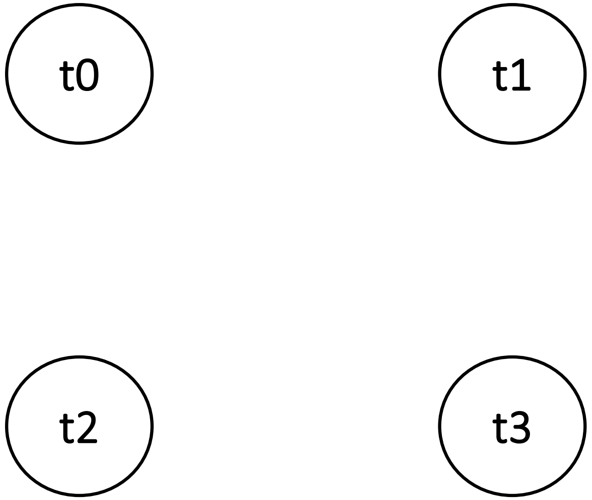
# Interference Graph

{ t0 }

{ t0, t1 }

{ t0, t2, t1 }

{ t0, t3 }




R1 R2 R3

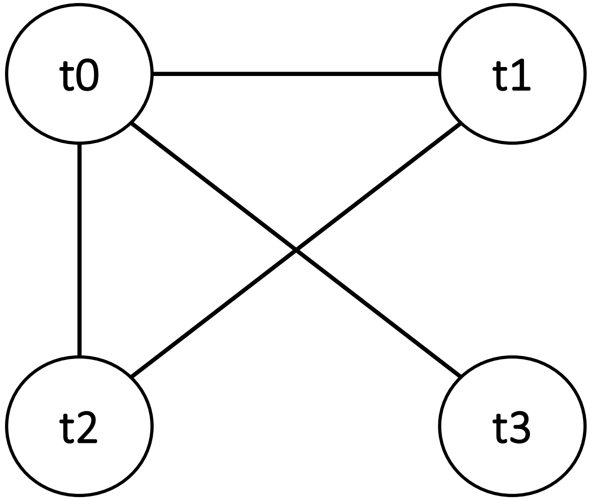
# Interference Graph

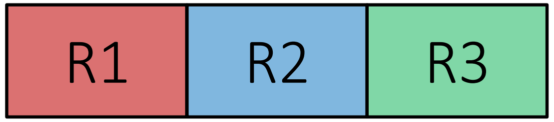
{ t0 }

{ t0, t1 }

{ t0, t2, t1 }

{ t0, t3 }



# Interference Graph

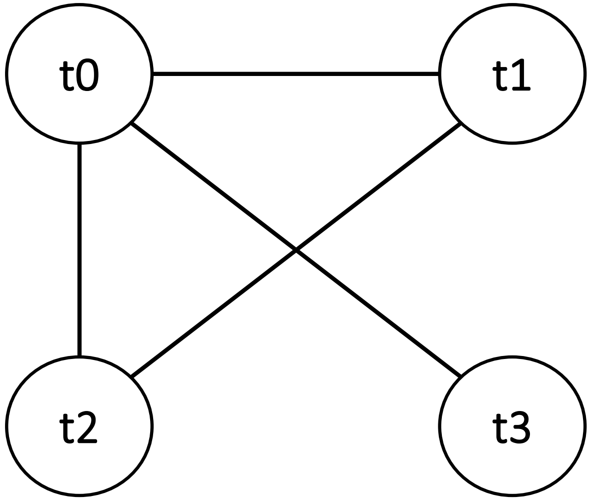
{ t0 }

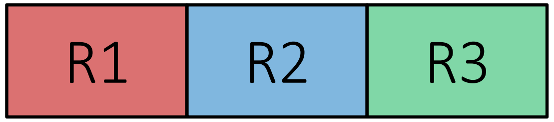
{ t0, t1 }

{ t0, t2, t1 }

{ t0, t3 }

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

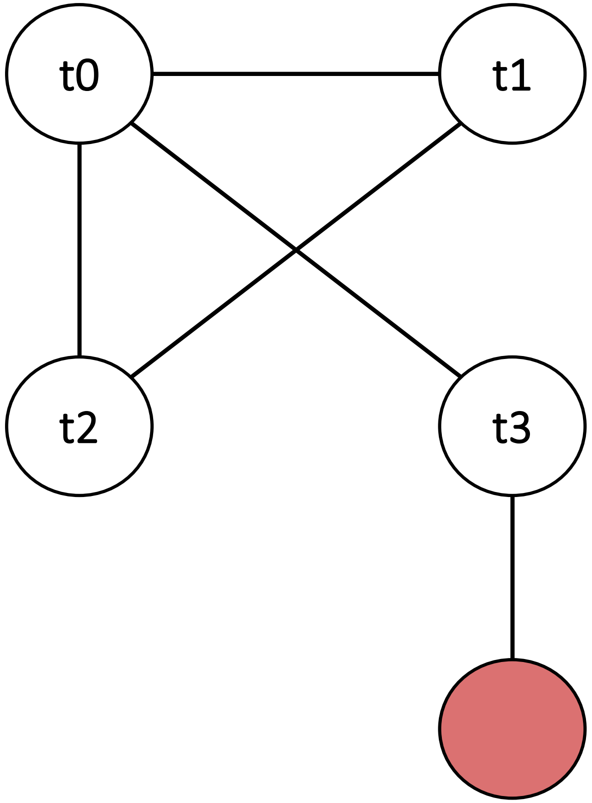


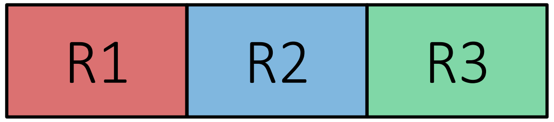



# Interference Graph

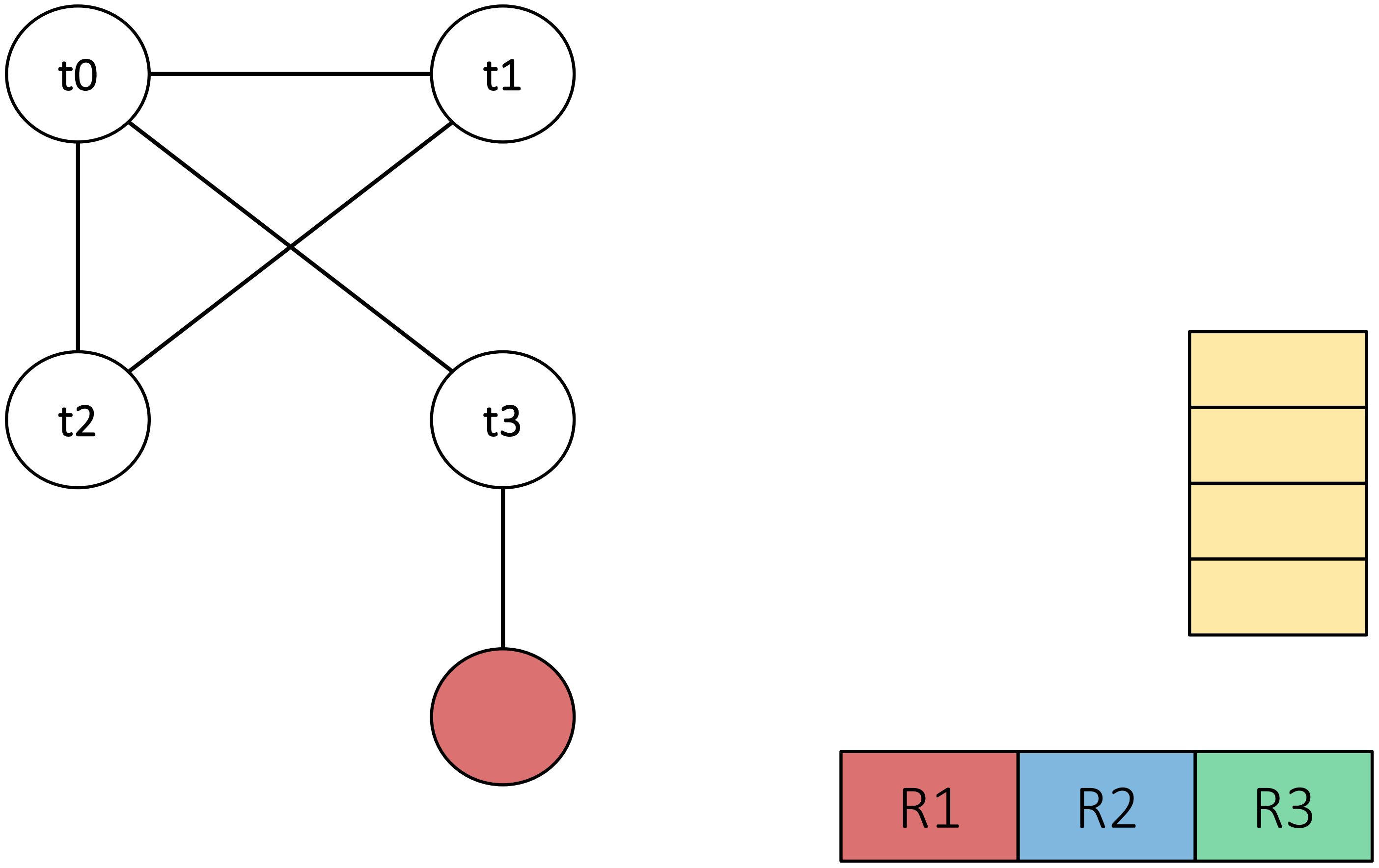
{ t0 }
{ t0, t1 }
{ t0, t2, t1 }
{ t0, t3 }

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

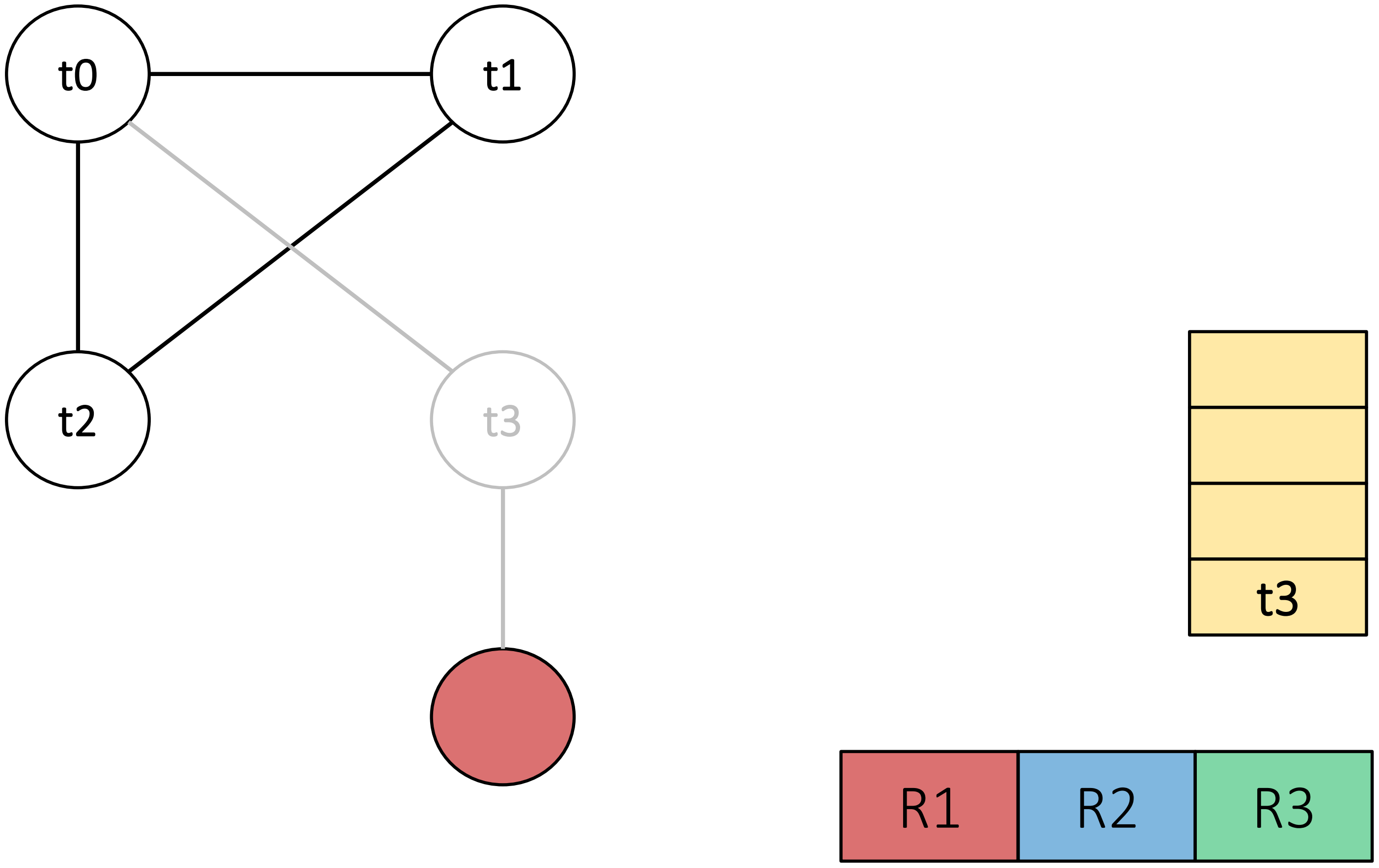


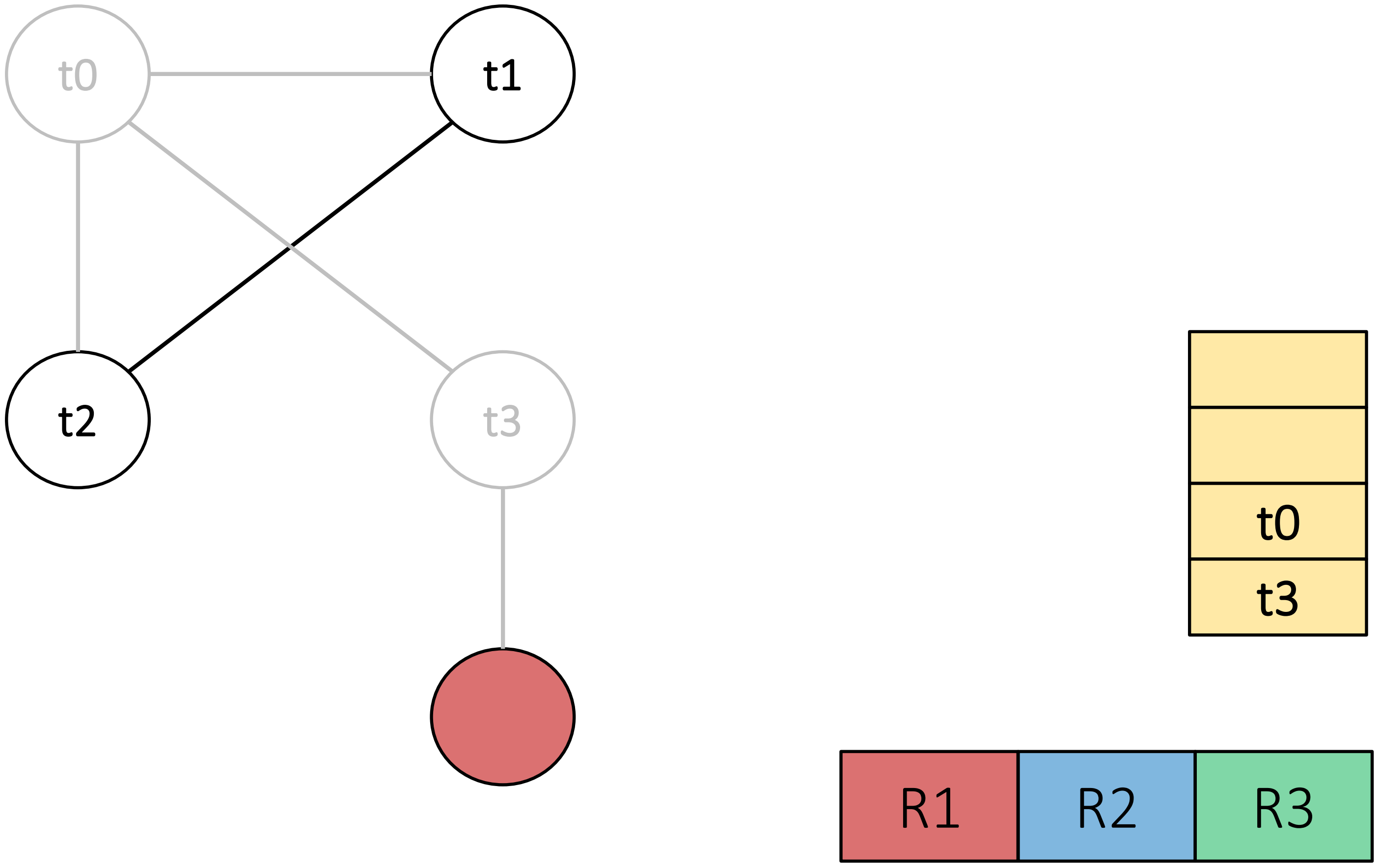
# Graph Coloring



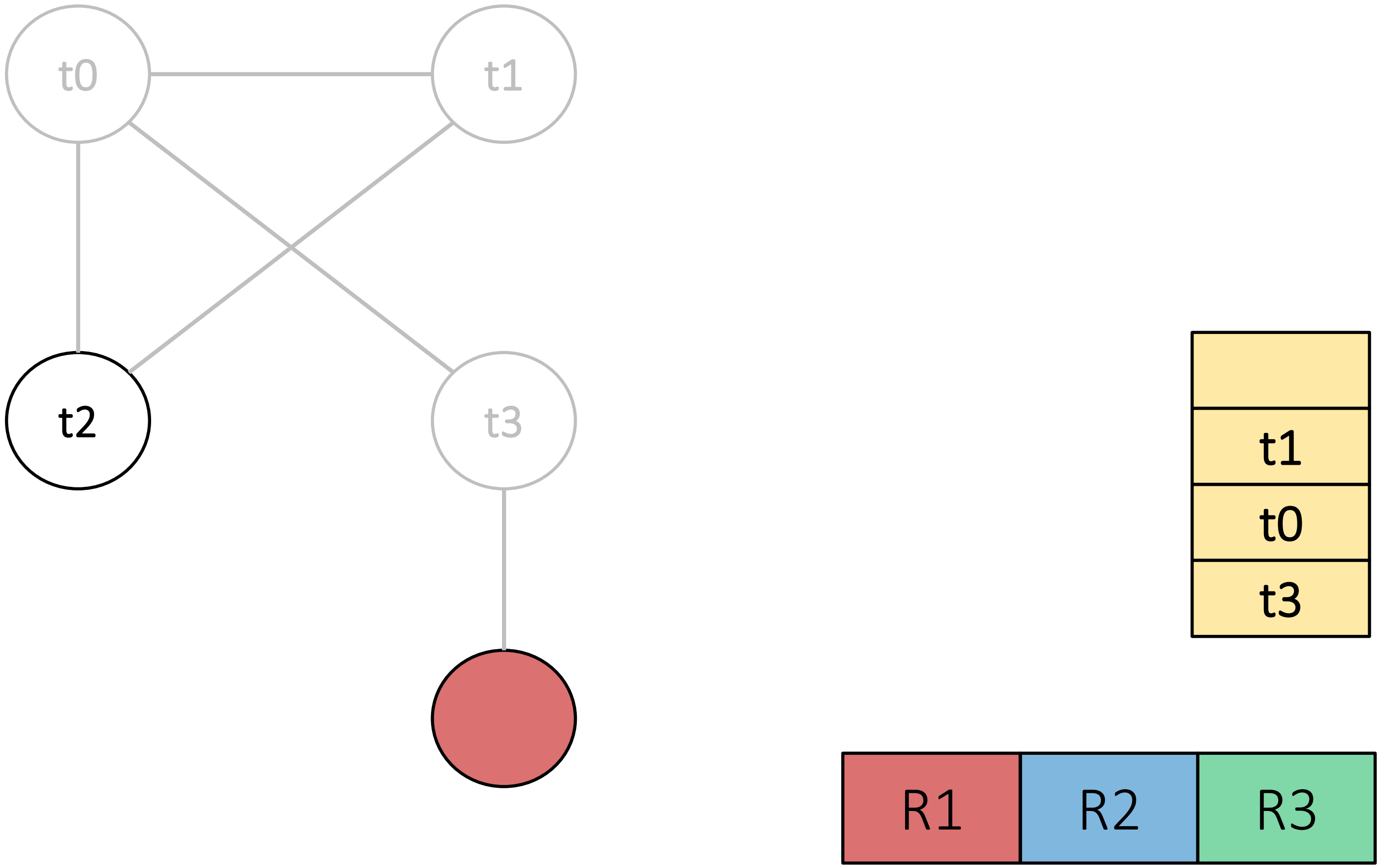
# Graph Coloring



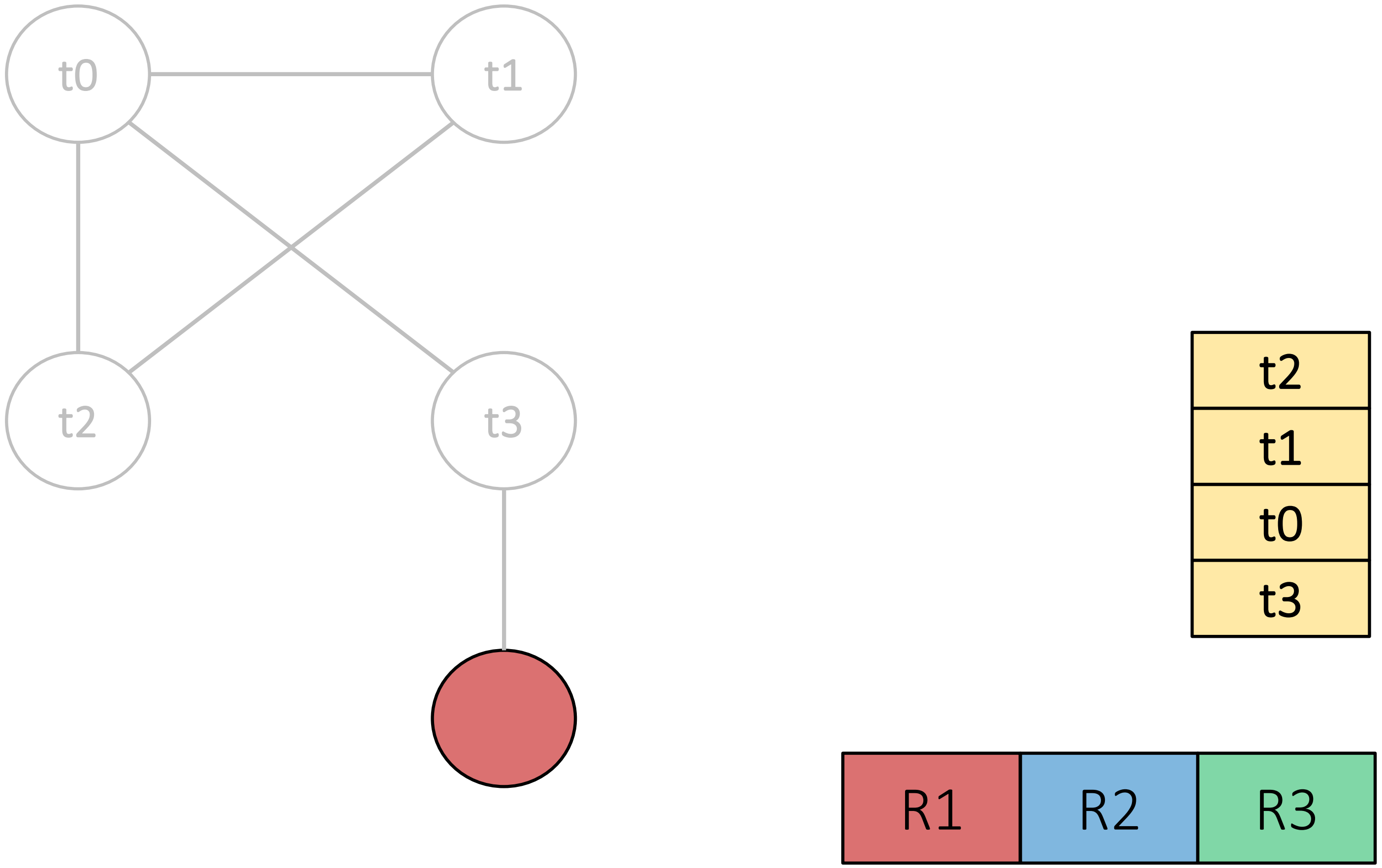
# Graph Coloring



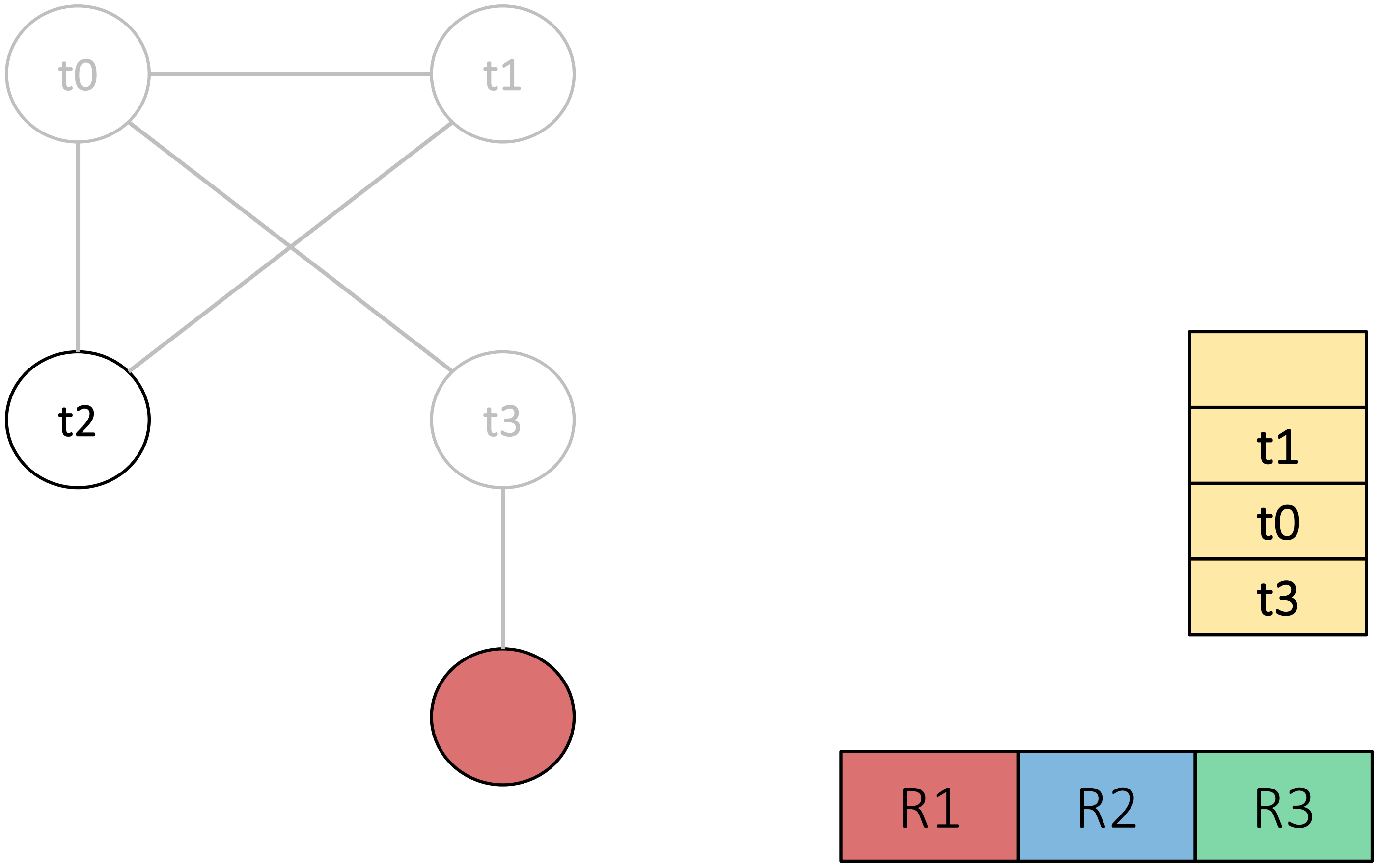
# Graph Coloring



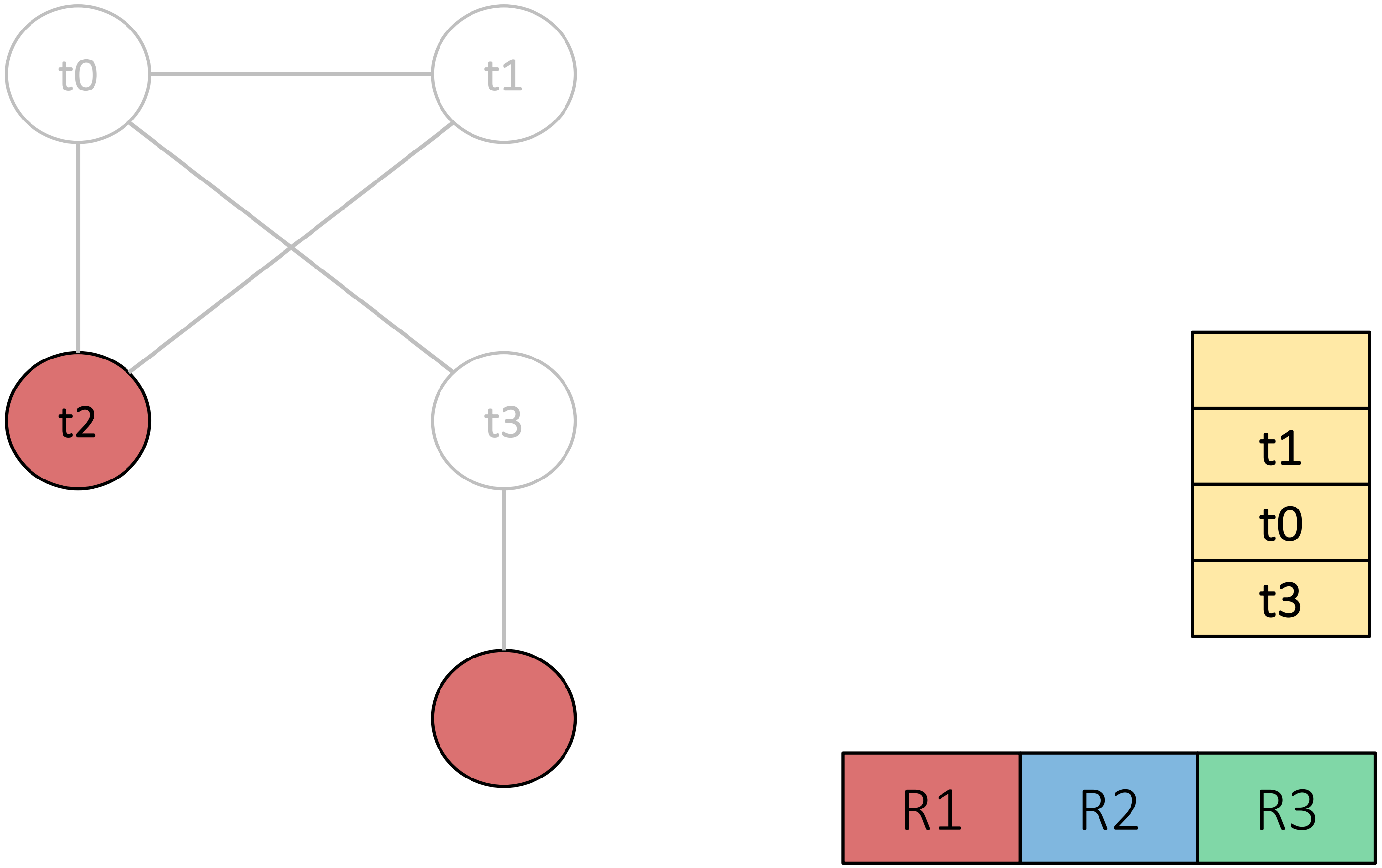
# Graph Coloring



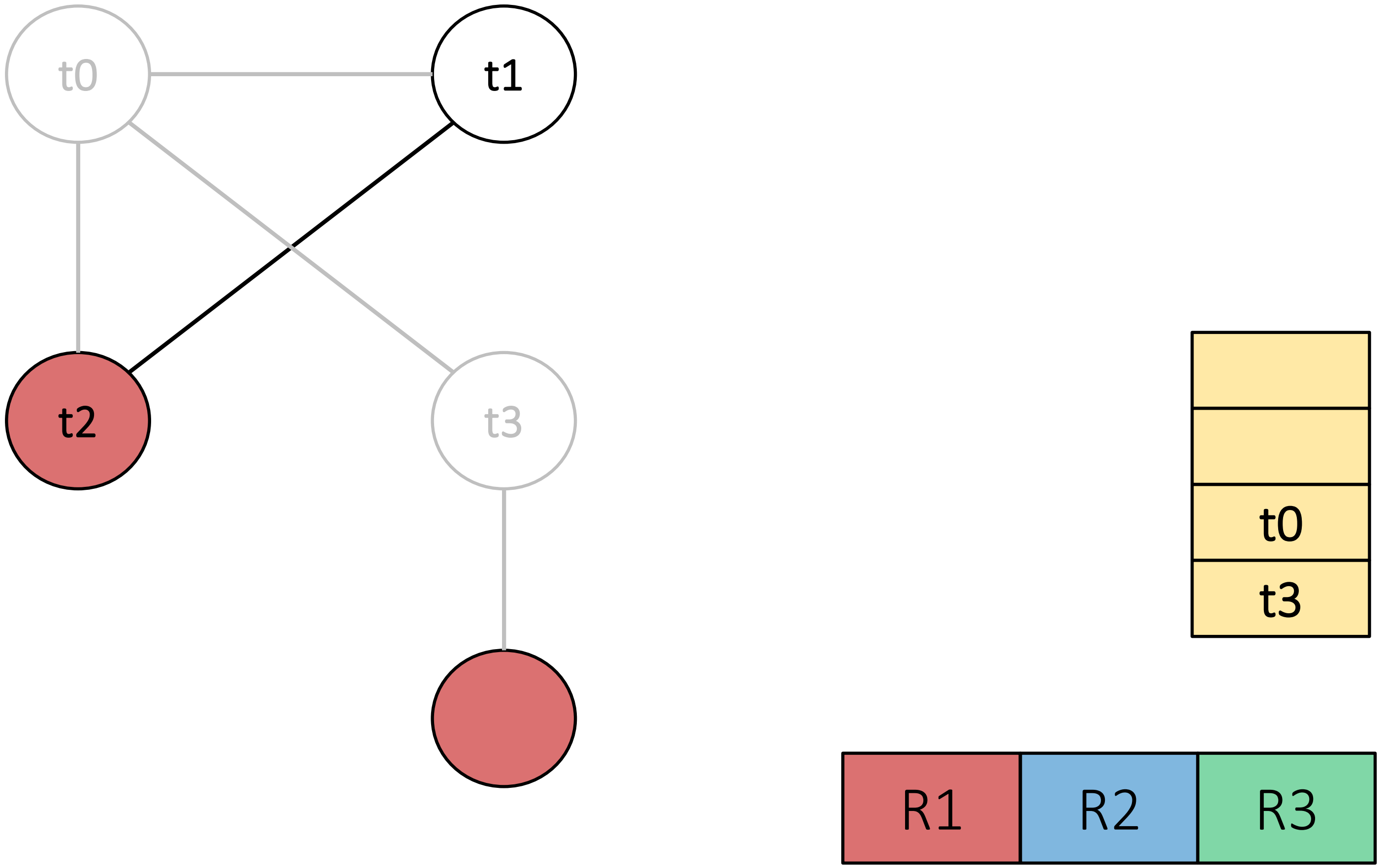
# Graph Coloring



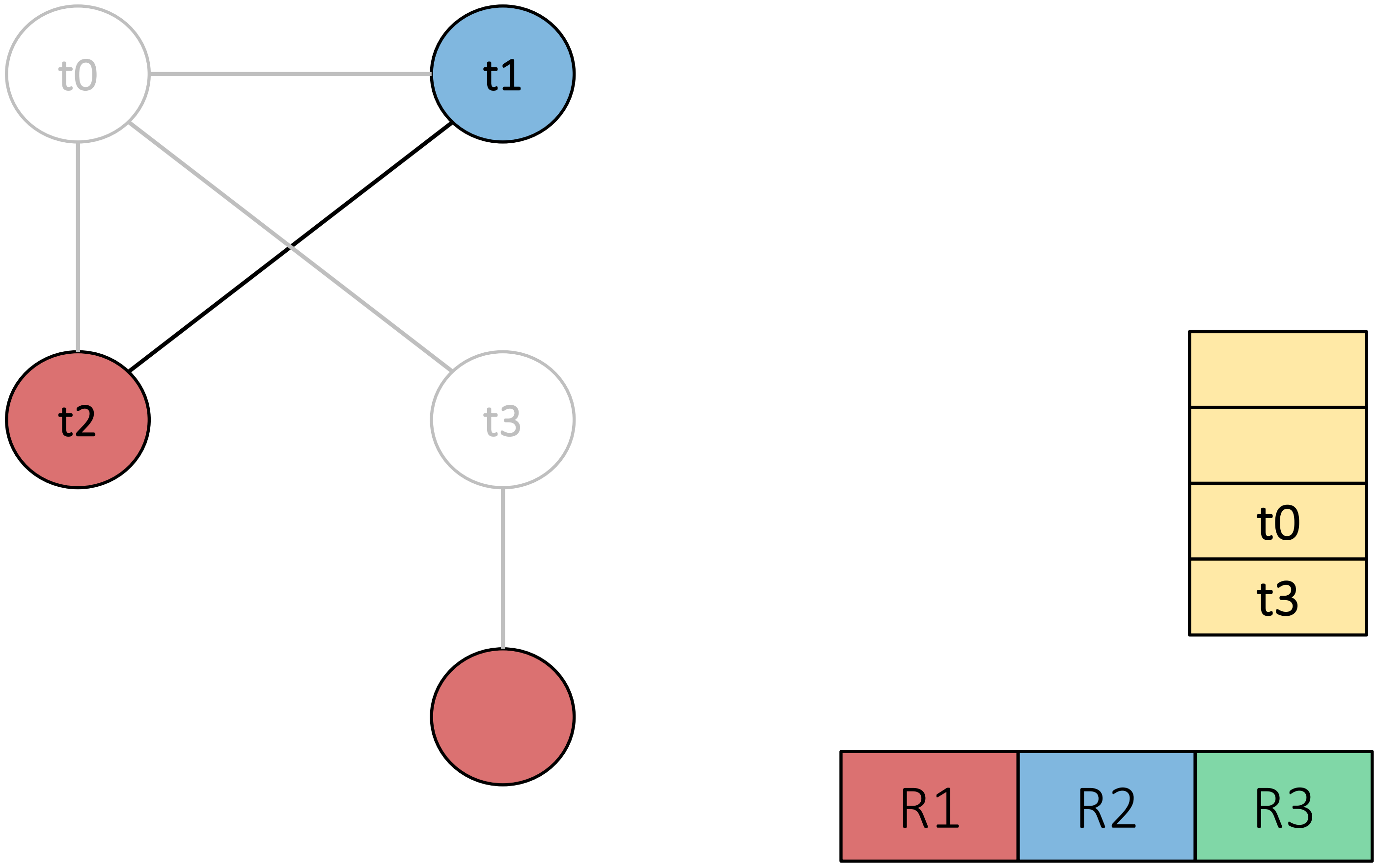
# Graph Coloring



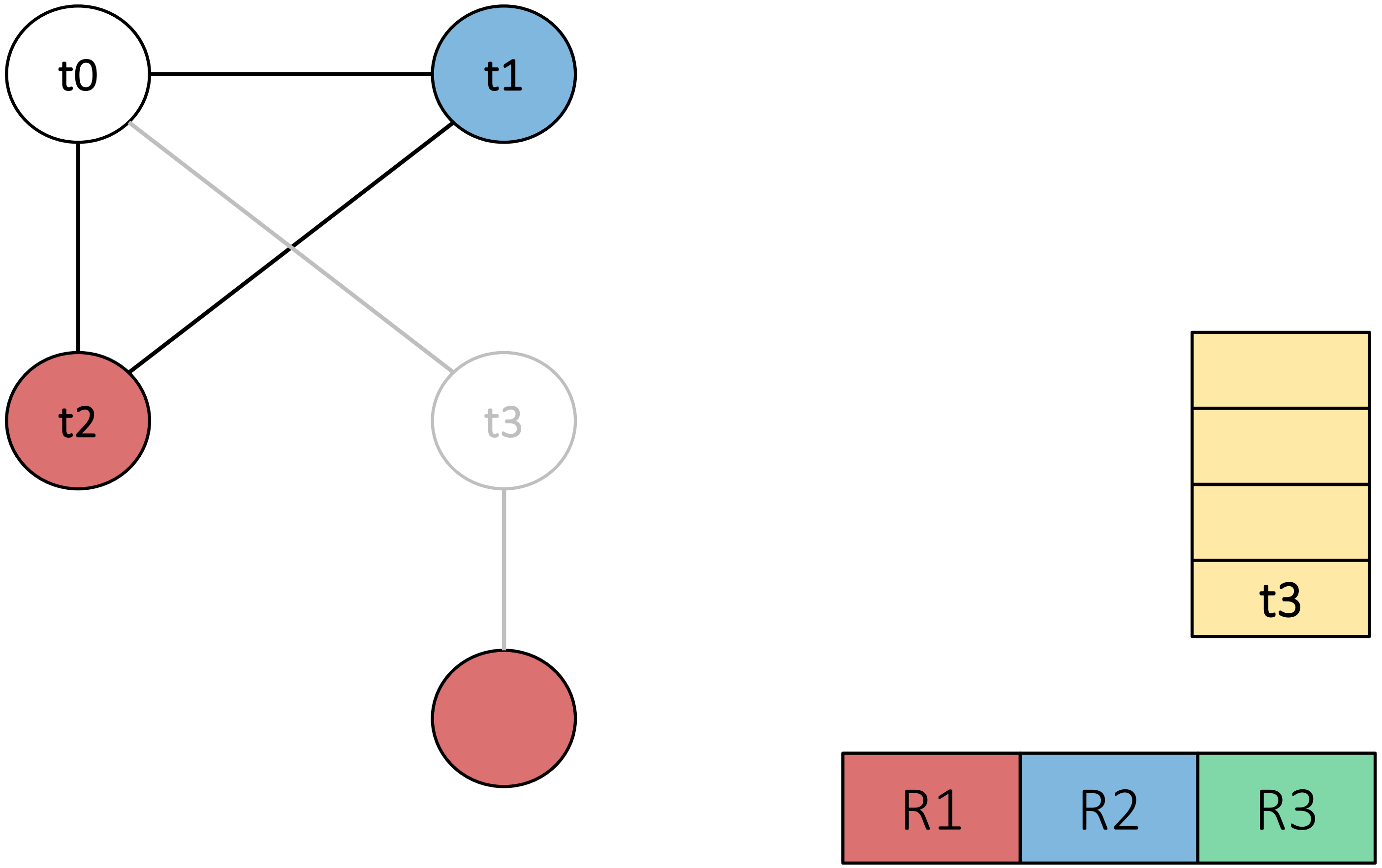
# Graph Coloring



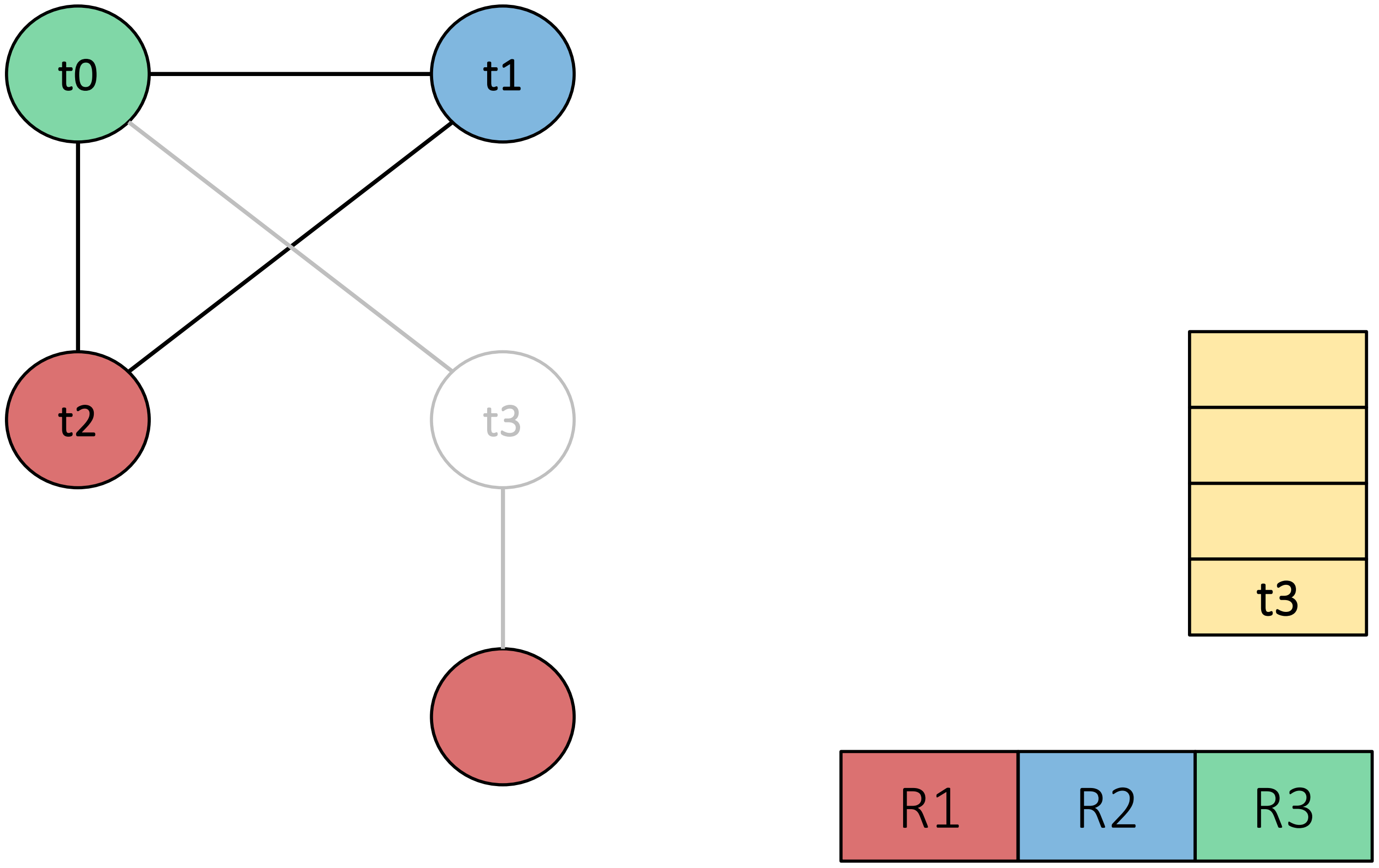
# Graph Coloring



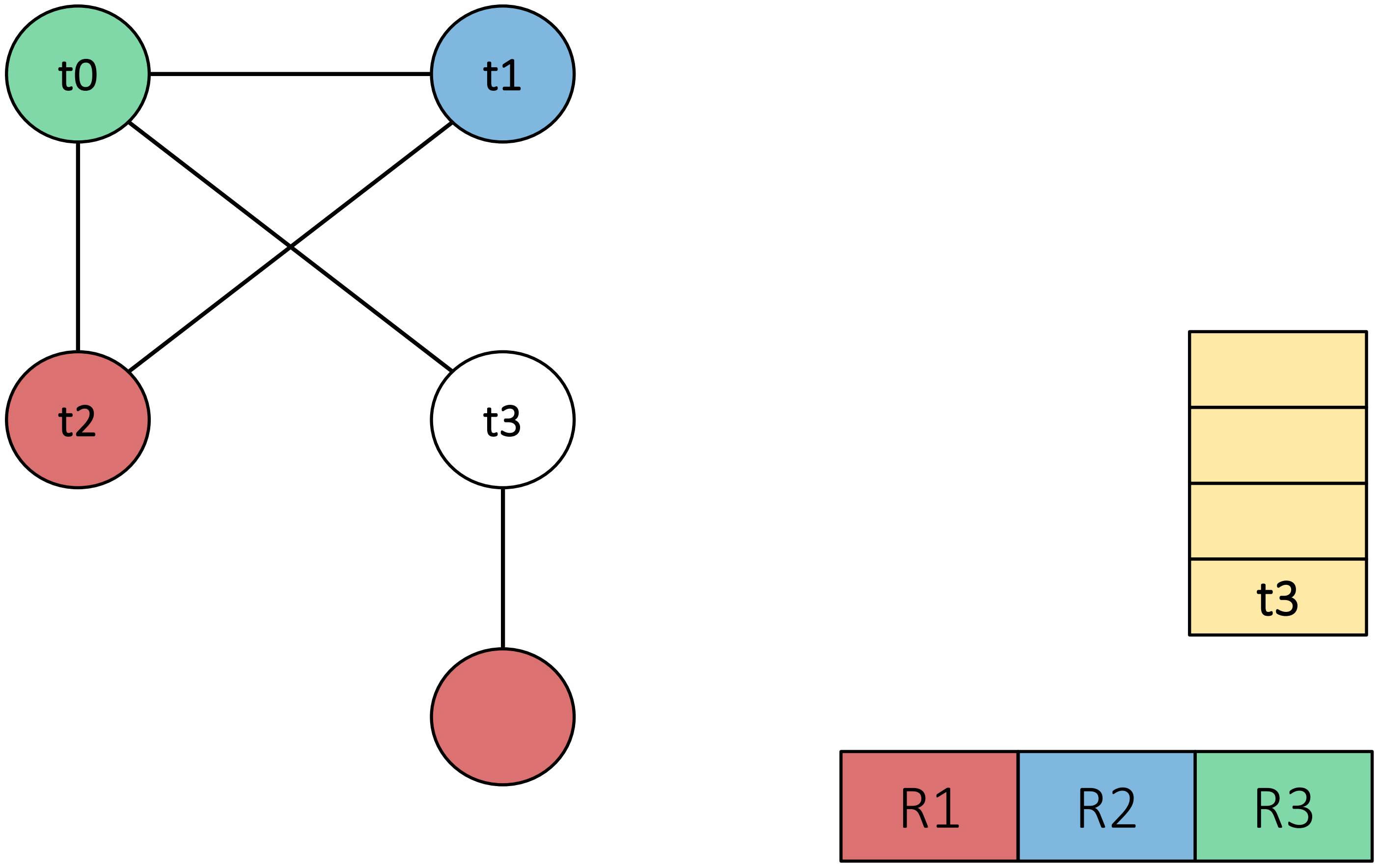
# Graph Coloring



# Graph Coloring



# Graph Coloring



# Graph Coloring