# Automatically Proving Memory Safety and Termination of C-Programs

Alexander Weinert
RWTH Aachen University

Research Area Computer Science 2
Published: (Termination with Pointer Arithmetic, Ströder et al., 2014)

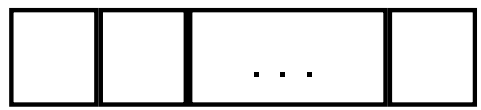
March 13, 2015

# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}
```

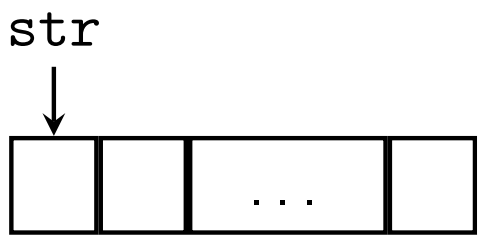
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}
```



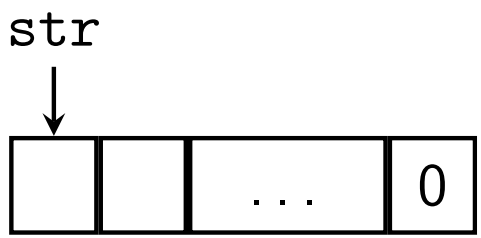
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}
```



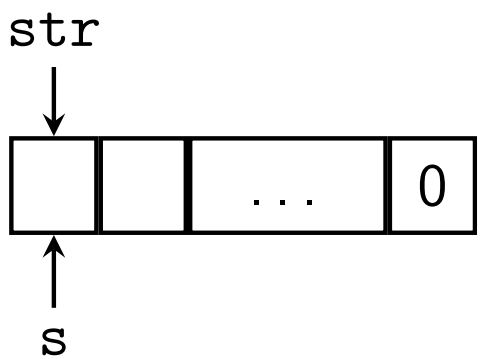
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}
```



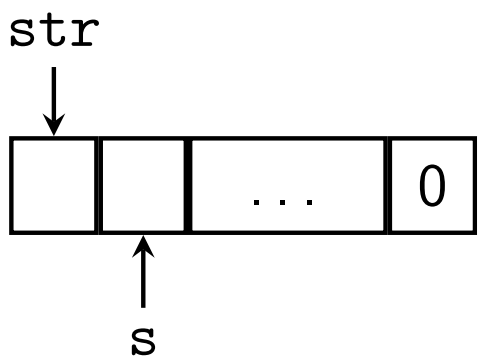
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}
```



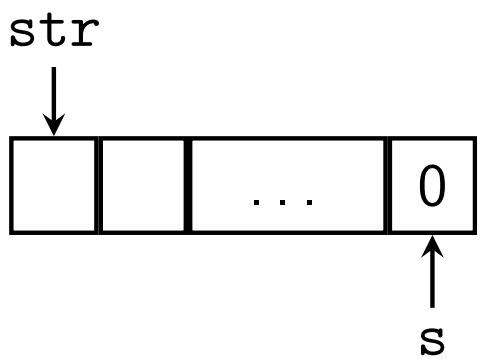
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}
```



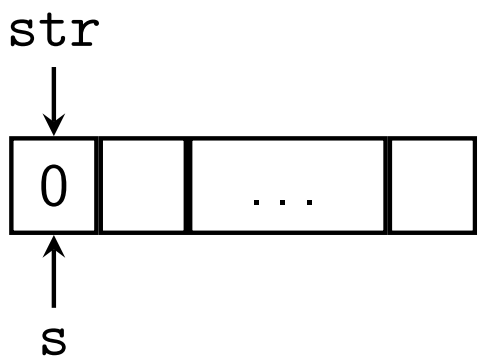
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}
```



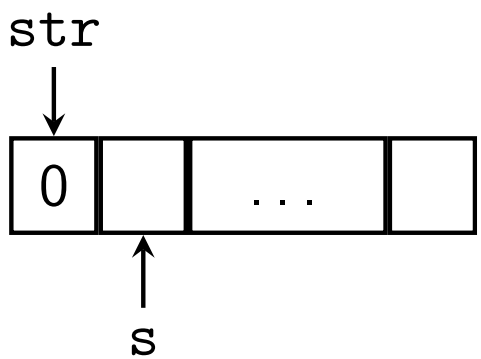
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}
```



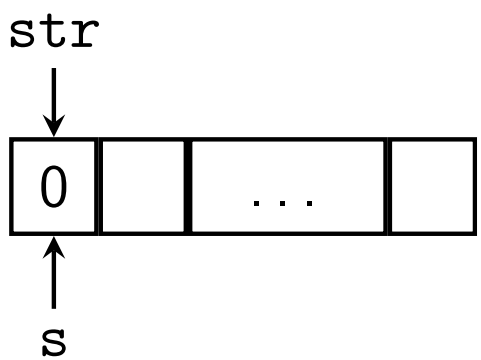
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}
```



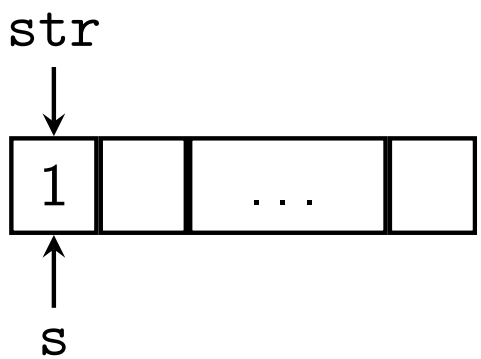
# An Example

```
int strlen(char* str) {
    char* s = str;
    while((*s)++);
    return s-str;
}
```



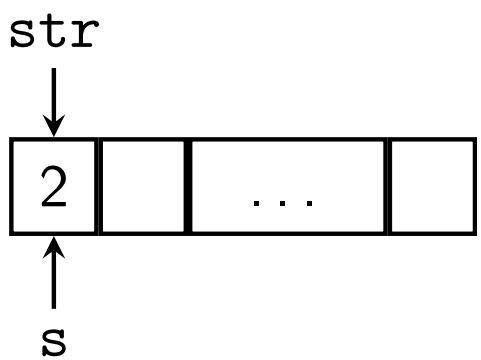
# An Example

```
int strlen(char* str) {
    char* s = str;
    while((*s)++);
    return s-str;
}
```



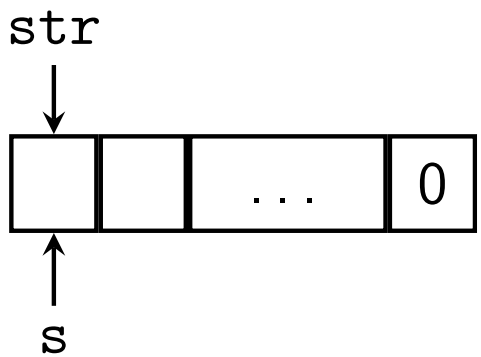
# An Example

```
int strlen(char* str) {
    char* s = str;
    while((*s)++);
    return s-str;
}
```



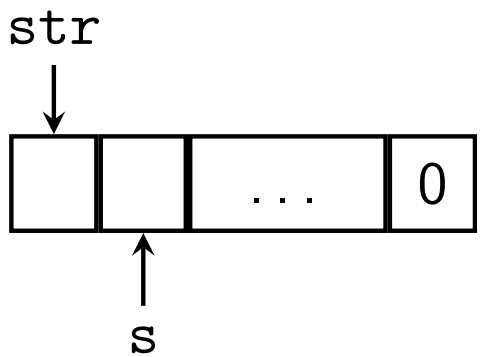
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(s++));
    return s-str;
}
```



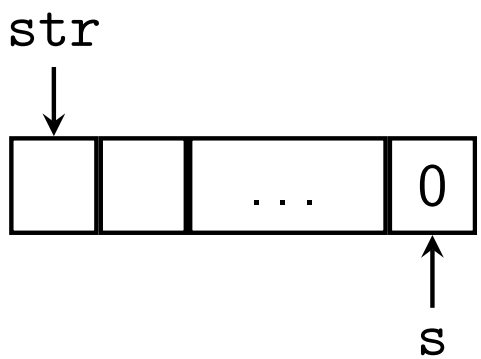
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(s++));
    return s-str;
}
```



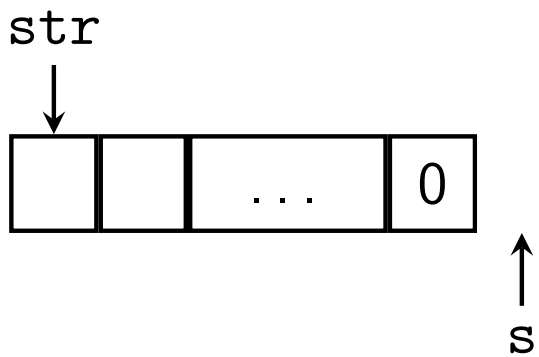
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(s++));
    return s-str;
}
```



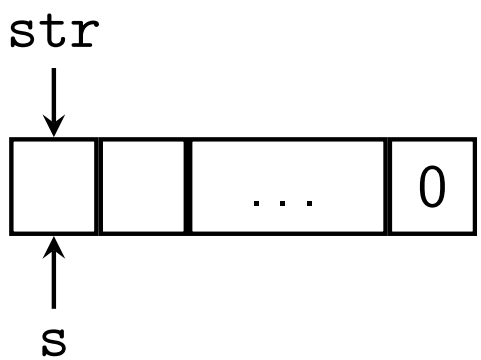
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*(s++));
    return s-str;
}
```



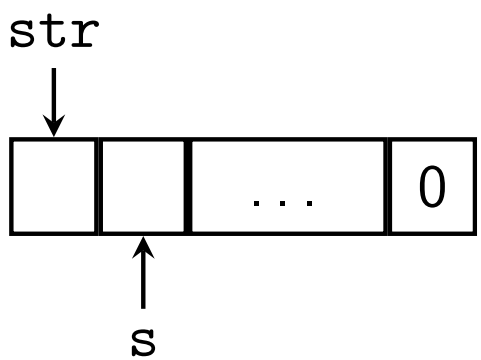
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*s) s++;
    return s-str;
}
```



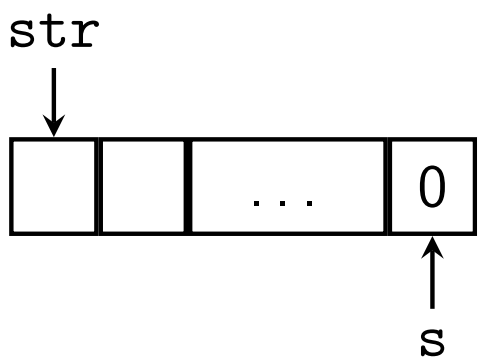
# An Example

```
int strlen(char* str) {
    char* s = str;
    while(*s) s++;
    return s-str;
}
```



# An Example

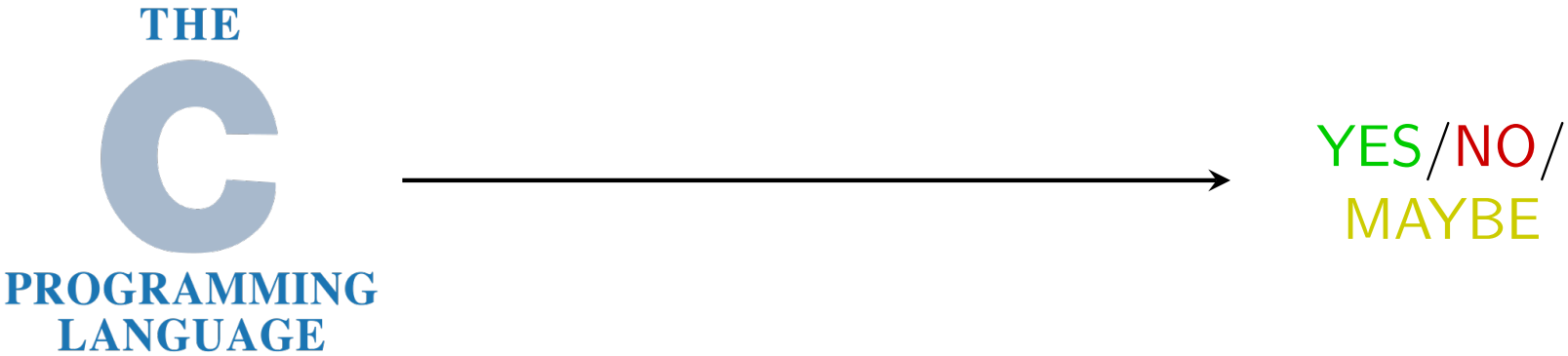
```
int strlen(char* str) {
    char* s = str;
    while(*s) s++;
    return s-str;
}
```



# Big Picture

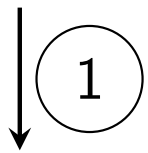

YES/NO/
MAYBE

# Big Picture



# Big Picture


YES/NO/
MAYBE



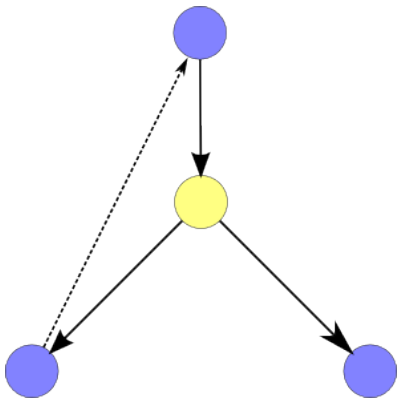

LLVM

# Big Picture


YES/NO/
MAYBE

1

2



LLVM

Symbolic Execution Graph

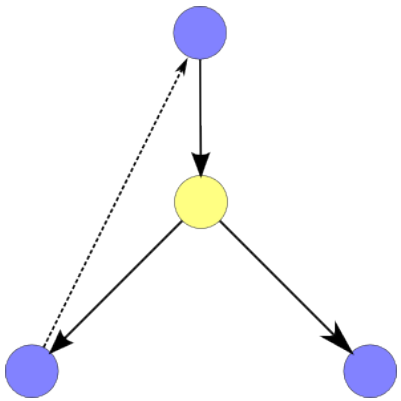
# Big Picture


YES/NO/
MAYBE

1

LLVM

2



Symbolic Execution Graph

3

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

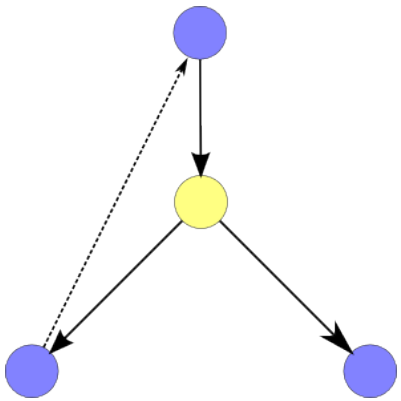
# Big Picture


YES/NO/
MAYBE

1

4

2



3

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

LLVM

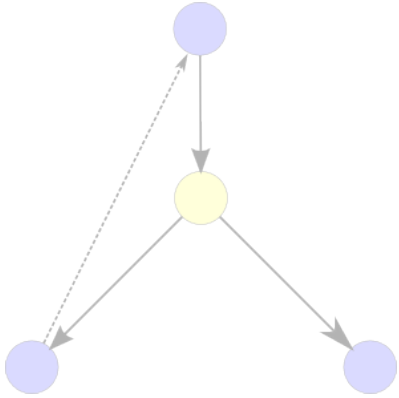
Symbolic Execution Graph

Integer Transition System

# C to LLVM


LLVM

?



Symbolic Execution Graph

?

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

?

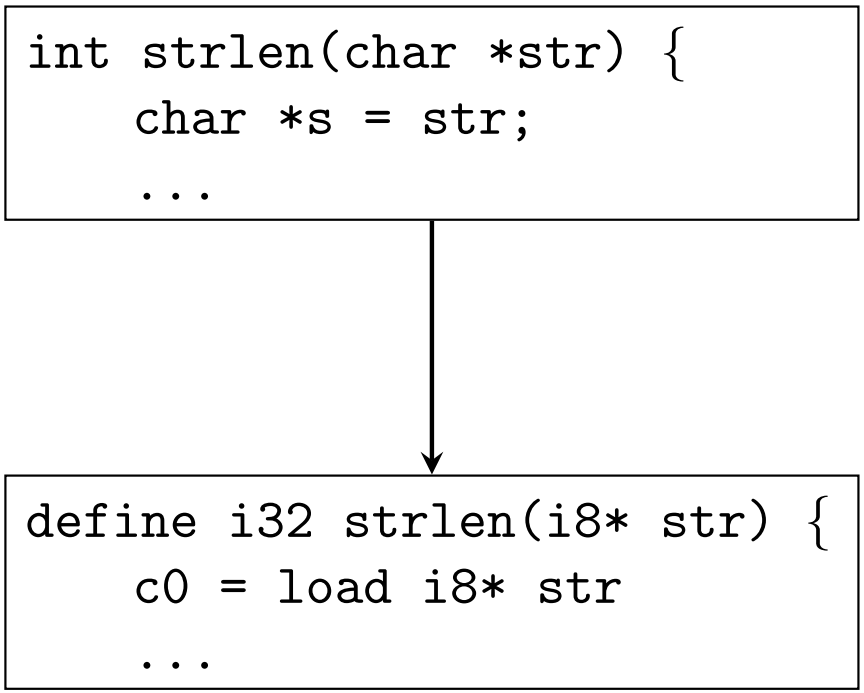
YES/NO/
MAYBE

# LLVM Compiler Infrastructure

```
int strlen(char *str) {
    char *s = str;
    ...
```

C-program

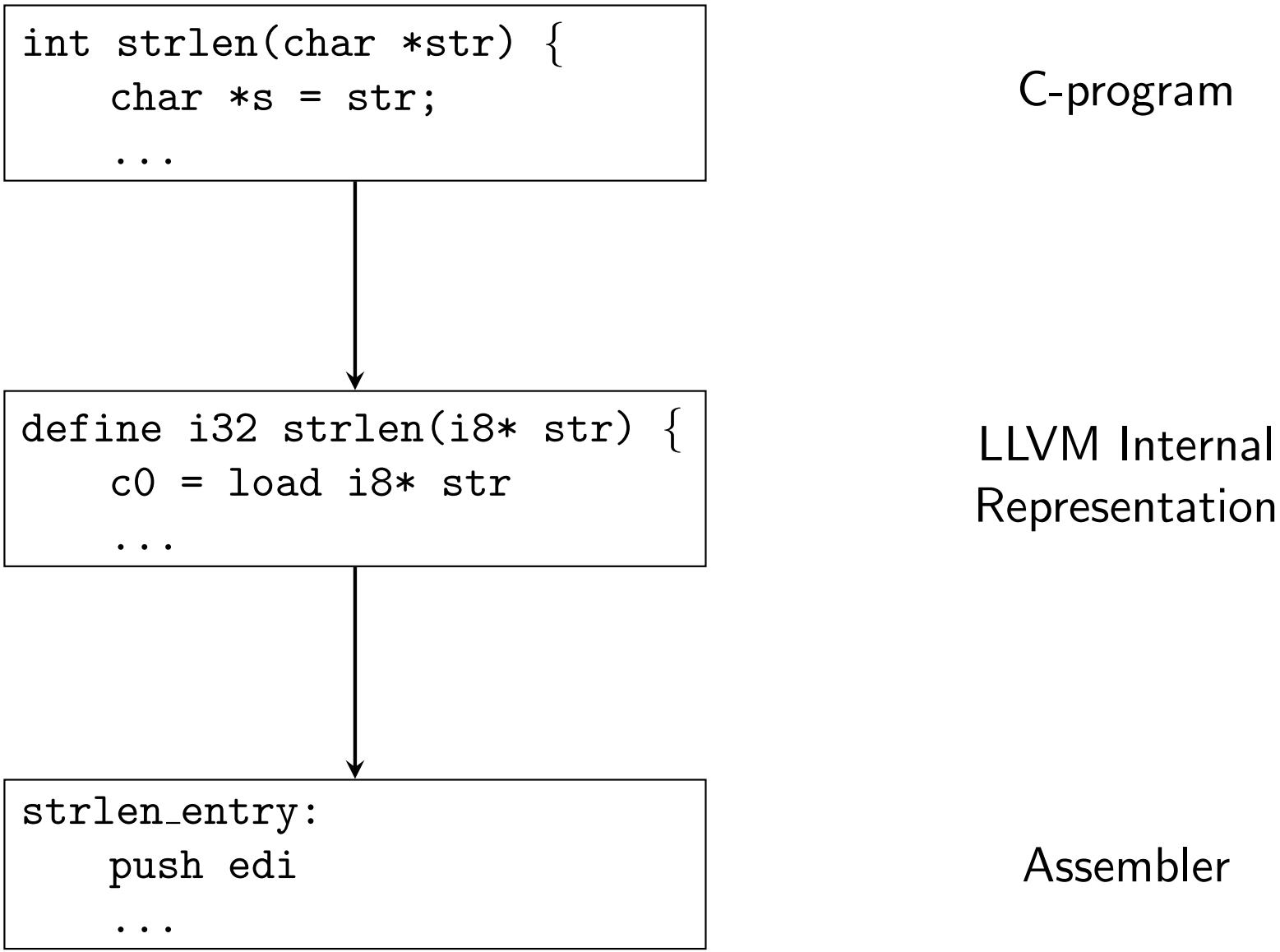
# LLVM Compiler Infrastructure



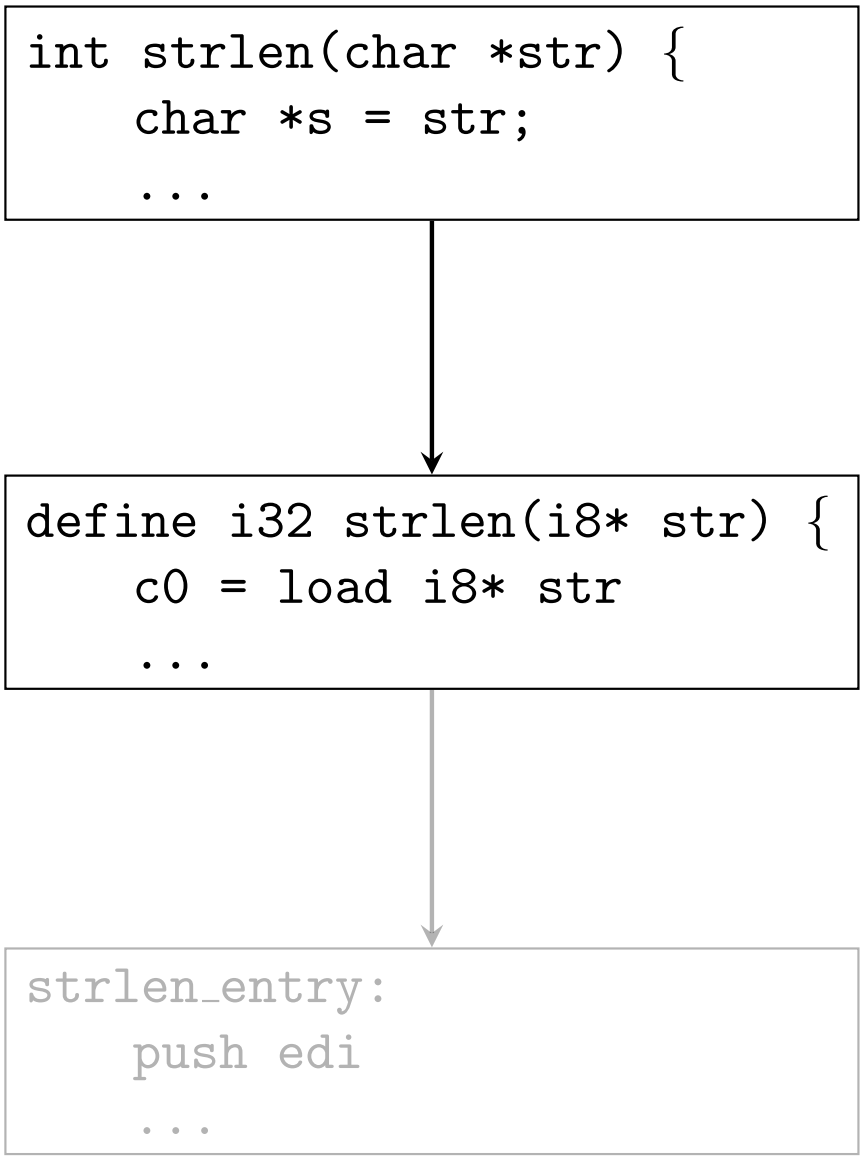

C-program

LLVM Internal Representation

# LLVM Compiler Infrastructure



# LLVM Compiler Infrastructure




C-program

LLVM Internal Representation

Assembler

# Some LLVM Instructions

- Control Flow Instructions
  - `ret`, `br`, `call`, ...

# Some LLVM Instructions

- Control Flow Instructions
  - `ret`, `br`, `call`, `...`
- Arithmetic Instructions
  - `add`, `icmp`, `...`

# Some LLVM Instructions

- Control Flow Instructions
  - `ret`, `br`, `call`, ...
- Arithmetic Instructions
  - `add`, `icmp`, ...
- Bitwise Instructions
  - `shl`, `and`, ...

# Some LLVM Instructions

- Control Flow Instructions
  - `ret`, `br`, `call`, ...
- Arithmetic Instructions
  - `add`, `icmp`, ...
- Bitwise Instructions
  - `shl`, `and`, ...
- Memory Instructions
  - `alloca`, `load`, `store`, ...

# Some LLVM Instructions

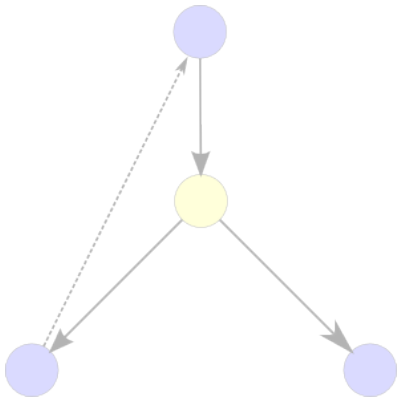
- Control Flow Instructions
  - `ret`, `br`, `call`, ...
- Arithmetic Instructions
  - `add`, `icmp`, ...
- Bitwise Instructions
  - `shl`, `and`, ...
- Memory Instructions
  - `alloca`, `load`, `store`, ...

Complete Reference: `http://llvm.org/docs/LangRef.html`

# LLVM to Symbolic Execution Graph


LLVM

?



Symbolic Execution Graph

?

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

?

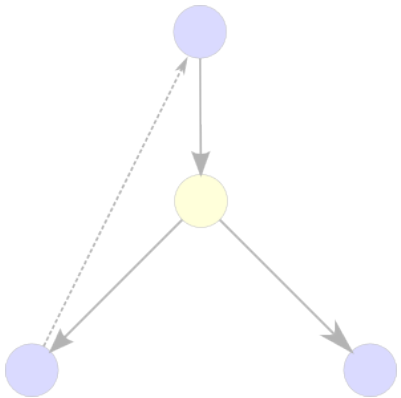
YES/NO/
MAYBE

# LLVM to Symbolic Execution Graph


✓

LLVM

?



Symbolic Execution Graph

?

YES/NO/
MAYBE

?

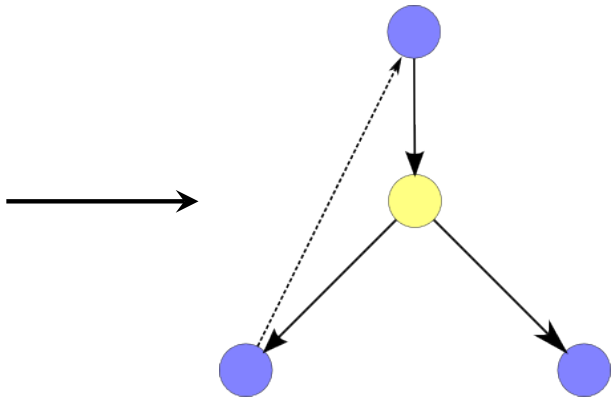
$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

# LLVM to Symbolic Execution Graph


✓

LLVM



Symbolic Execution Graph

?

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

?

YES/NO/
MAYBE

# Problem Definition

Given: LLVM Program, Entry Point

Goal: Description of *at least all possible runs* from the entry point

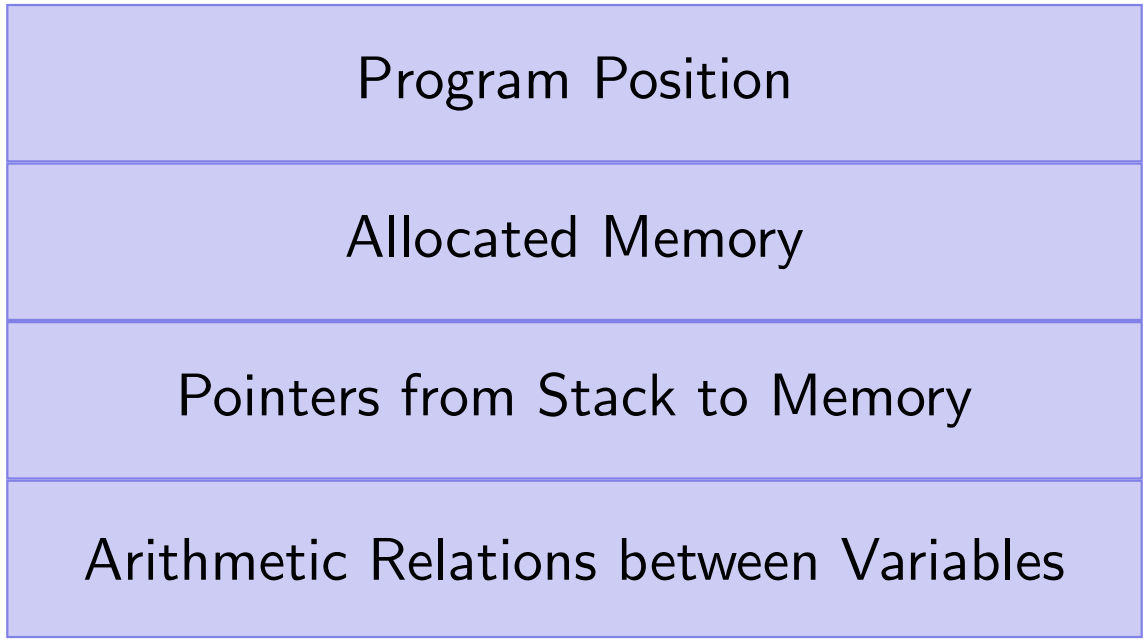
# Problem Definition

Given: LLVM Program, Entry Point

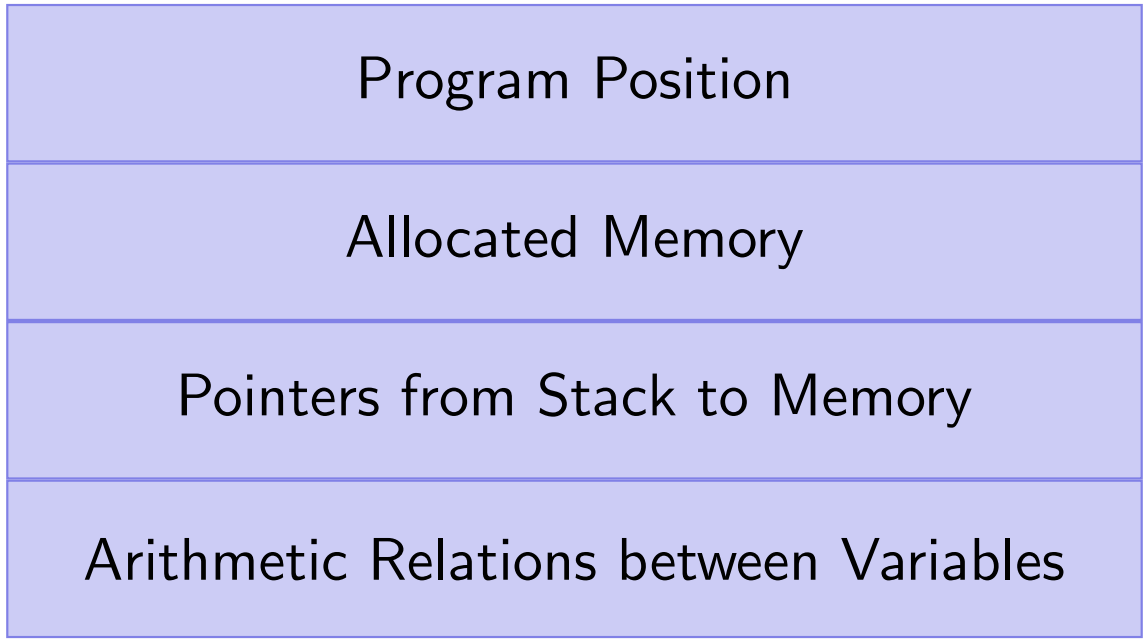
Goal: Description of *at least all possible runs* from the entry point

Idea: Abstract interpretation of program states

# Abstract States



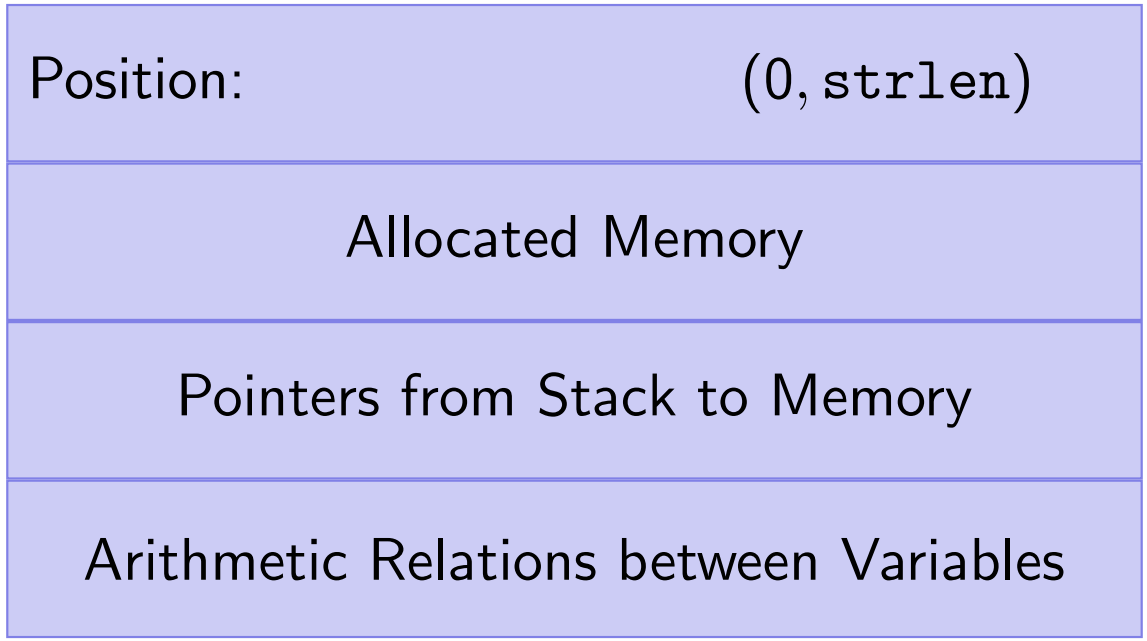
# Abstract States




Entry Point:

```
define i32 strlen(i8* str) {
    c0 = load i8* str
    ...
```

# Abstract States




Entry Point:

```
define i32 strlen(i8* str) {
    c0 = load i8* str
    ...
```

# Abstract States

| | |
|---|---|
| Position: | $(0, \texttt{strlen})$ |
| Alloc: | $[alloc(str, v_{end})]$ |
| Pointers from Stack to Memory | |
| Arithmetic Relations between Variables | |

Entry Point:

```
define i32 strlen(i8* str) {
    c0 = load i8* str
    ...
```

# Abstract States

| | |
|---|---|
| Position: | $(0, \texttt{strlen})$ |
| Alloc: | $[alloc(str, v_{end})]$ |
| Stack $\rightarrow$ Memory: | $v_{end} \hookrightarrow 0$ |
| Arithmetic Relations between Variables | |

Entry Point:

```
define i32 strlen(i8* str) {
    c0 = load i8* str
    ...
```

# Abstract States

| | |
|---|---|
| Position: | $(0, \texttt{strlen})$ |
| Alloc: | $[alloc(str, v_{end})]$ |
| Stack $\rightarrow$ Memory: | $v_{end} \hookrightarrow 0$ |
| Relations: | $\emptyset$ |

Entry Point:

```
define i32 strlen(i8* str) {
    c0 = load i8* str
    ...
```

# Evaluation of Abstract States

```
c0 = load i8* str
```

| |
|---|
| Position: $(0, \texttt{strlen})$ |
| Alloc: $[alloc(str, v_{end})]$ |
| Stack $\rightarrow$ Memory: $v_{end} \hookrightarrow 0$ |
| Relations: $\emptyset$ |

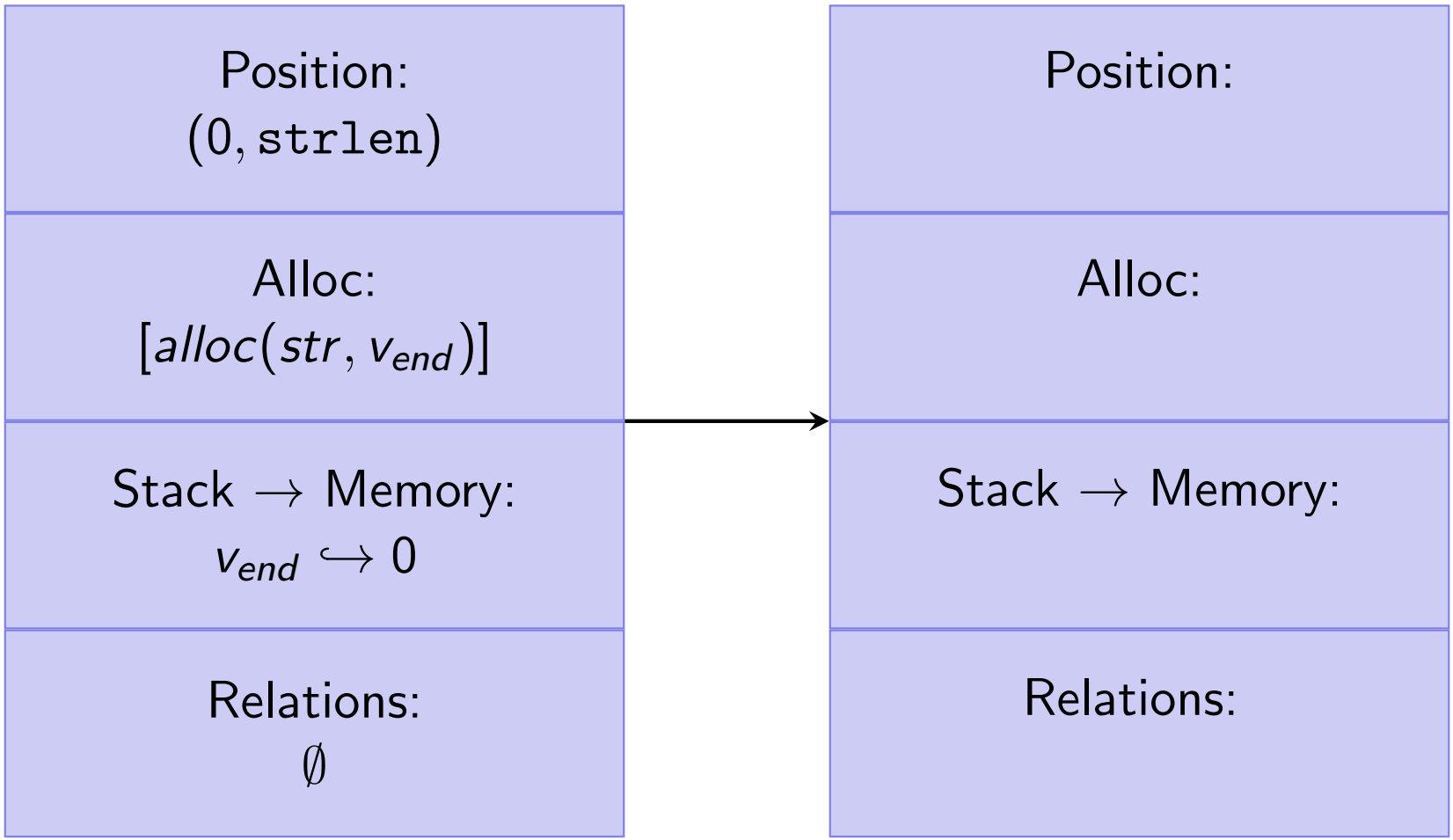
# Evaluation of Abstract States

```
c0 = load i8* str
```

| Position: |
| --- |
| $(0, \texttt{strlen})$ |
| Alloc: |
| $[alloc(str, v_{end})]$ |
| Stack $\rightarrow$ Memory: |
| $v_{end} \hookrightarrow 0$ |
| Relations: |
| $\emptyset$ |

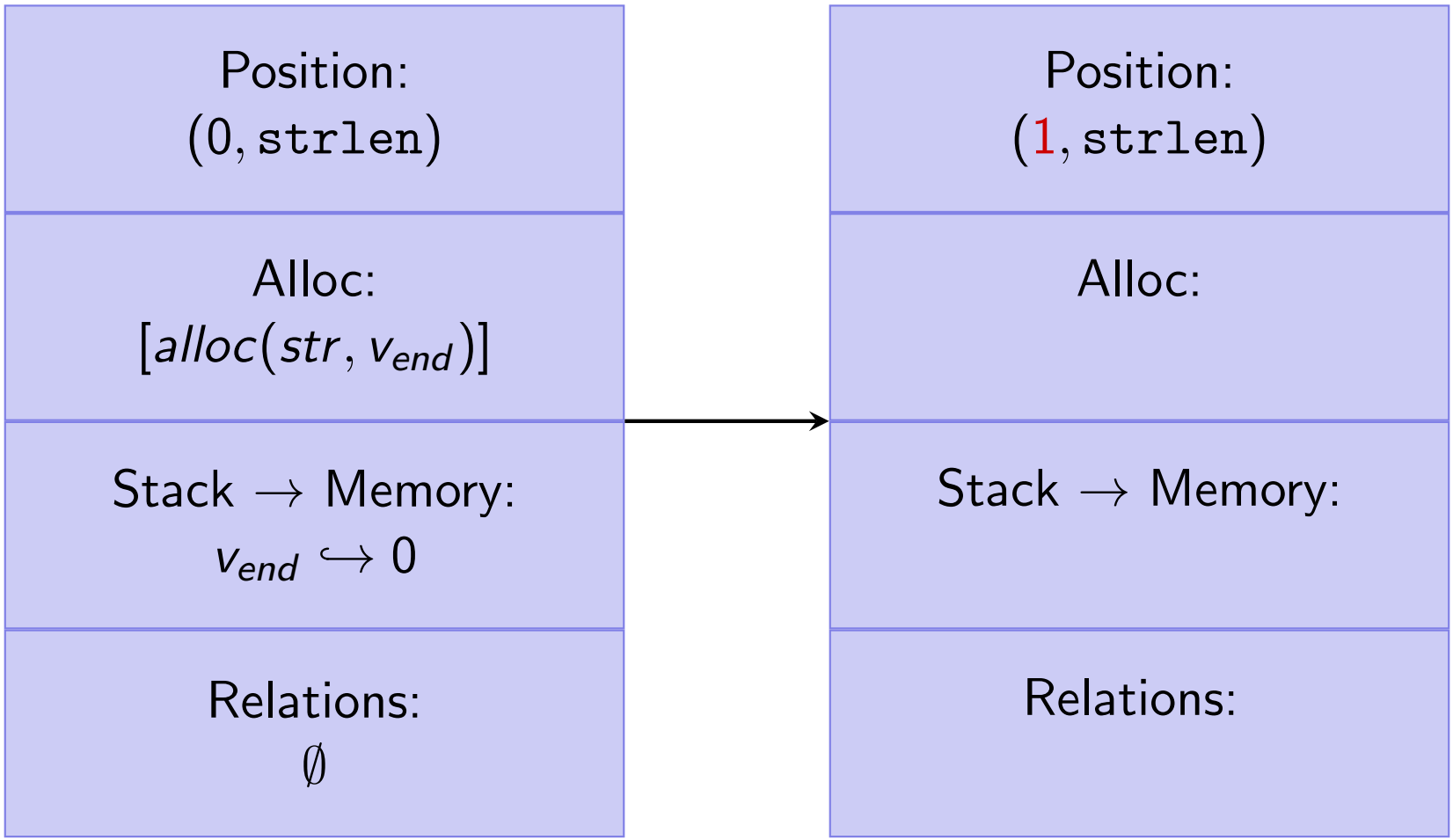
# Evaluation of Abstract States

`c0 = load i8* str`



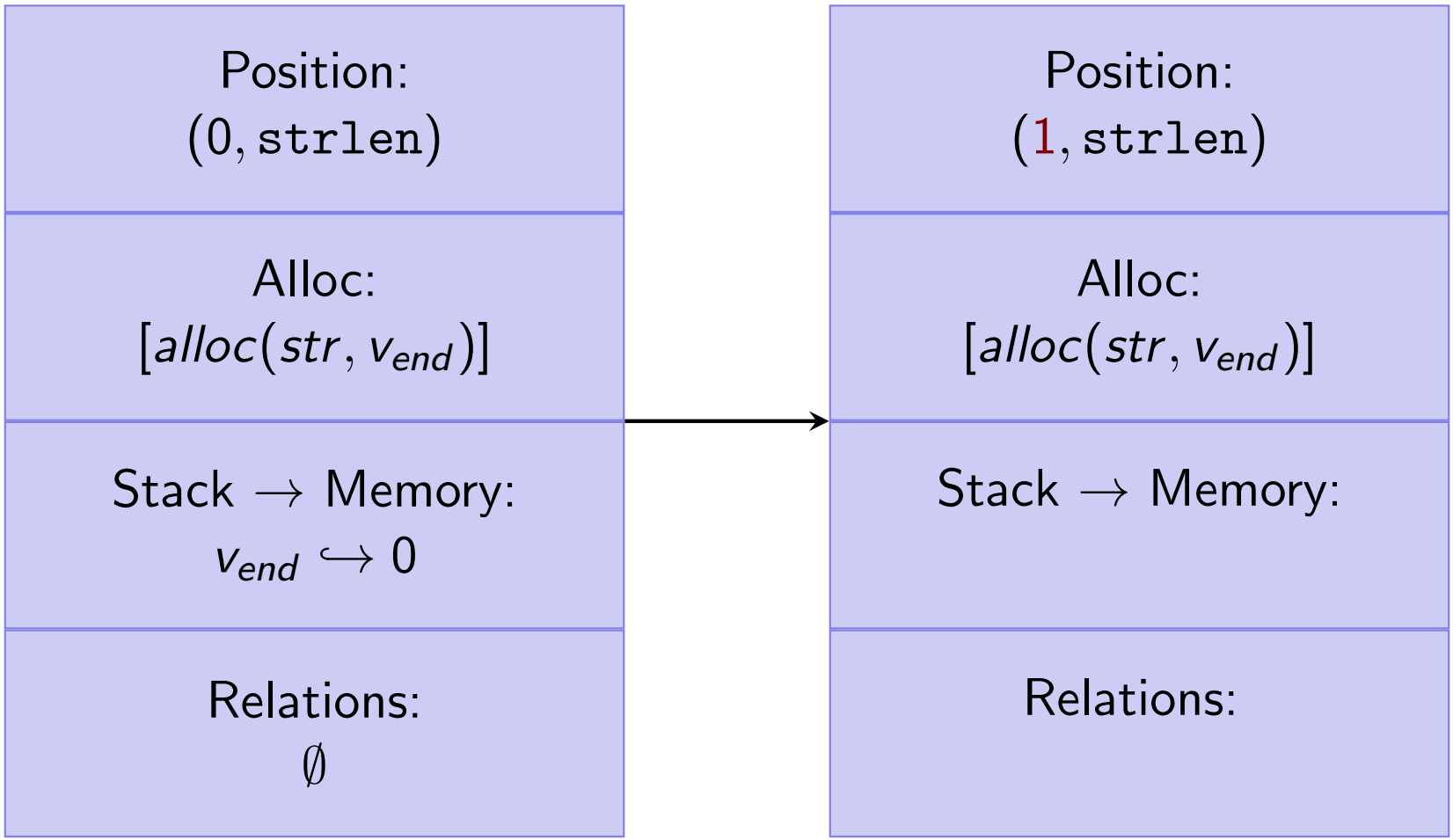
# Evaluation of Abstract States

`c0 = load i8* str`



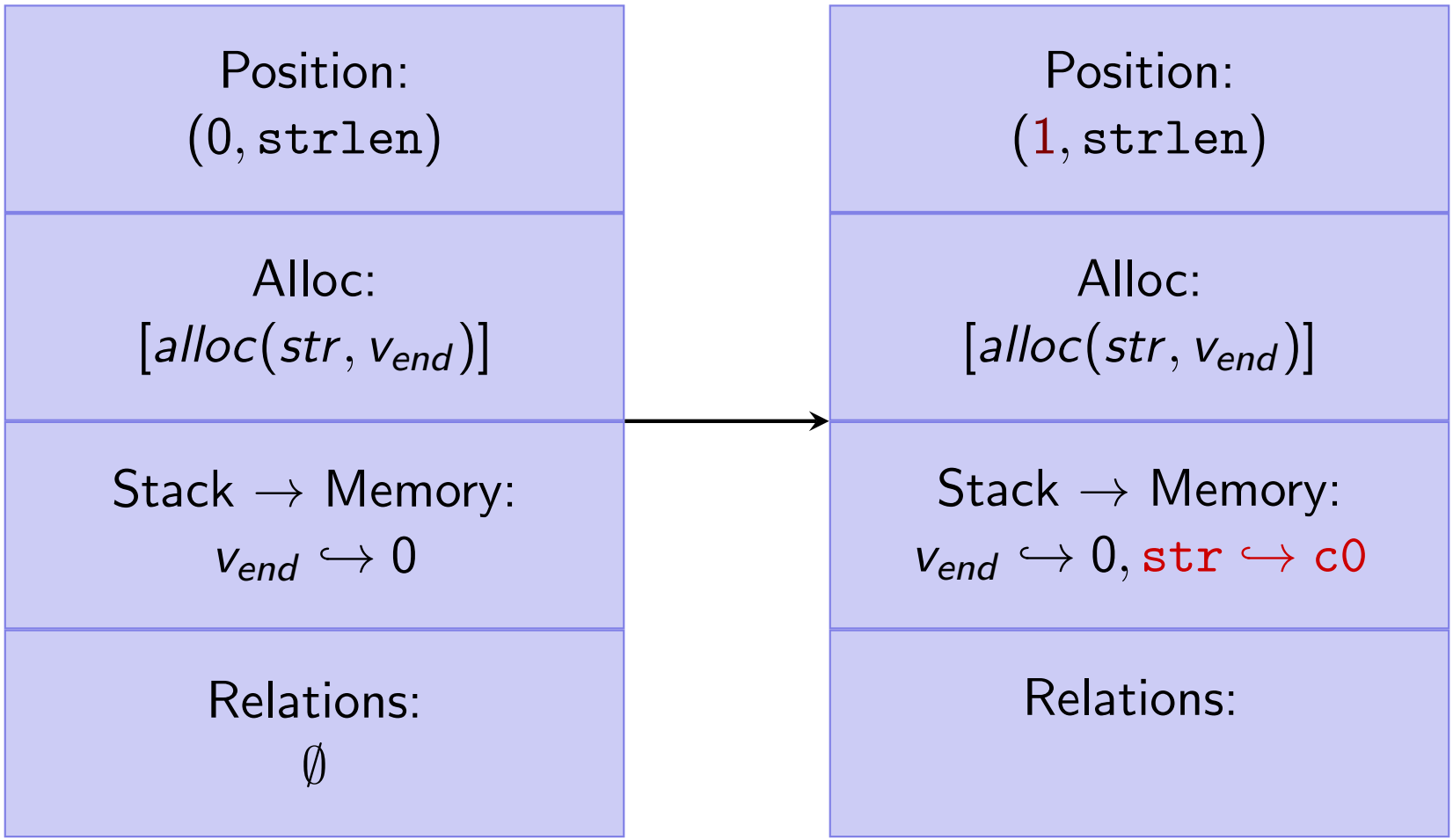
# Evaluation of Abstract States

`c0 = load i8* str`



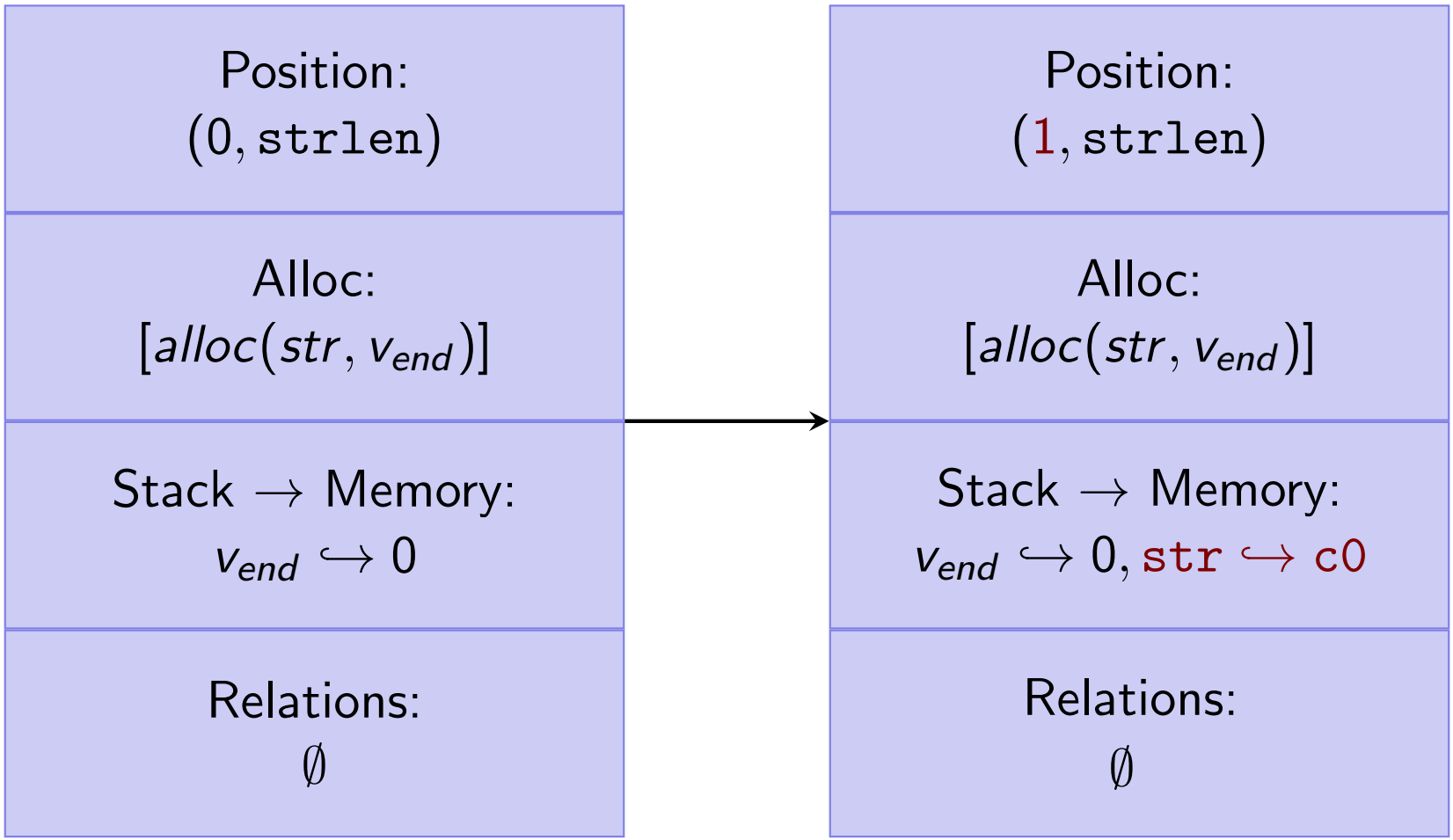
# Evaluation of Abstract States

```
c0 = load i8* str
```



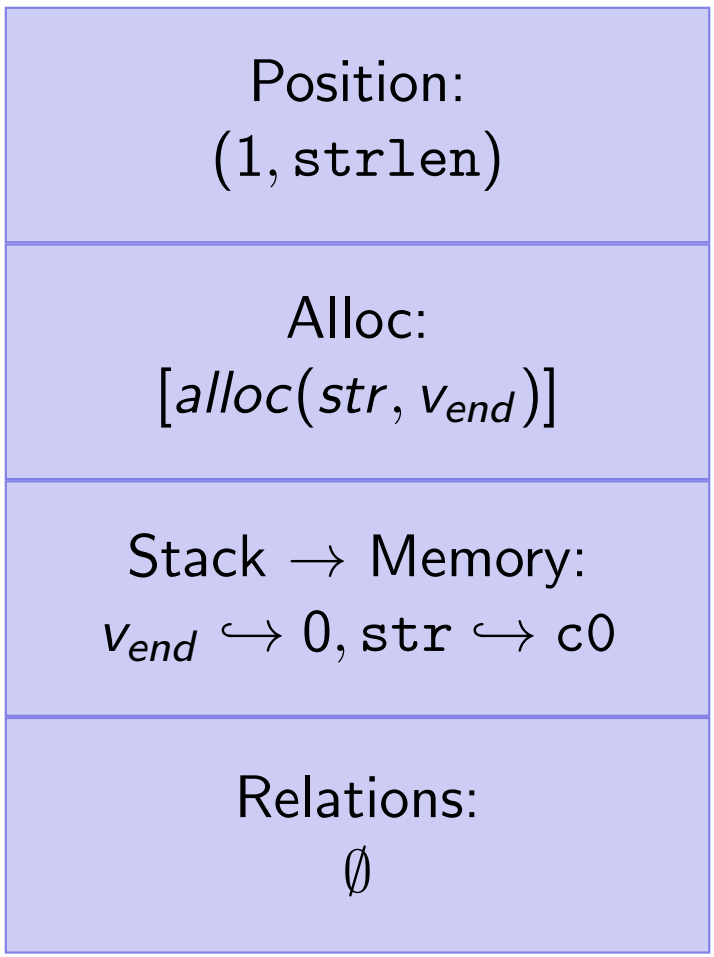
# Evaluation of Abstract States

`c0 = load i8* str`



# Refinement of Abstract States

```
c0zero = icmp eq i8 c0, 0
```



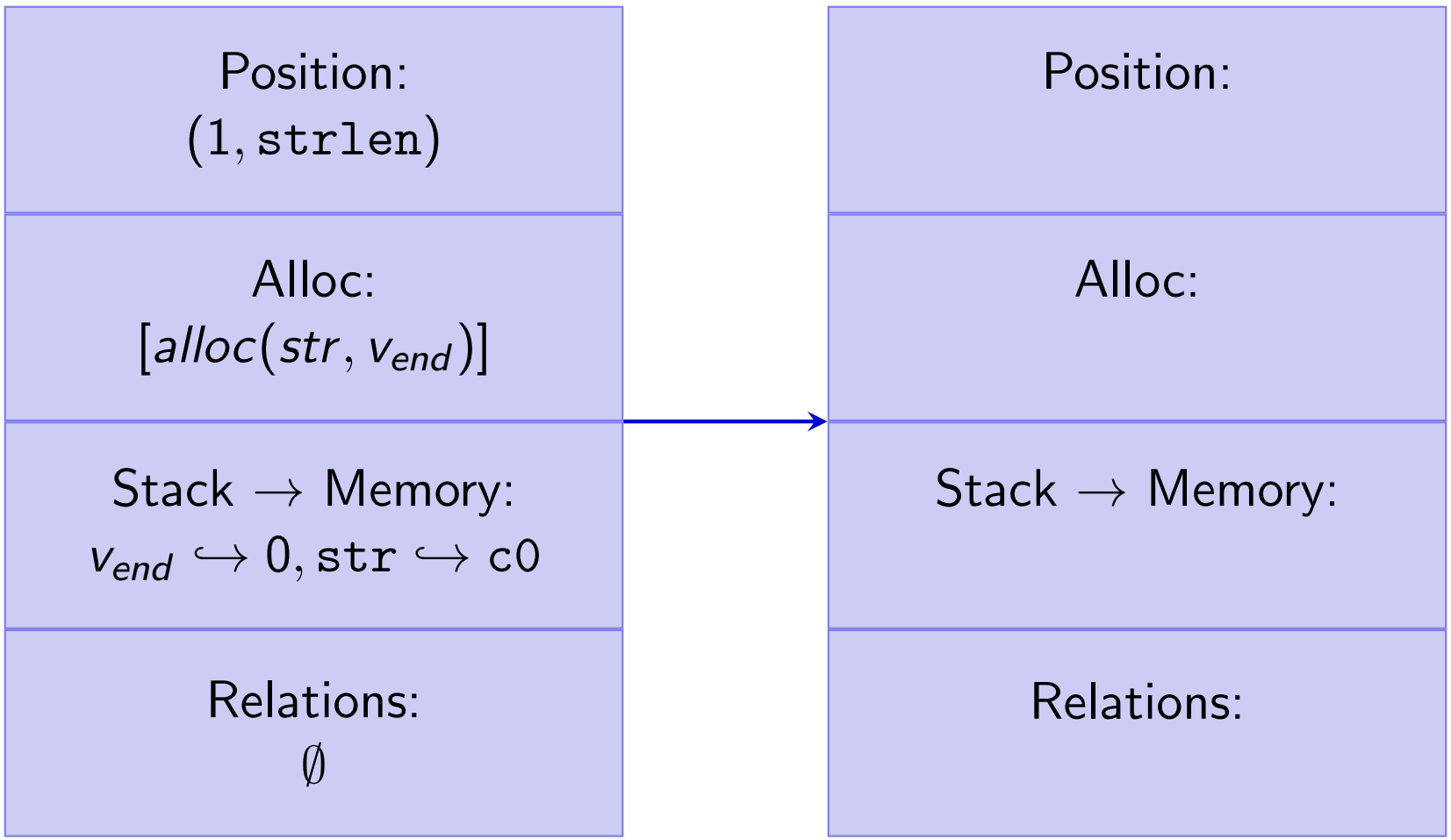
# Refinement of Abstract States

$$\texttt{c0zero} = \underbrace{\texttt{icmp eq i8 c0, 0}}_{\texttt{c0==0}}$$

| Position: $(1, \texttt{strlen})$ |
| --- |
| Alloc: $[alloc(str, v_{end})]$ |
| Stack $\rightarrow$ Memory: $v_{end} \hookrightarrow 0, \texttt{str} \hookrightarrow \texttt{c0}$ |
| Relations: $\emptyset$ |

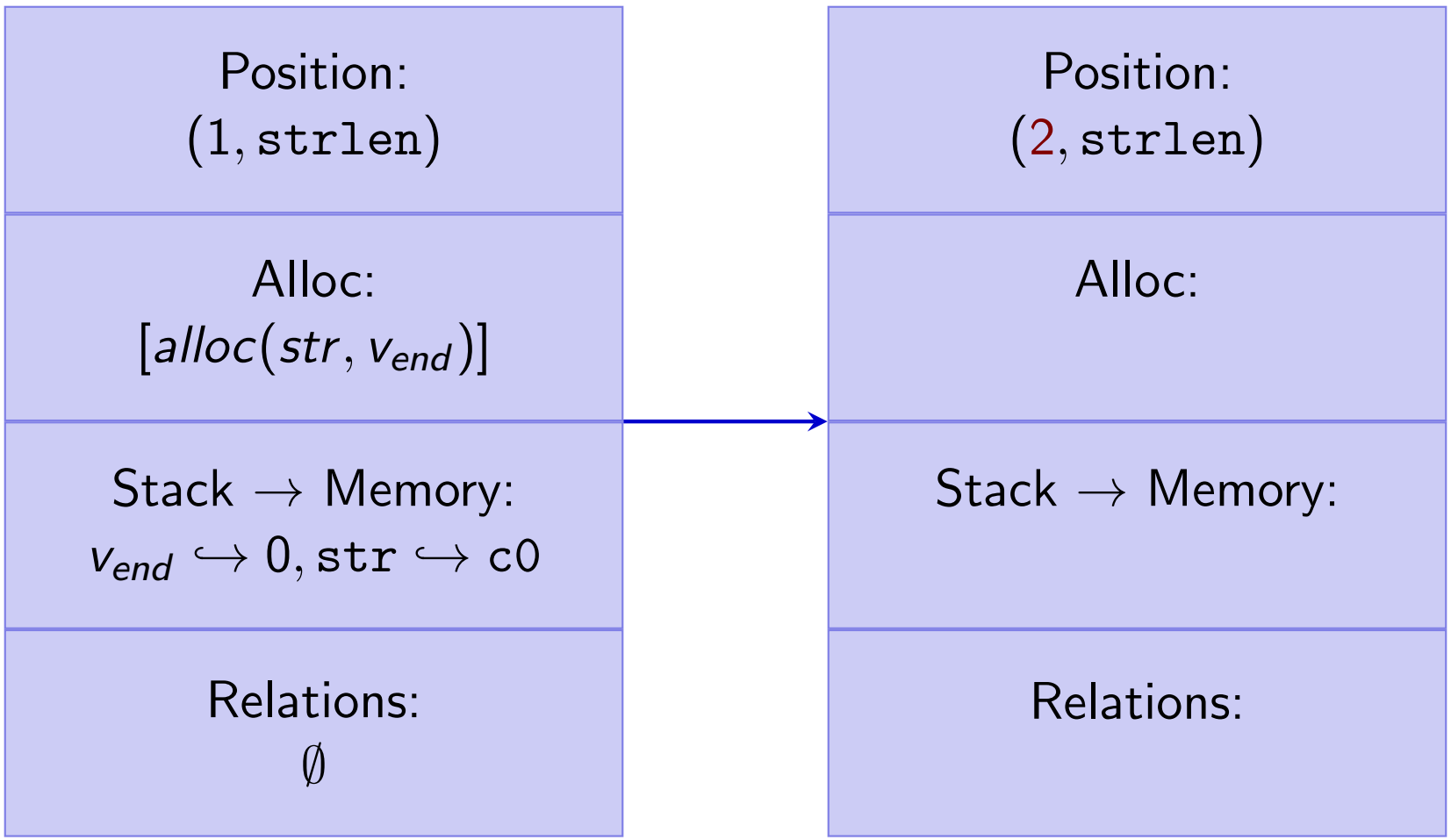
# Refinement of Abstract States

$$\texttt{c0zero} = \underbrace{\texttt{icmp eq i8 c0, 0}}_{\texttt{c0==0}}$$



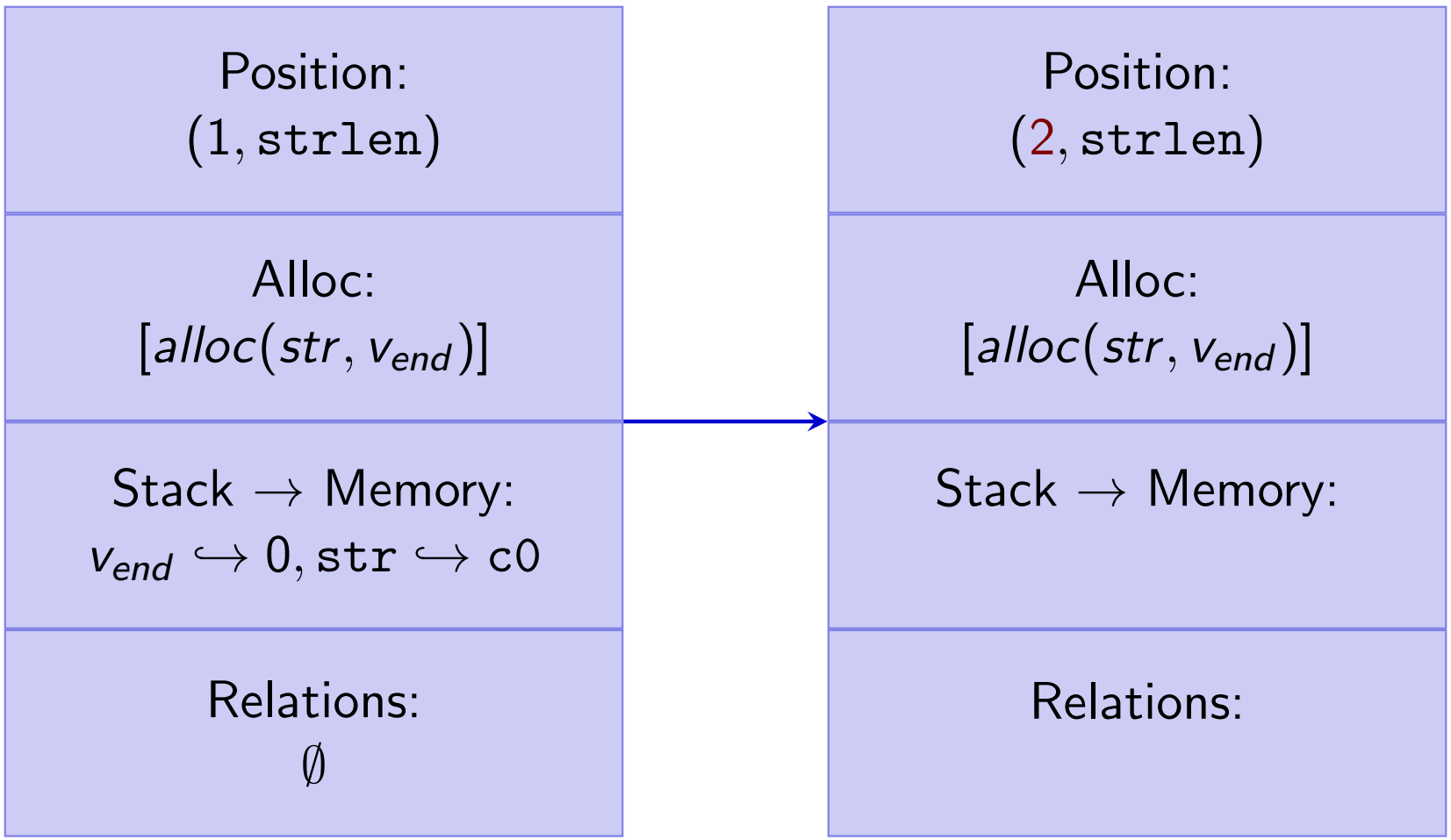
# Refinement of Abstract States

$$\texttt{c0zero} = \underbrace{\texttt{icmp eq i8 c0, 0}}_{\texttt{c0==0}}$$



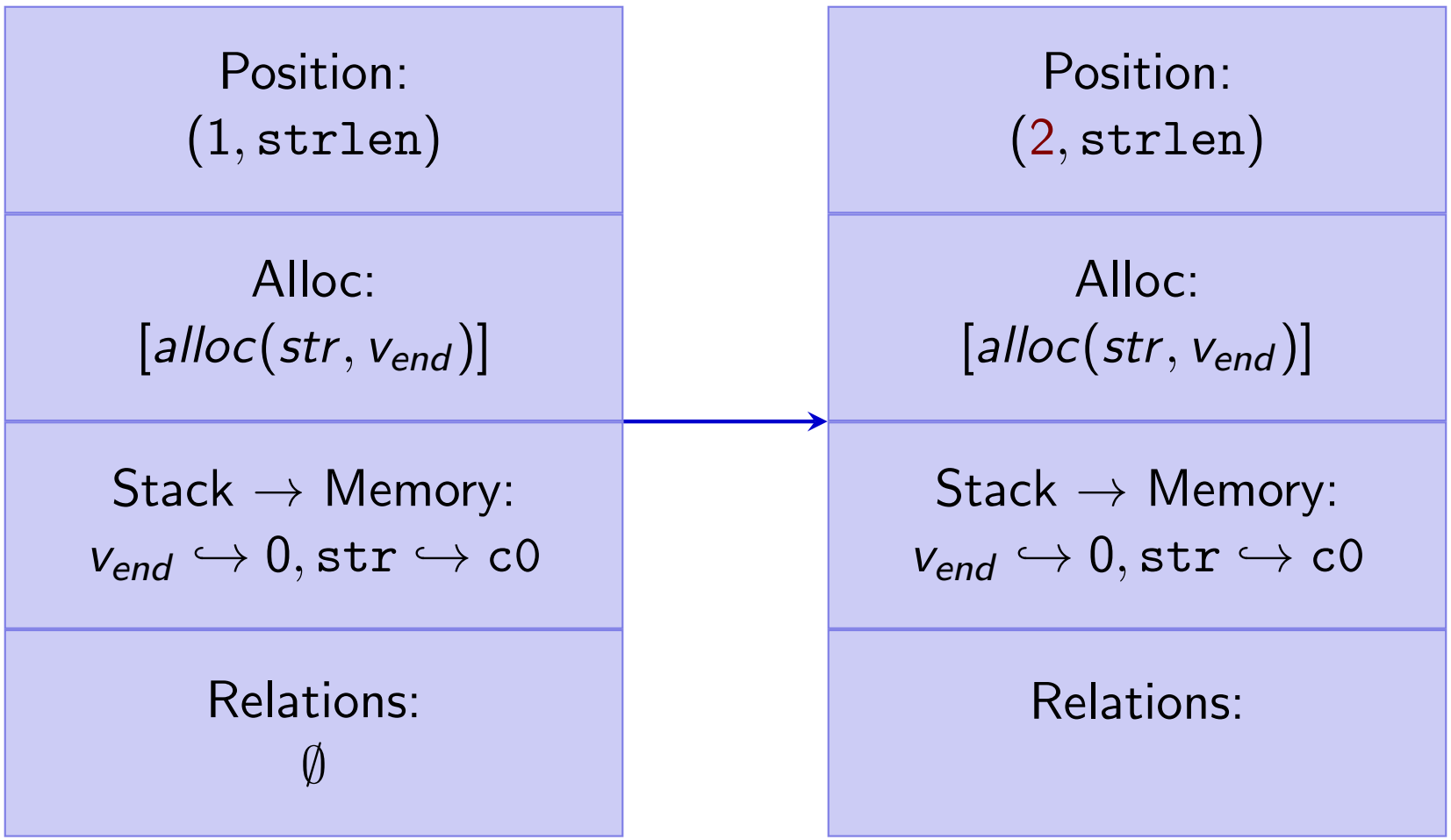
# Refinement of Abstract States

$$\texttt{c0zero} = \underbrace{\texttt{icmp eq i8 c0, 0}}_{\texttt{c0==0}}$$



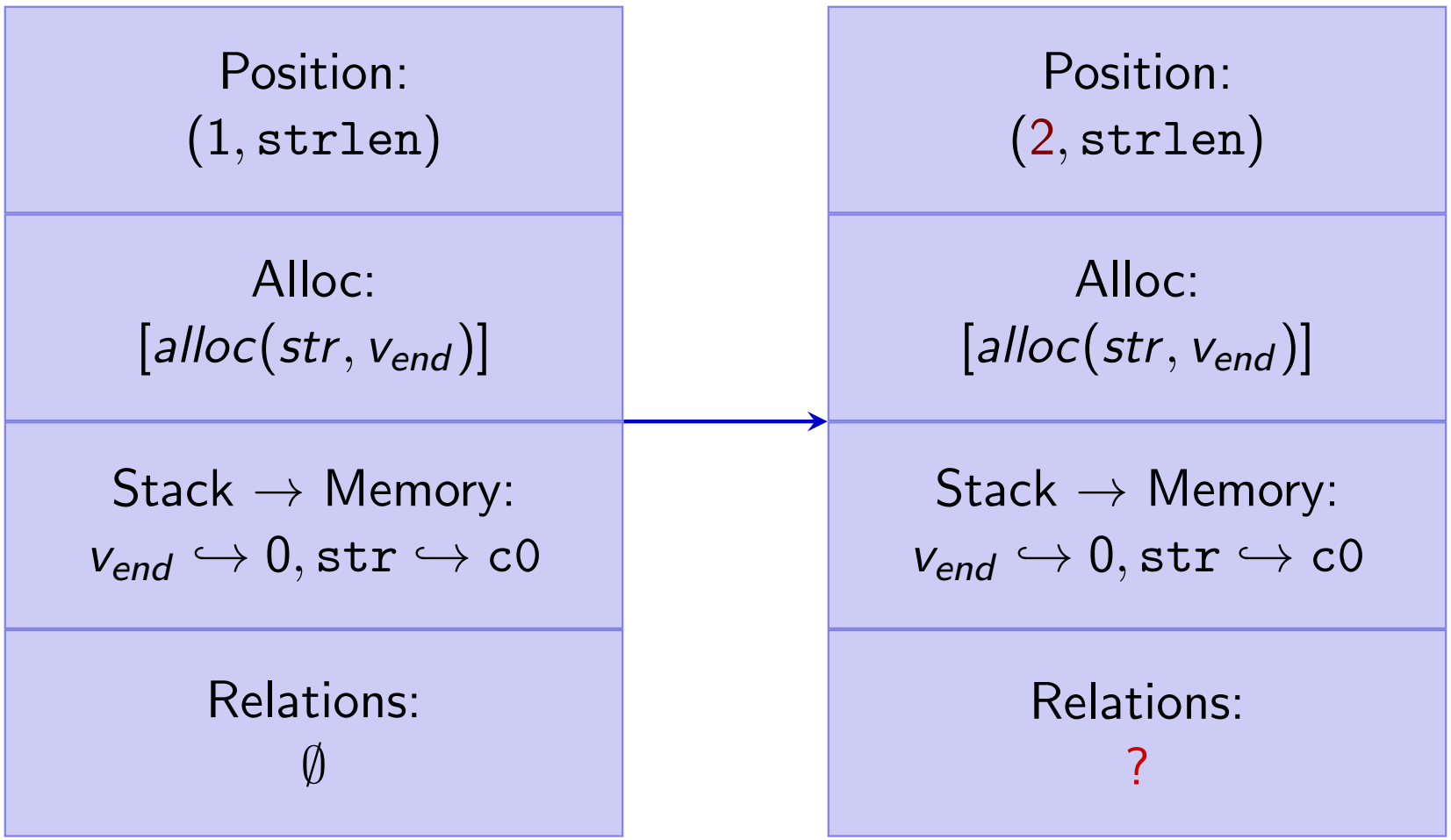
# Refinement of Abstract States

$$\texttt{c0zero} = \underbrace{\texttt{icmp eq i8 c0, 0}}_{\texttt{c0==0}}$$



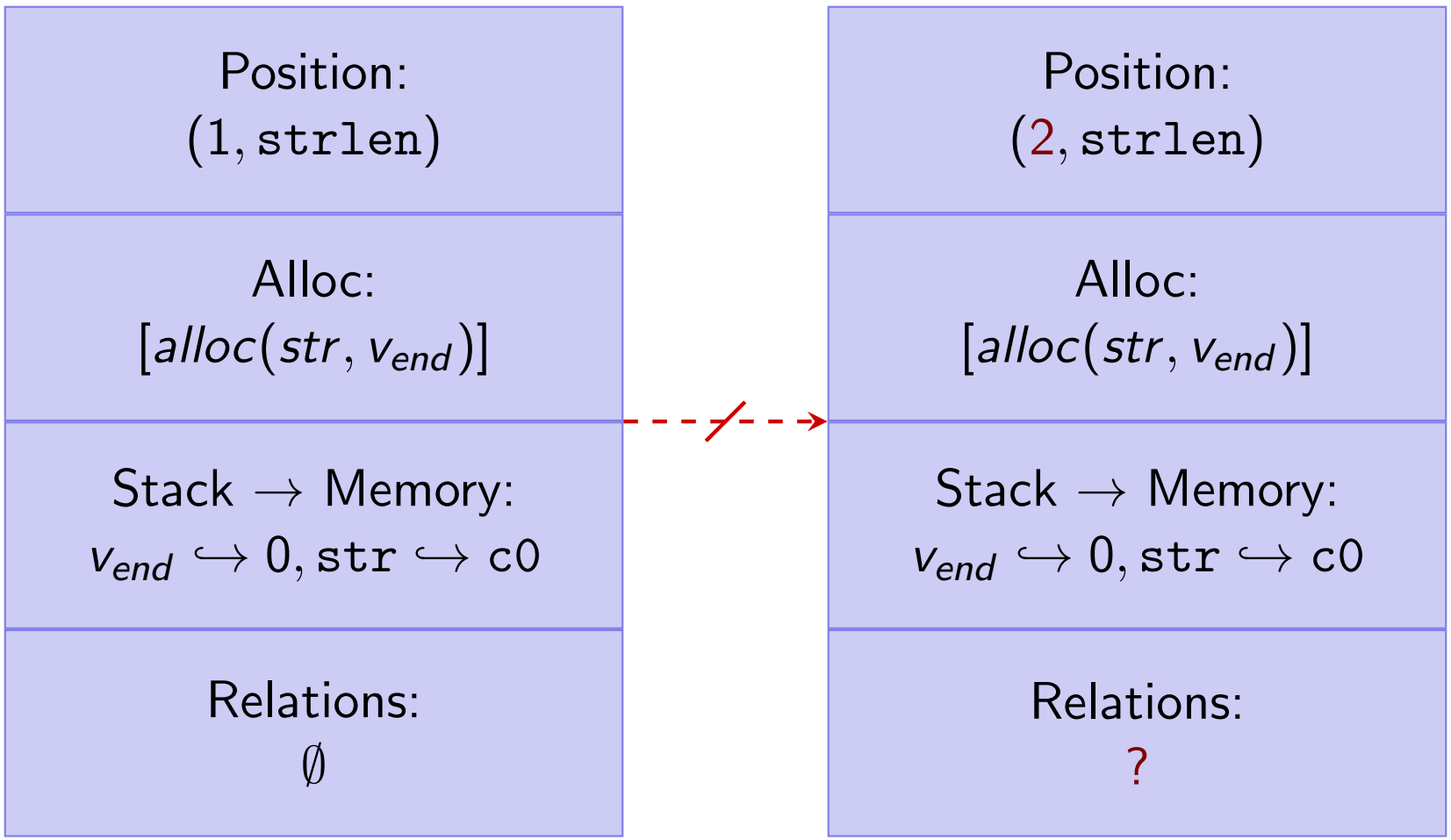
# Refinement of Abstract States

$$\texttt{c0zero} = \underbrace{\texttt{icmp eq i8 c0, 0}}_{\texttt{c0==0}}$$



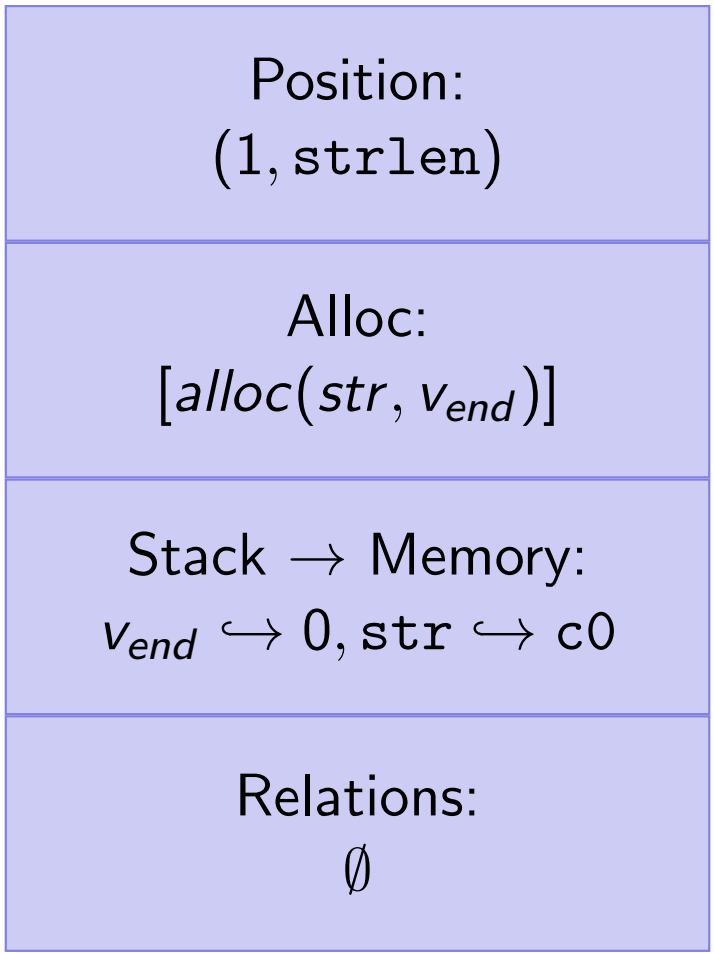
# Refinement of Abstract States

$$\texttt{c0zero} = \underbrace{\texttt{icmp eq i8 c0, 0}}_{\texttt{c0==0}}$$



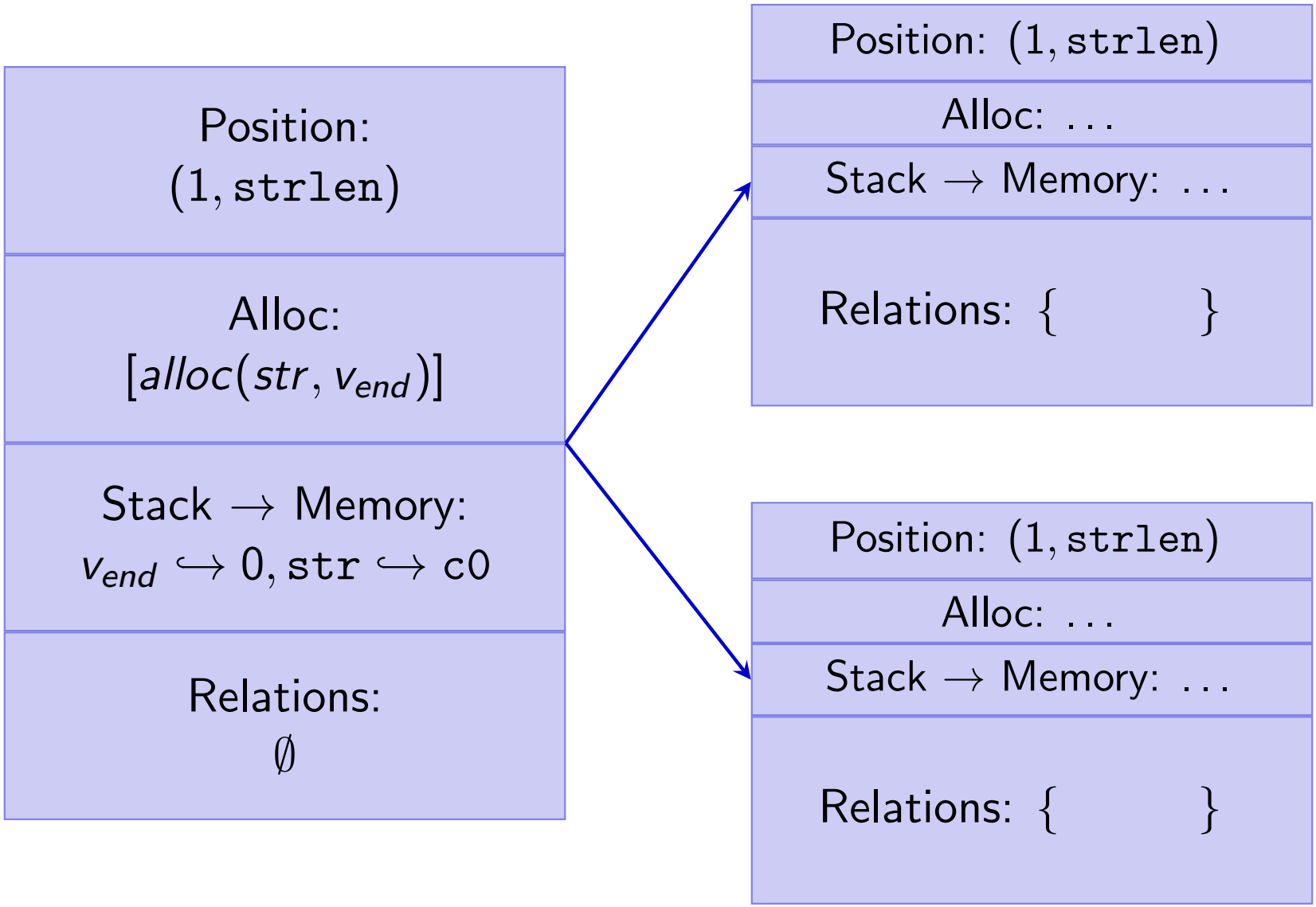
# Refinement of Abstract States

```
c0zero = icmp eq i8 c0, 0
```



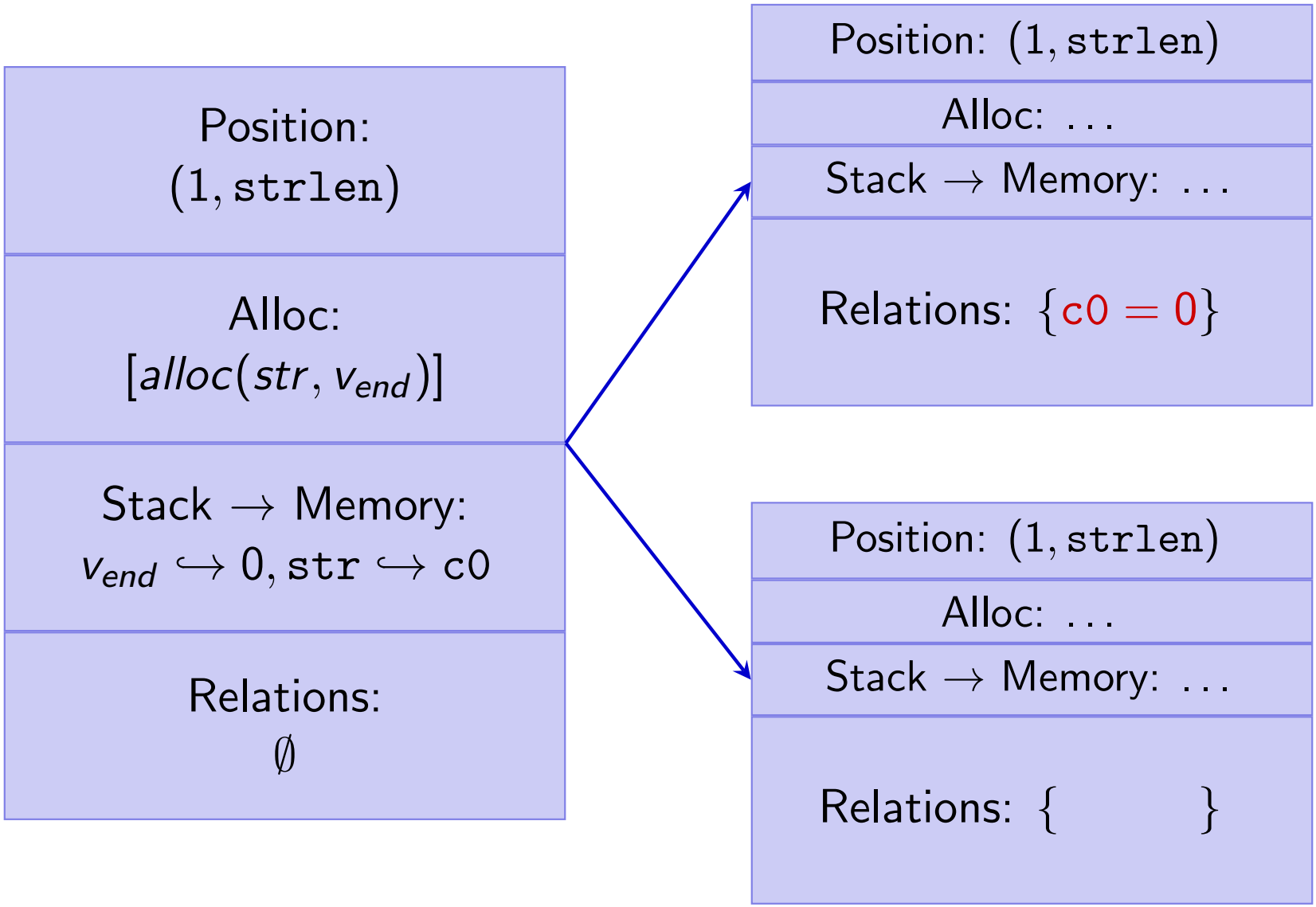
# Refinement of Abstract States

```
c0zero = icmp eq i8 c0, 0
```



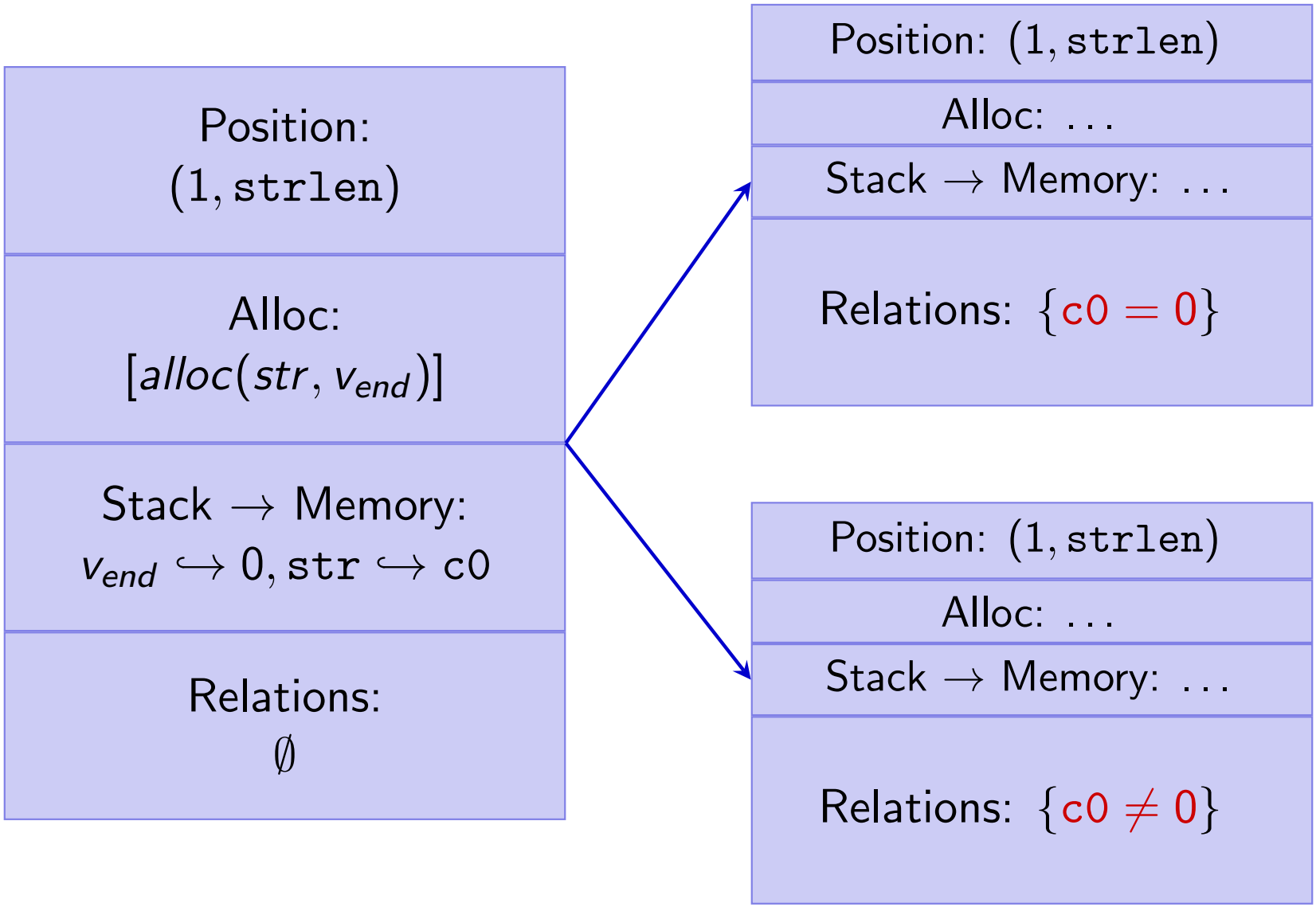
# Refinement of Abstract States

```
c0zero = icmp eq i8 c0, 0
```



# Refinement of Abstract States

```
c0zero = icmp eq i8 c0, 0
```

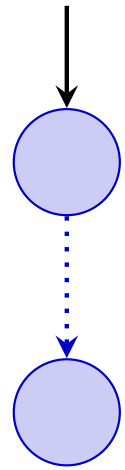


# Merging of Abstract States

```
int strlen(char *str) {
    char *s = str;
    while(*s) s++;
    return s-str;
}
```

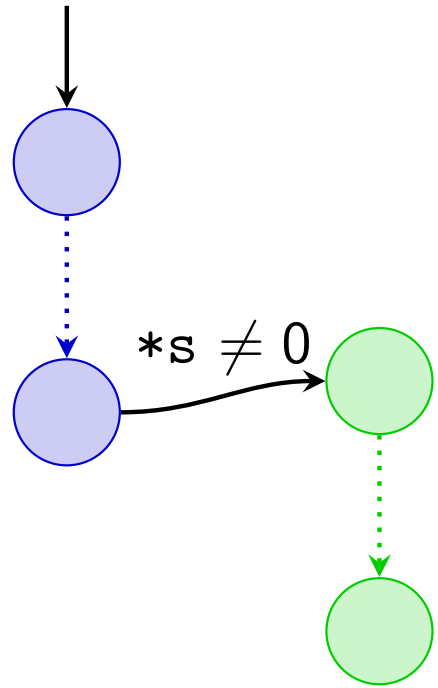
# Merging of Abstract States

```
int strlen(char *str) {
    char *s = str;
    while(*s) s++;
    return s-str;
}
```



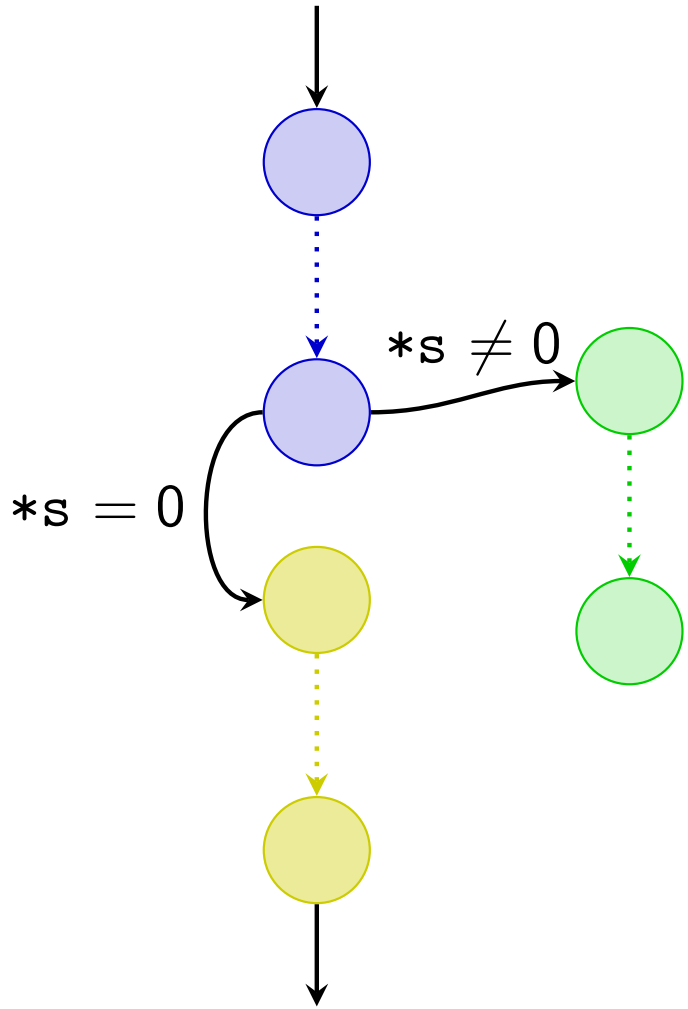
# Merging of Abstract States

```
int strlen(char *str) {
    char *s = str;
    while(*s) s++;
    return s-str;
}
```



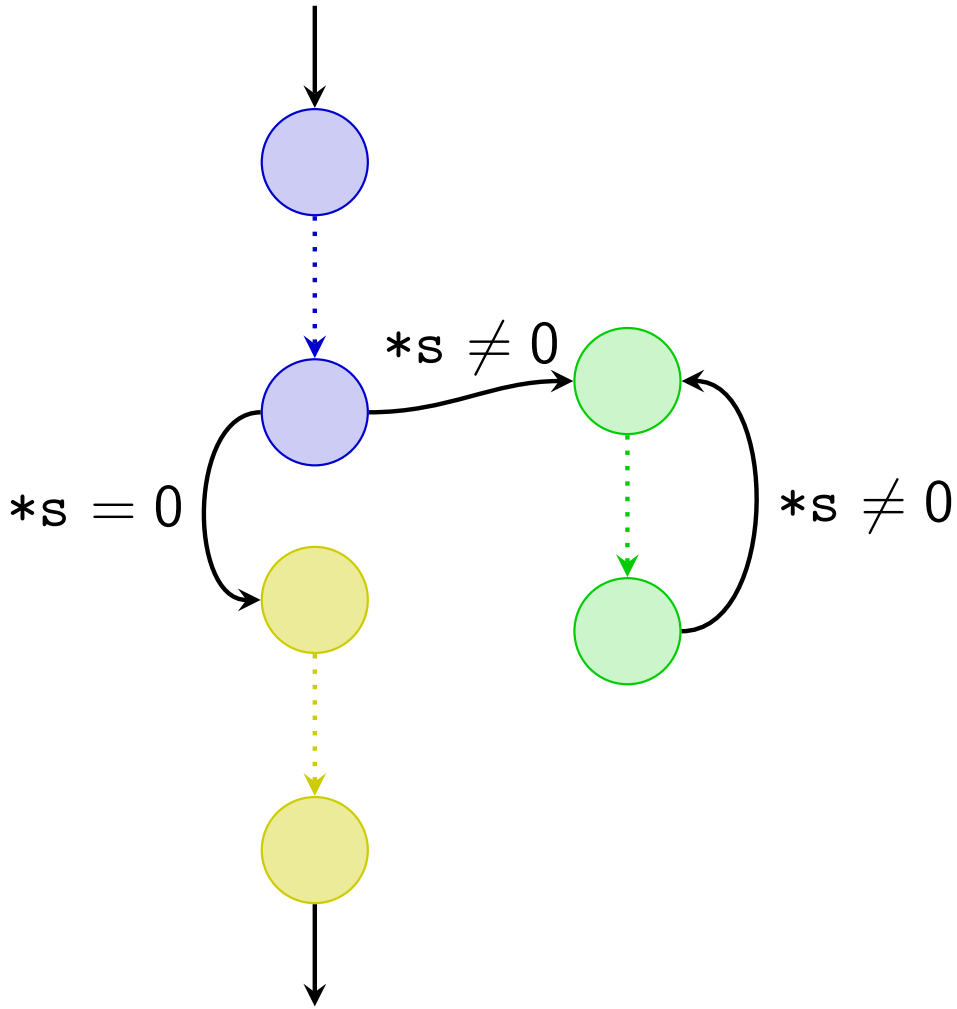
# Merging of Abstract States

```
int strlen(char *str) {
    char *s = str;
    while(*s) s++;
    return s-str;
}
```



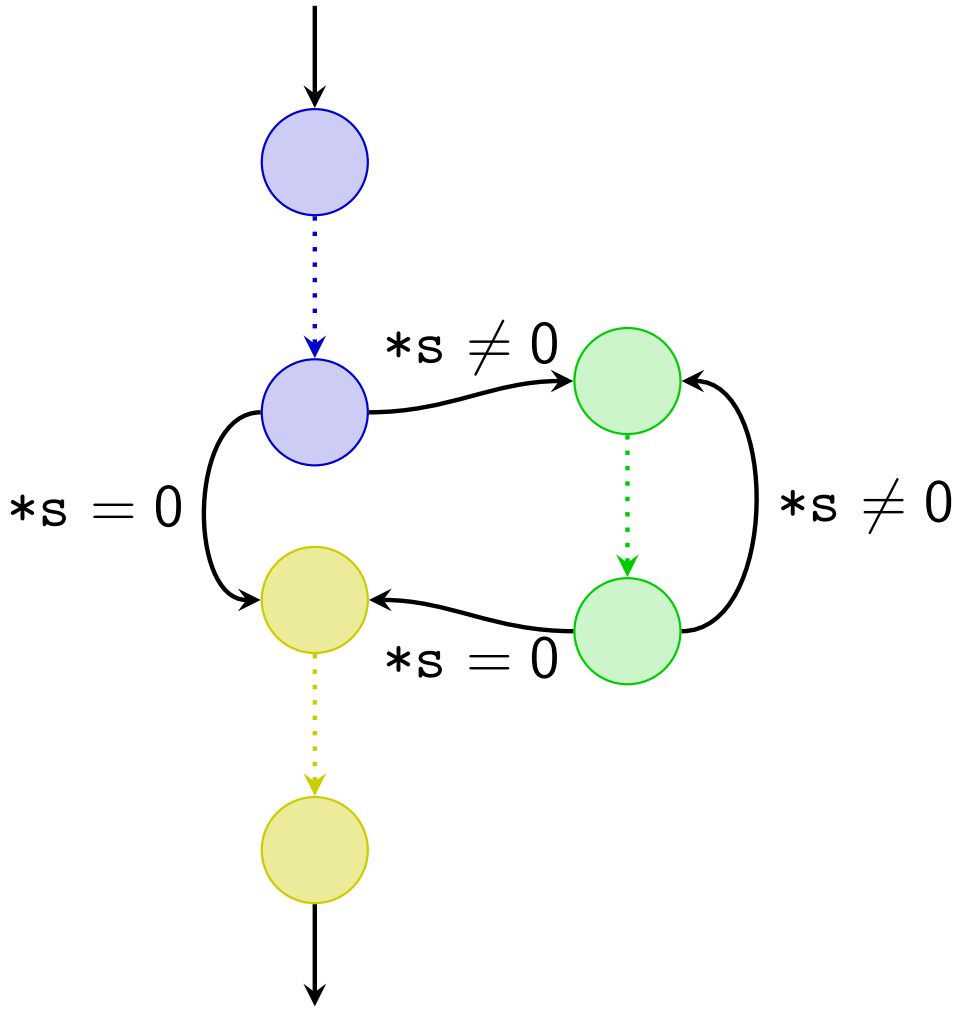
# Merging of Abstract States

```
int strlen(char *str) {
    char *s = str;
    while(*s) s++;
    return s-str;
}
```

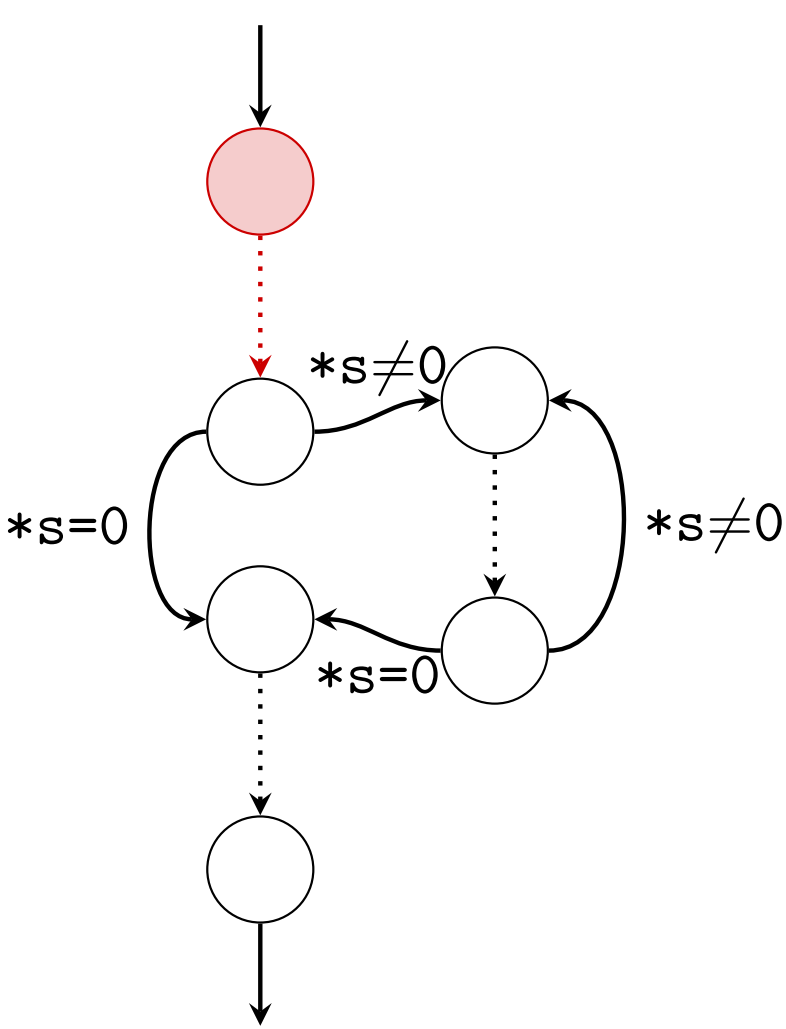


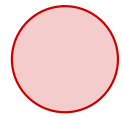
# Merging of Abstract States

```
int strlen(char *str) {
    char *s = str;
    while(*s) s++;
    return s-str;
}
```

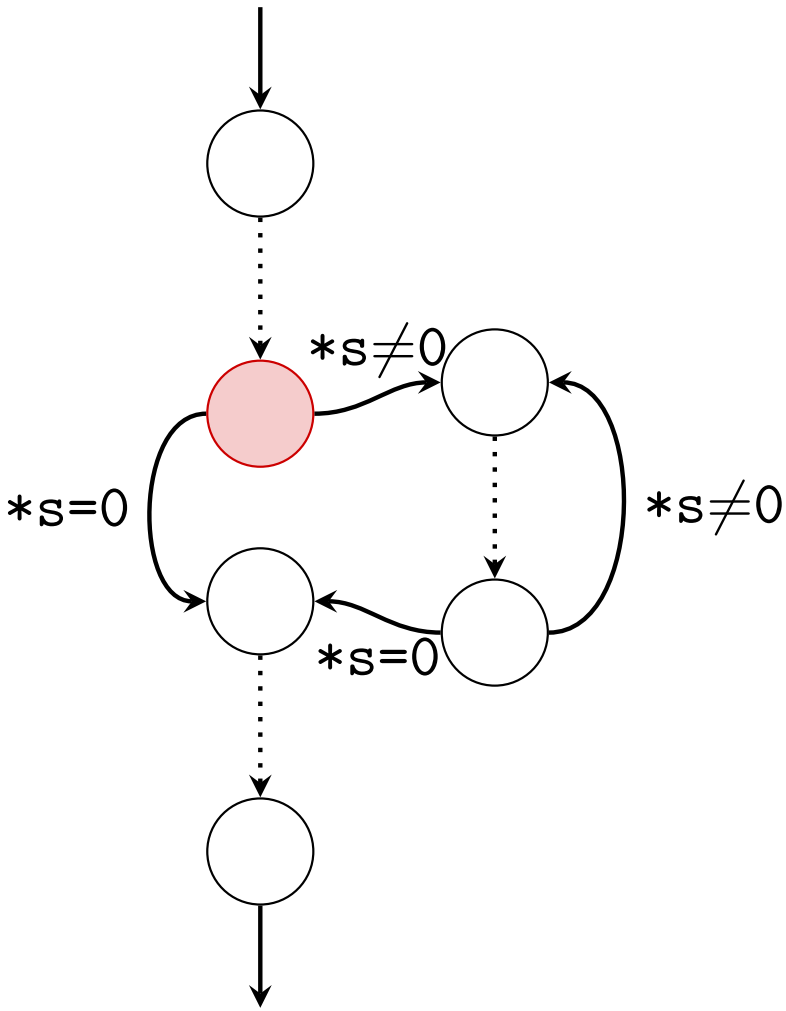


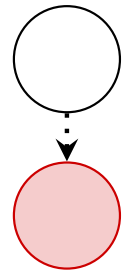
# Merging of Abstract States



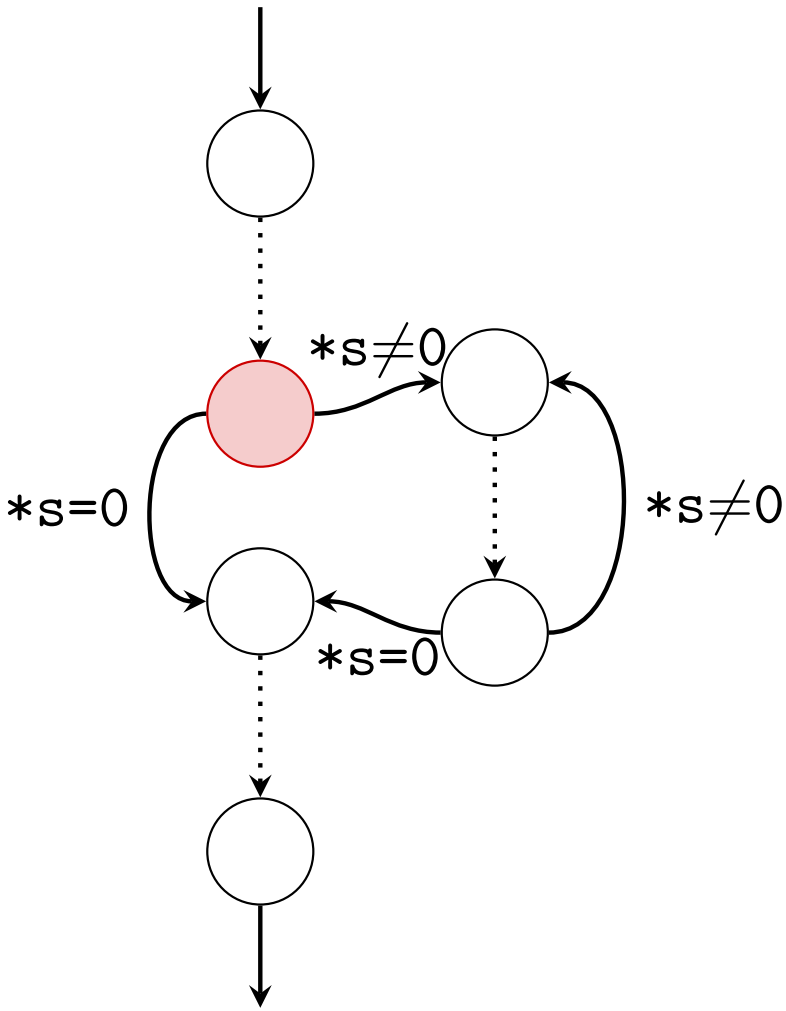



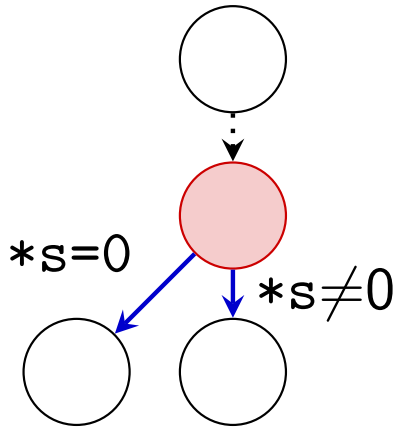
# Merging of Abstract States



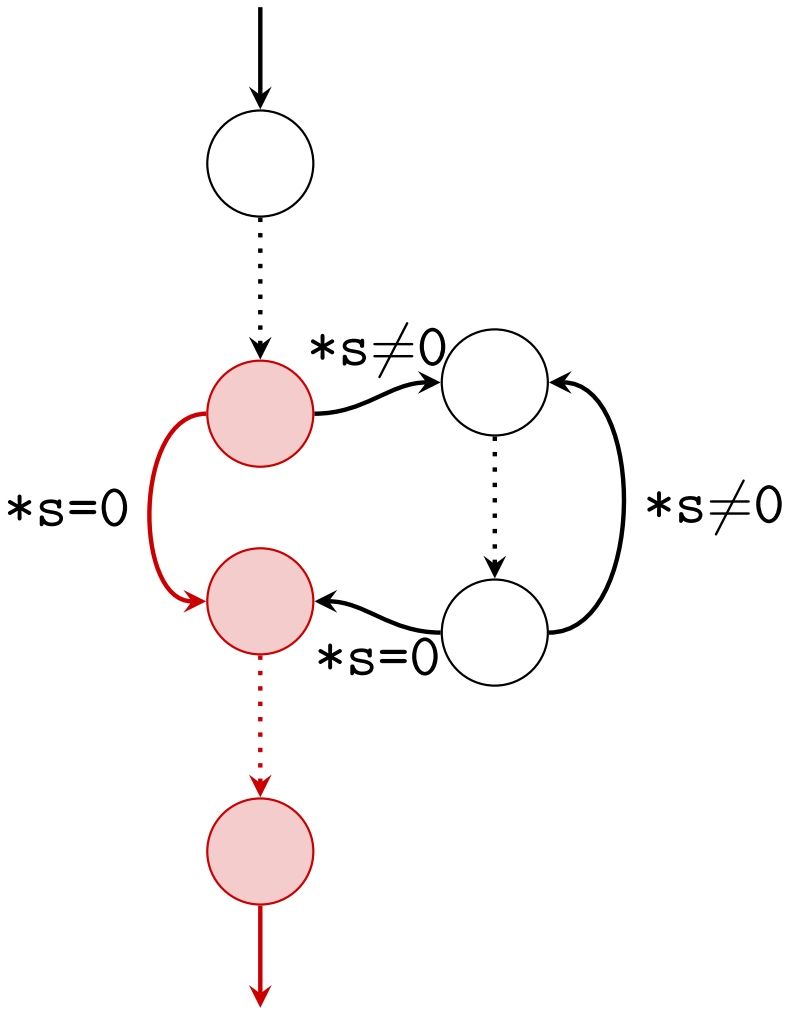



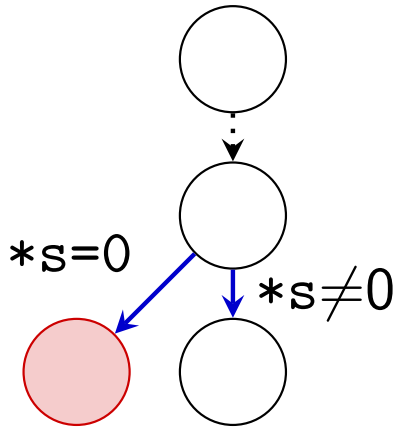
# Merging of Abstract States



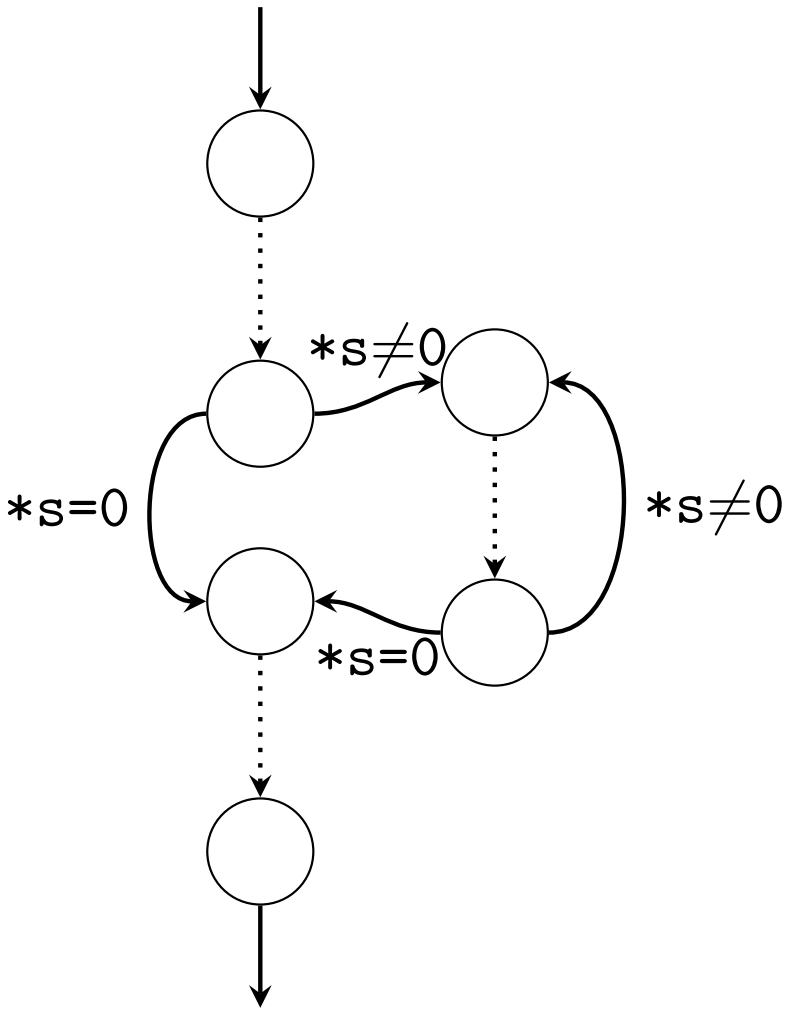



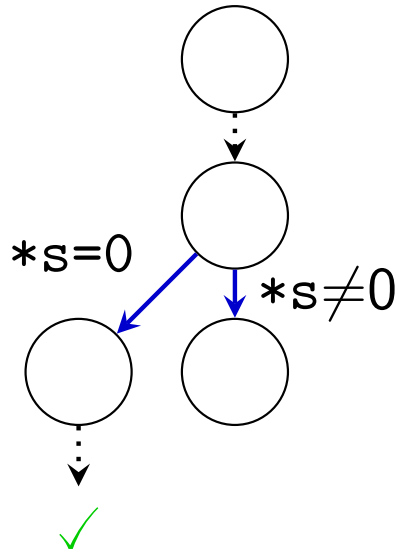
# Merging of Abstract States



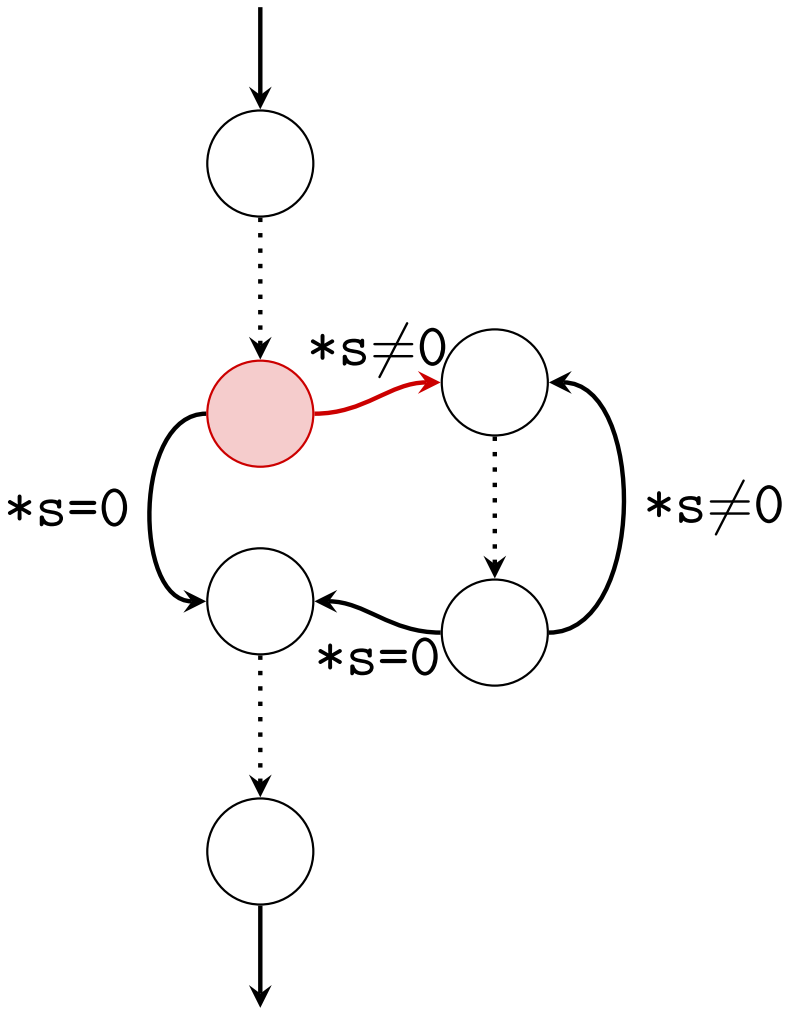



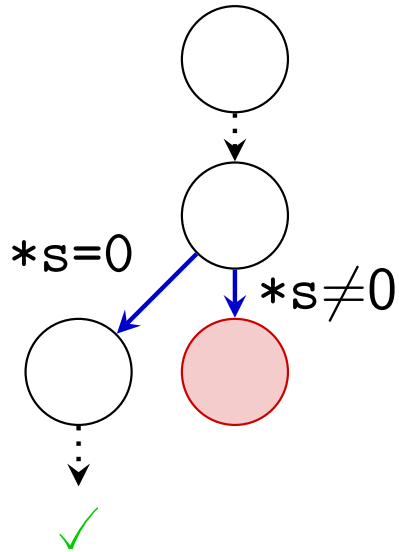
# Merging of Abstract States



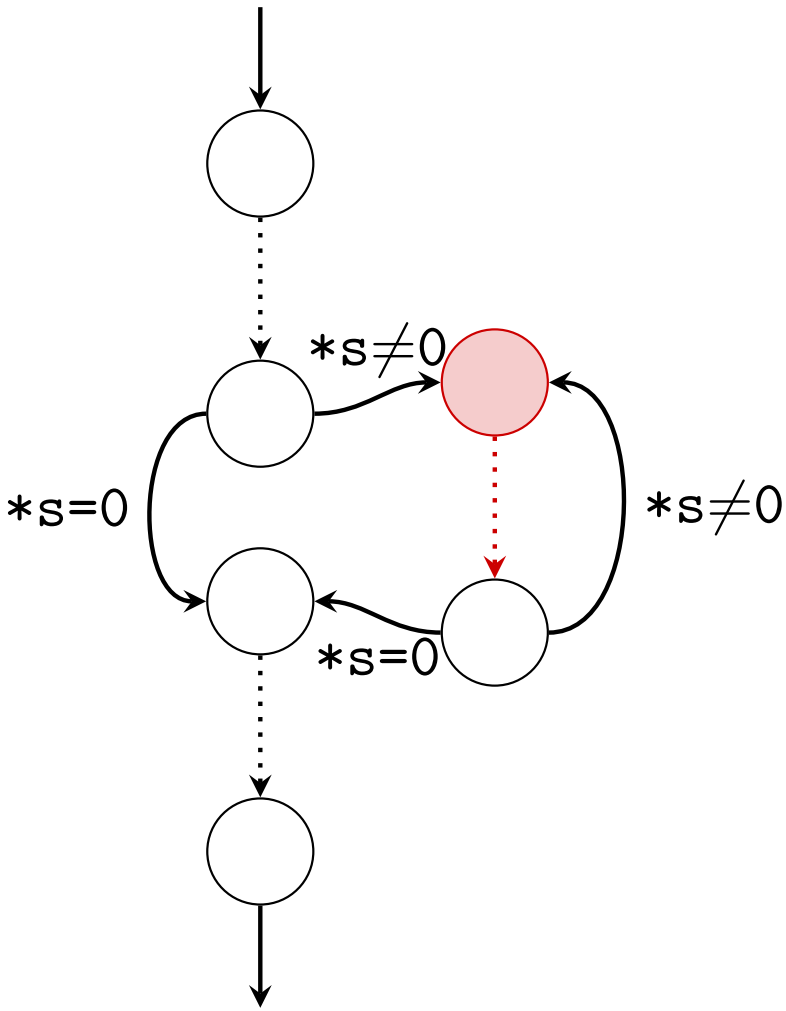



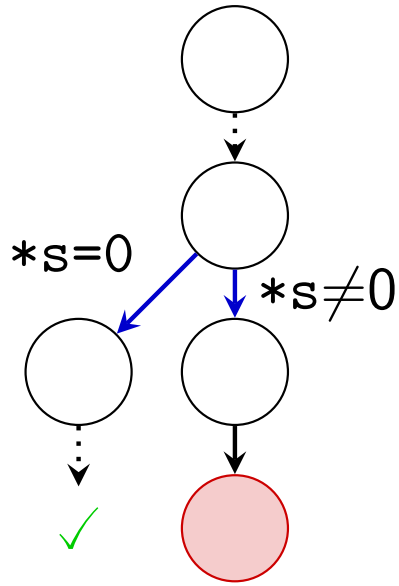
# Merging of Abstract States



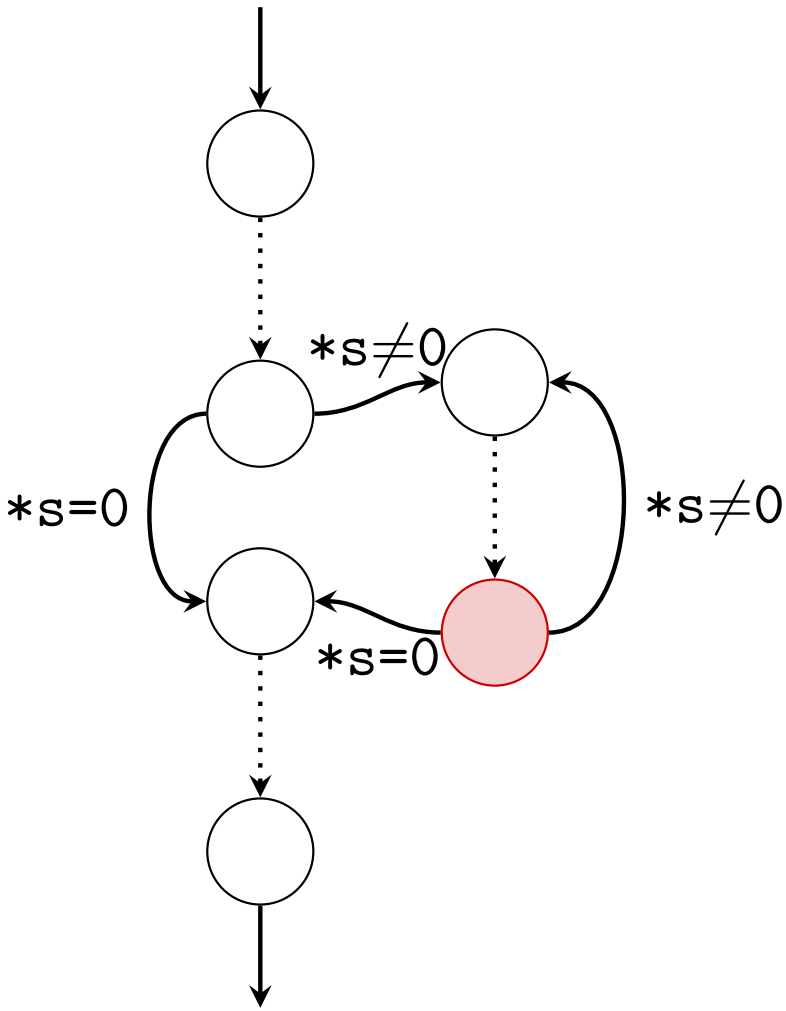



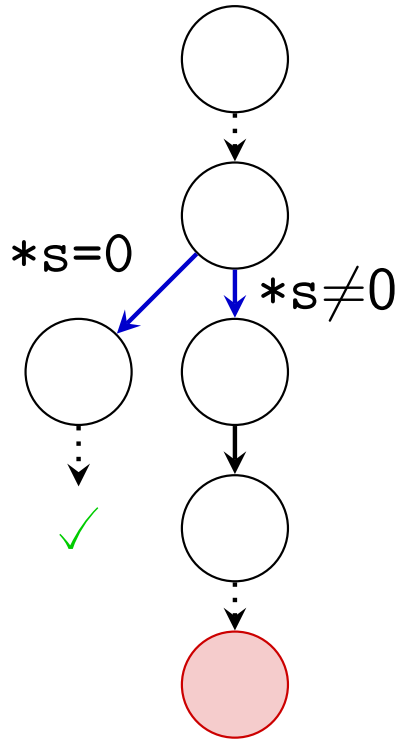
# Merging of Abstract States



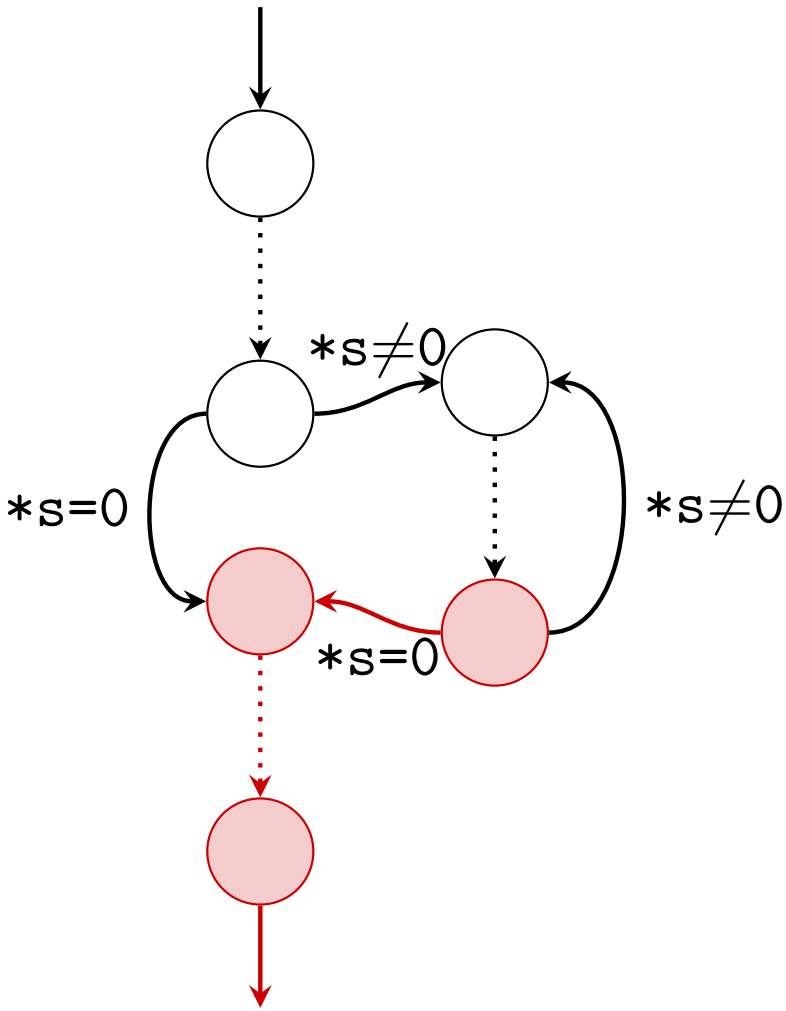



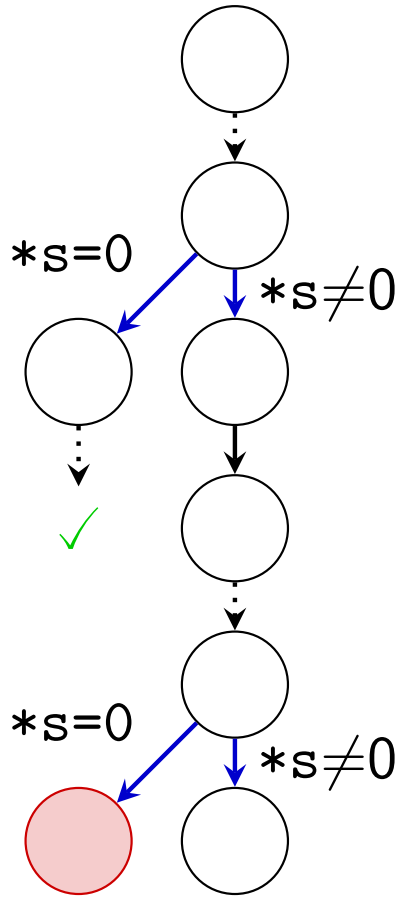
# Merging of Abstract States



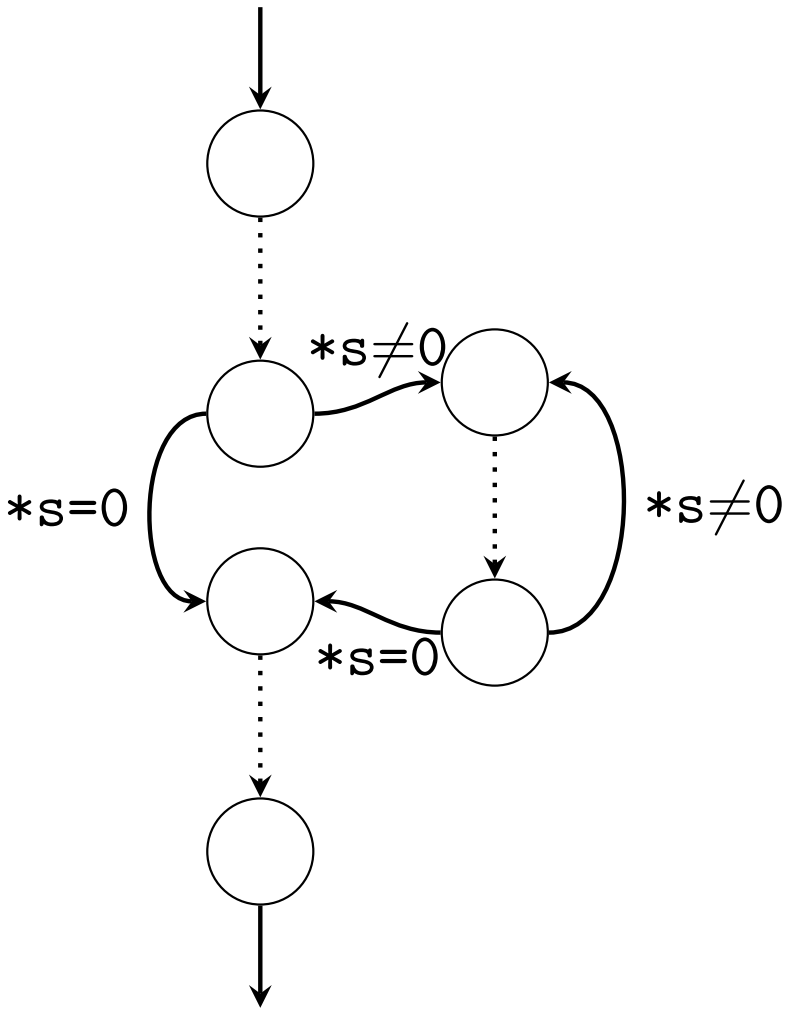



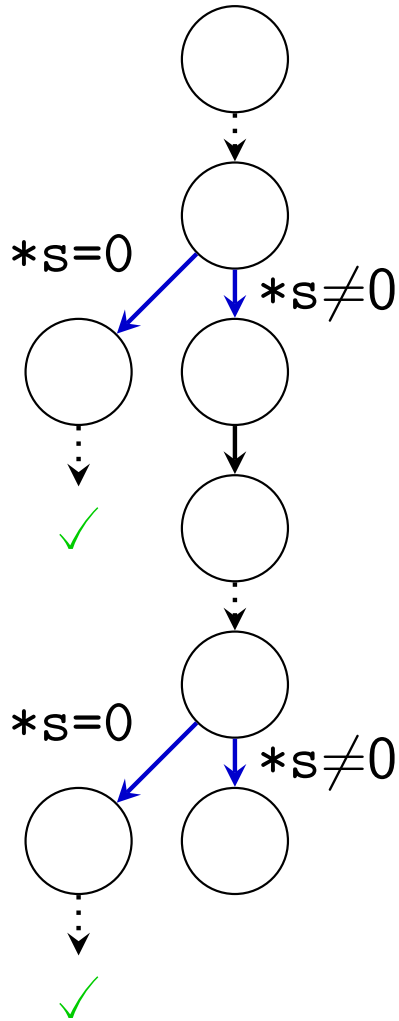
# Merging of Abstract States



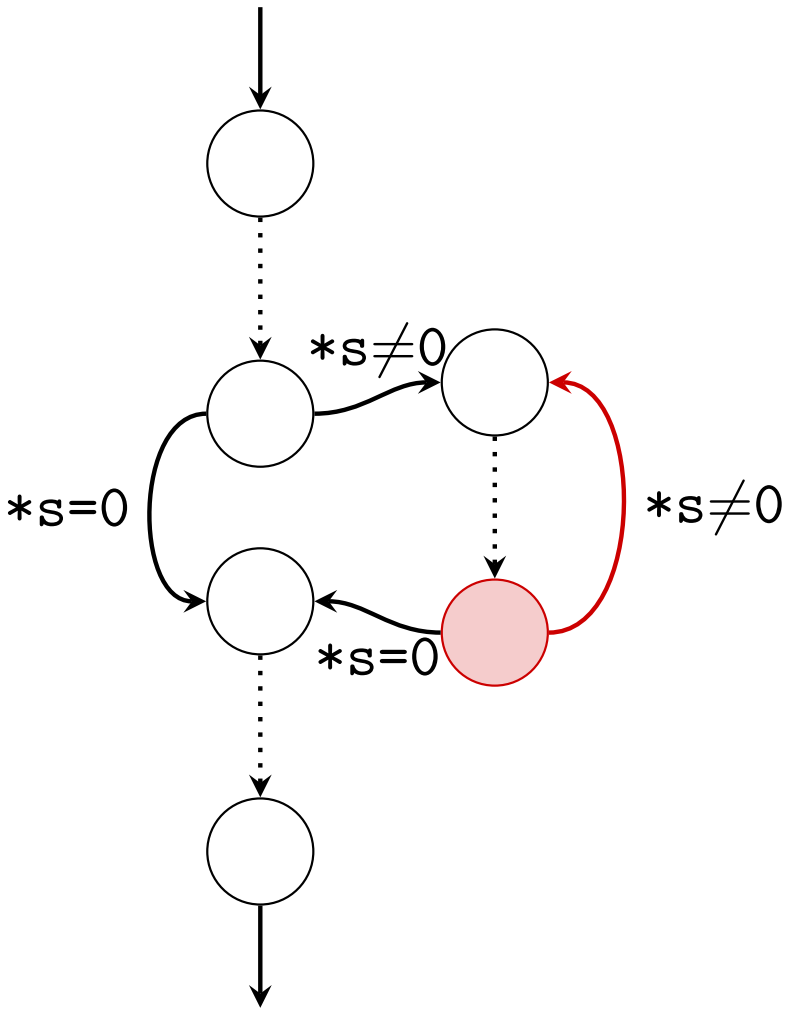



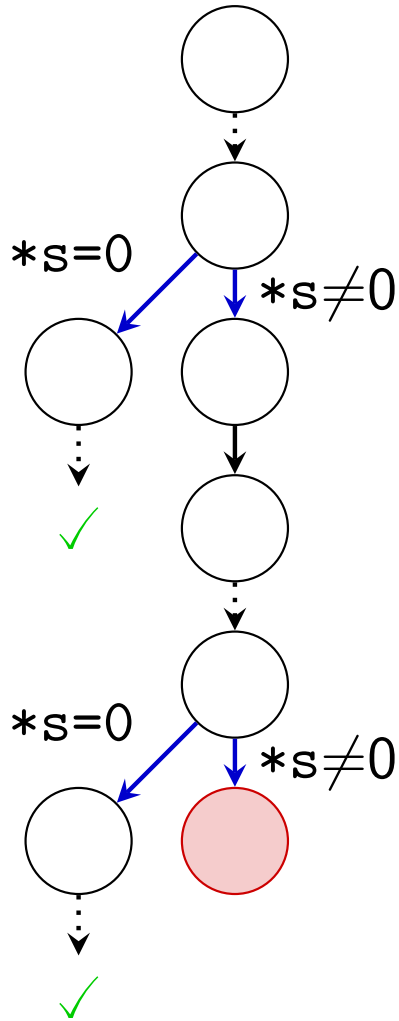
# Merging of Abstract States



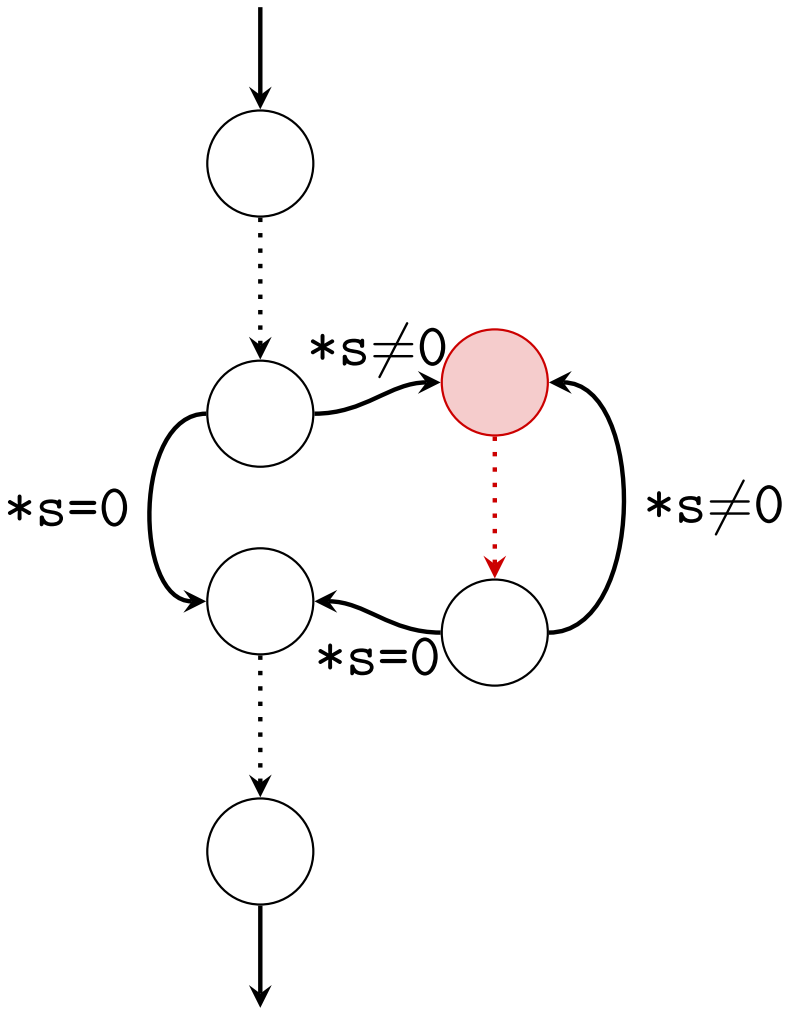



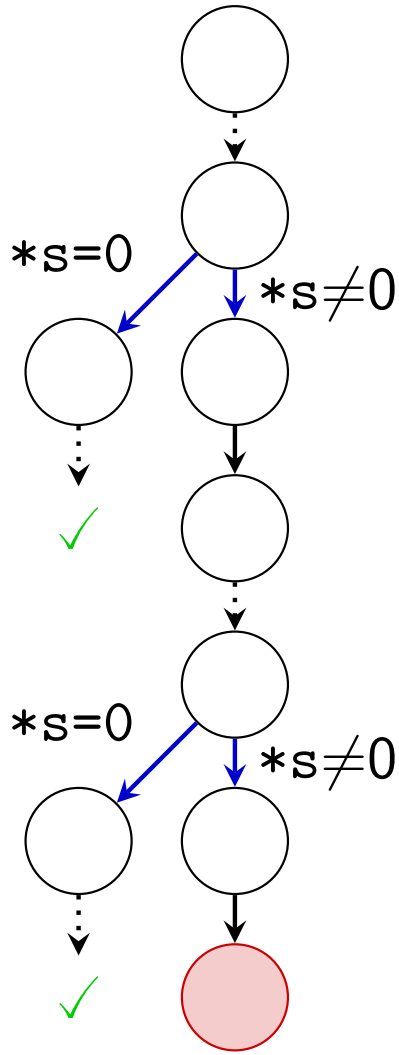
# Merging of Abstract States



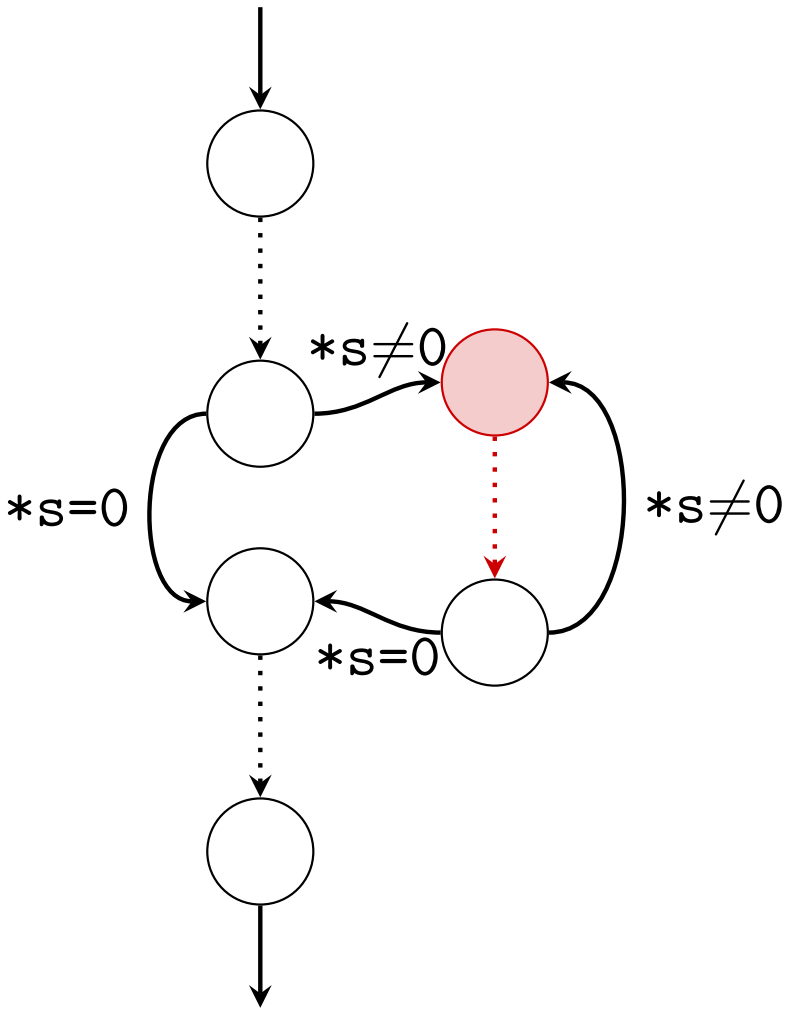



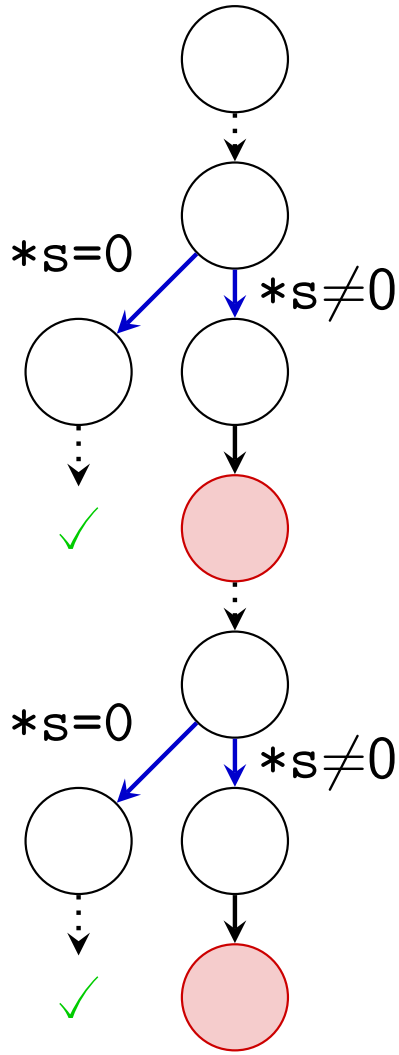
# Merging of Abstract States



# Merging of Abstract States



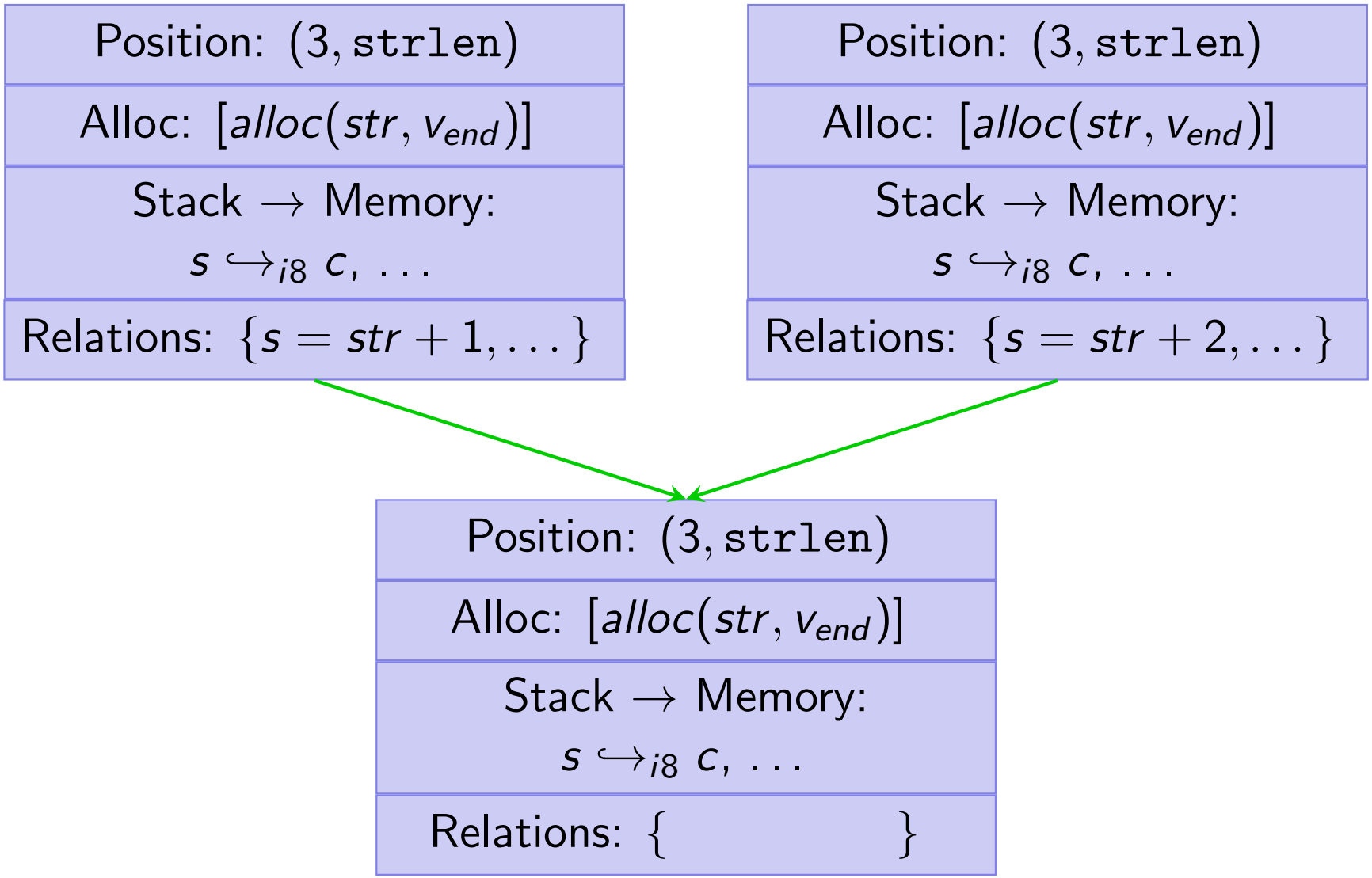



# Merging of Abstract States

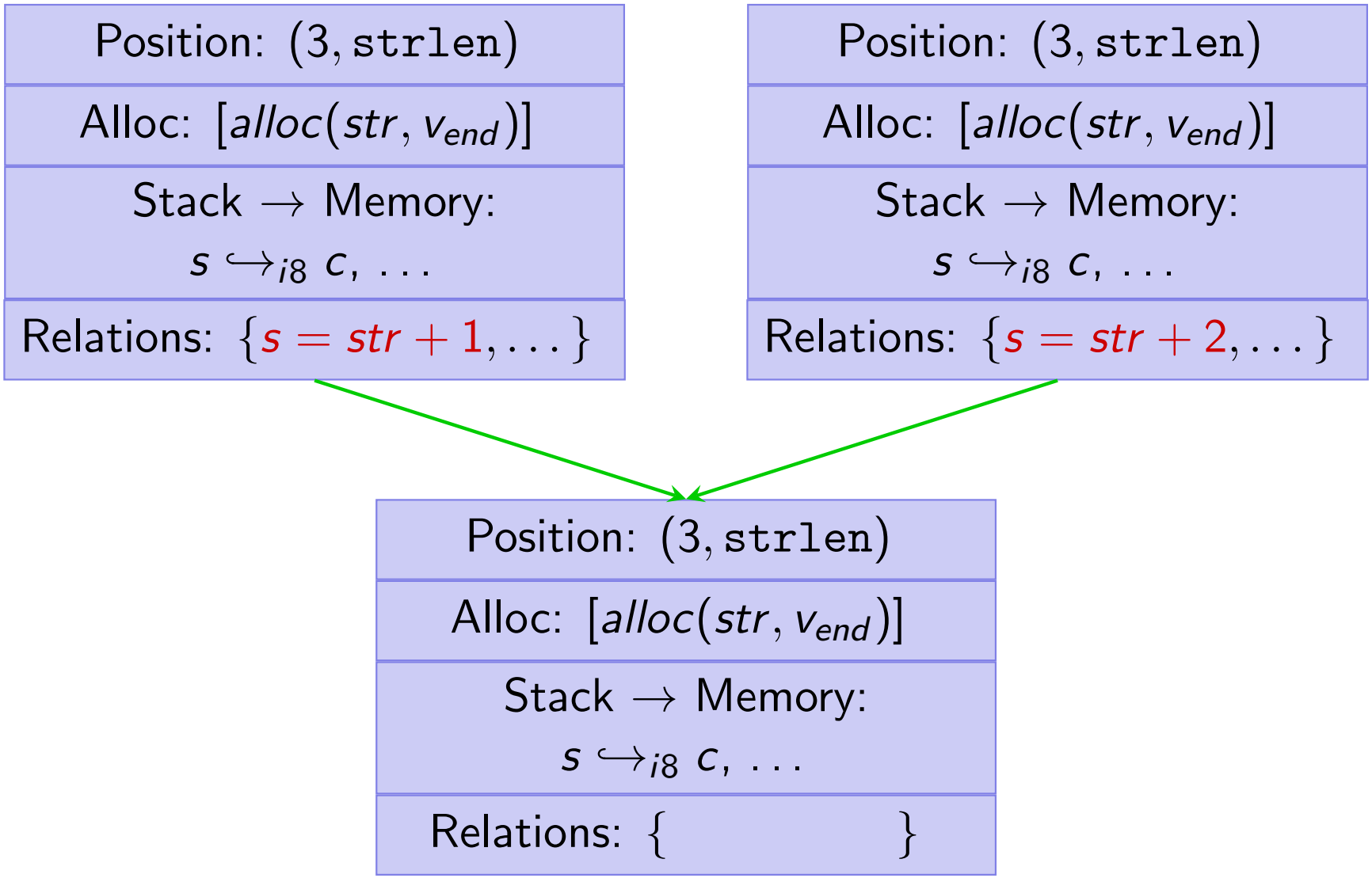
| Position: $(3, \texttt{strlen})$ |
| --- |
| Alloc: $[alloc(str, v_{end})]$ |
| Stack $\rightarrow$ Memory: $s \hookrightarrow_{i8} c, \ldots$ |
| Relations: $\{s = str + 1, \ldots\}$ |

| Position: $(3, \texttt{strlen})$ |
| --- |
| Alloc: $[alloc(str, v_{end})]$ |
| Stack $\rightarrow$ Memory: $s \hookrightarrow_{i8} c, \ldots$ |
| Relations: $\{s = str + 2, \ldots\}$ |

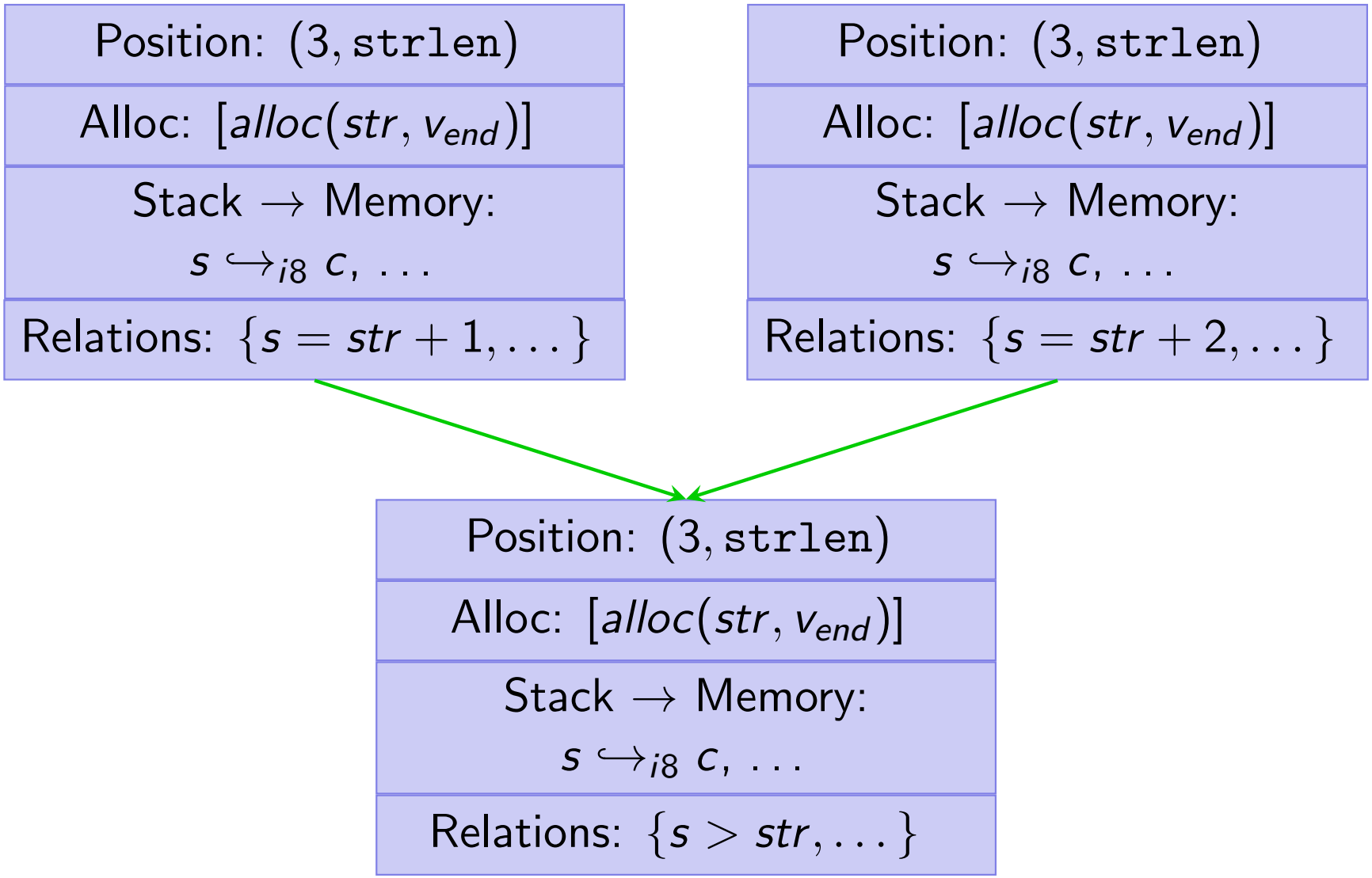
# Merging of Abstract States



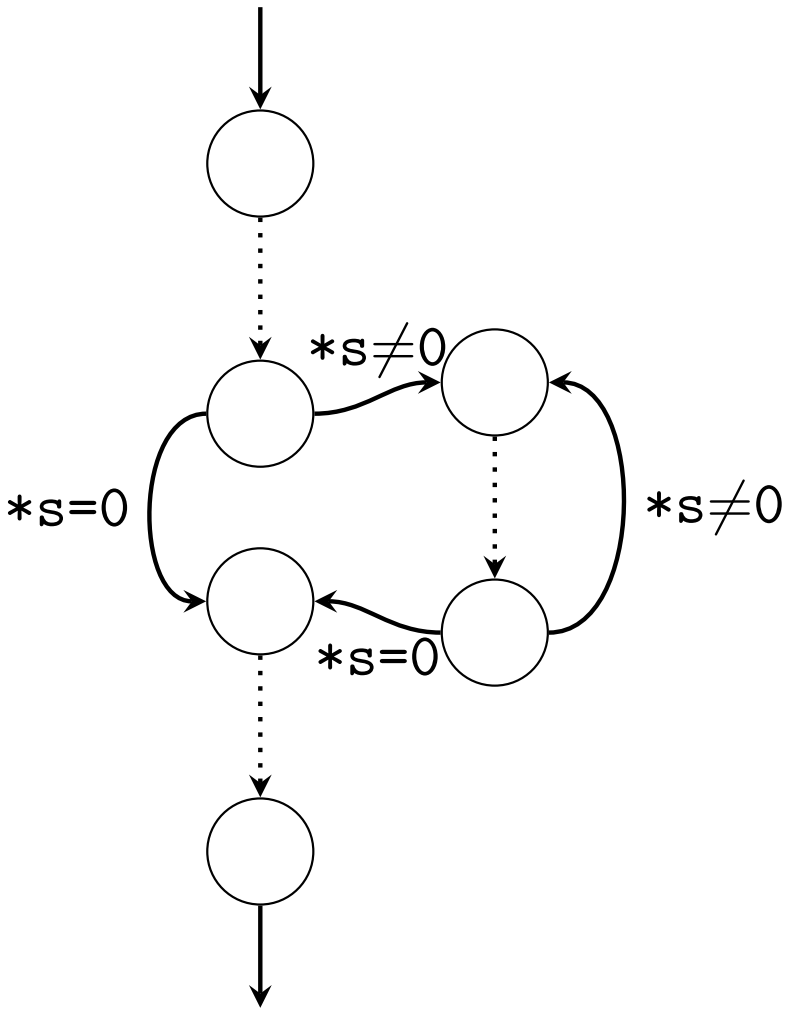
# Merging of Abstract States

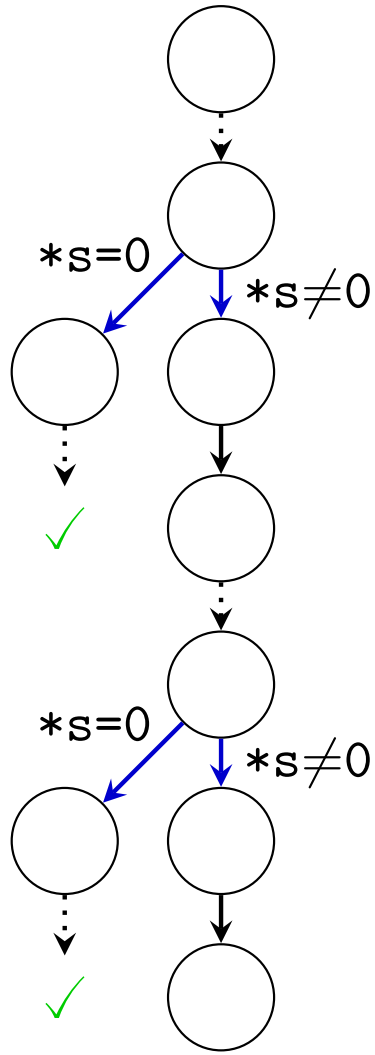


# Merging of Abstract States

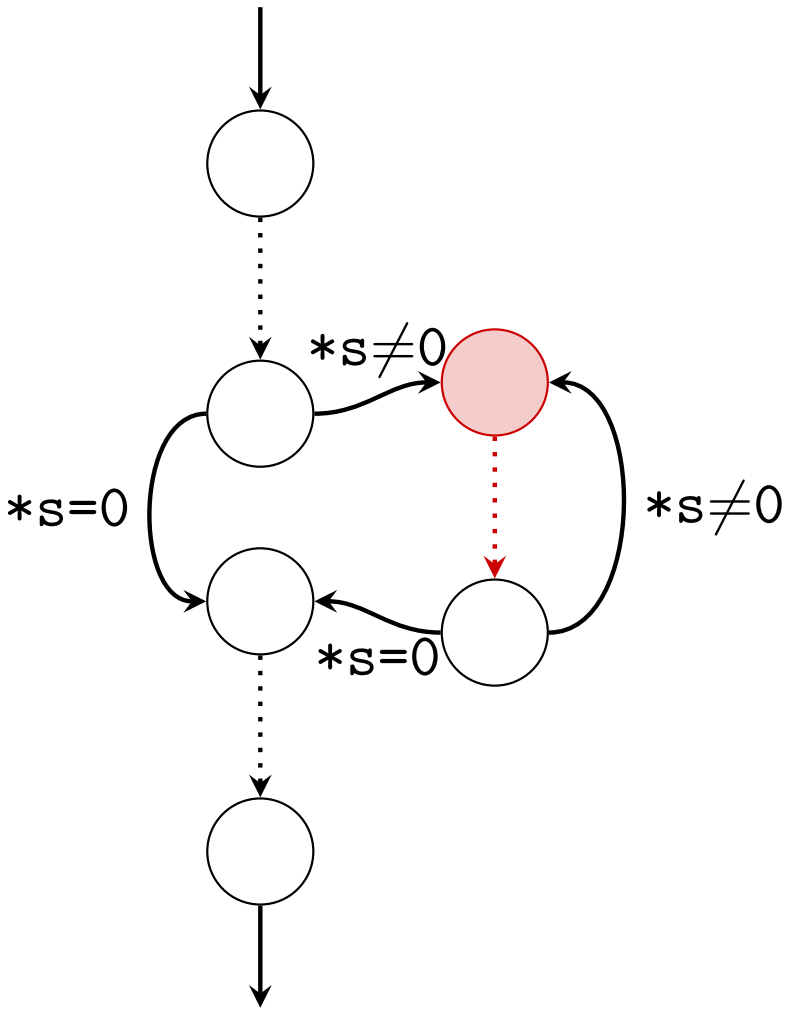


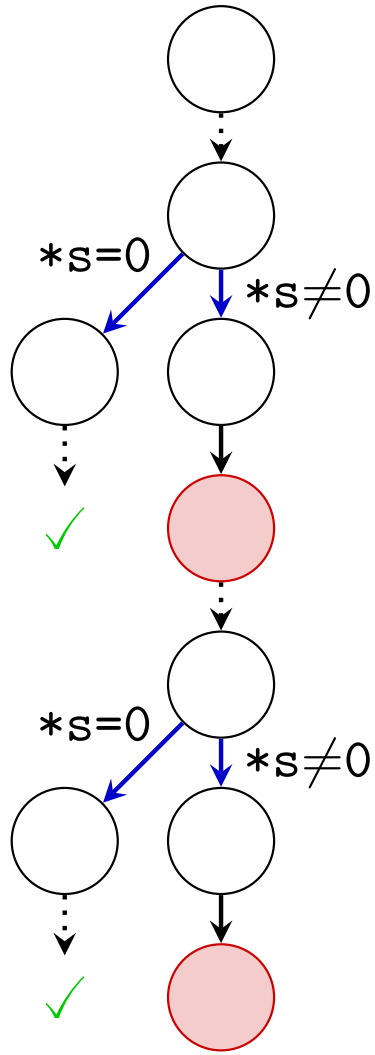
# Merging of Abstract States



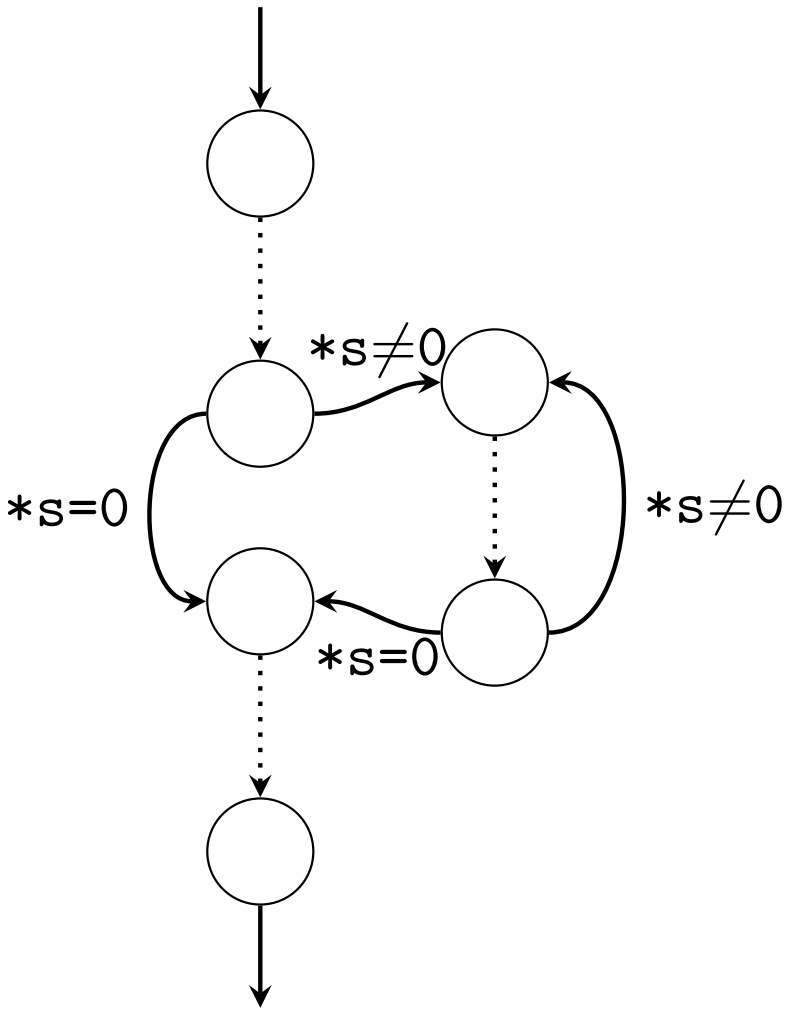



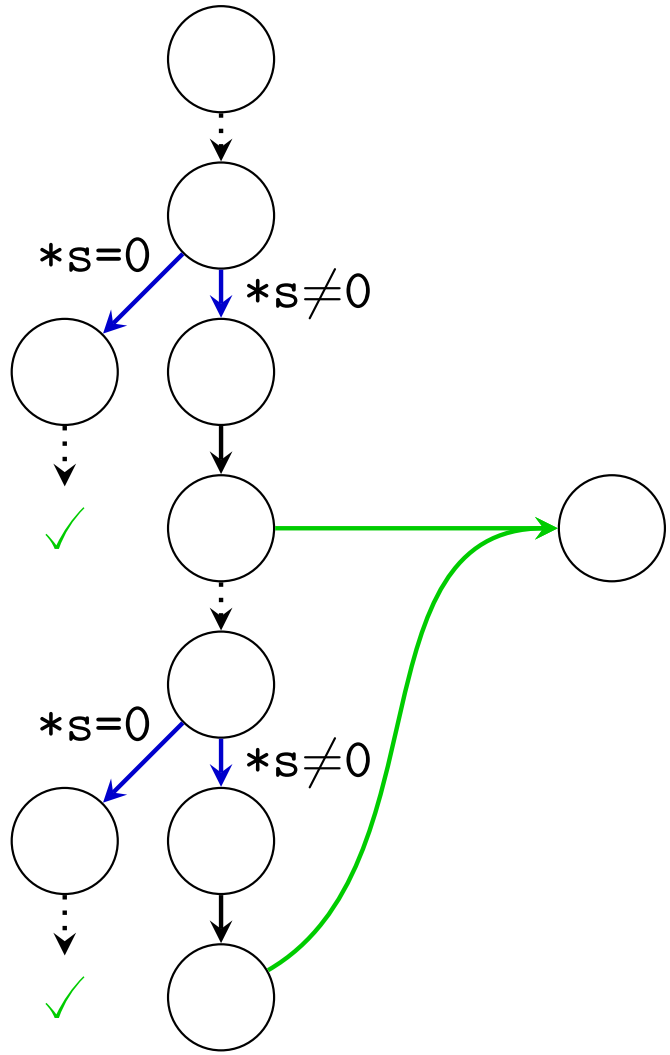
# Merging of Abstract States



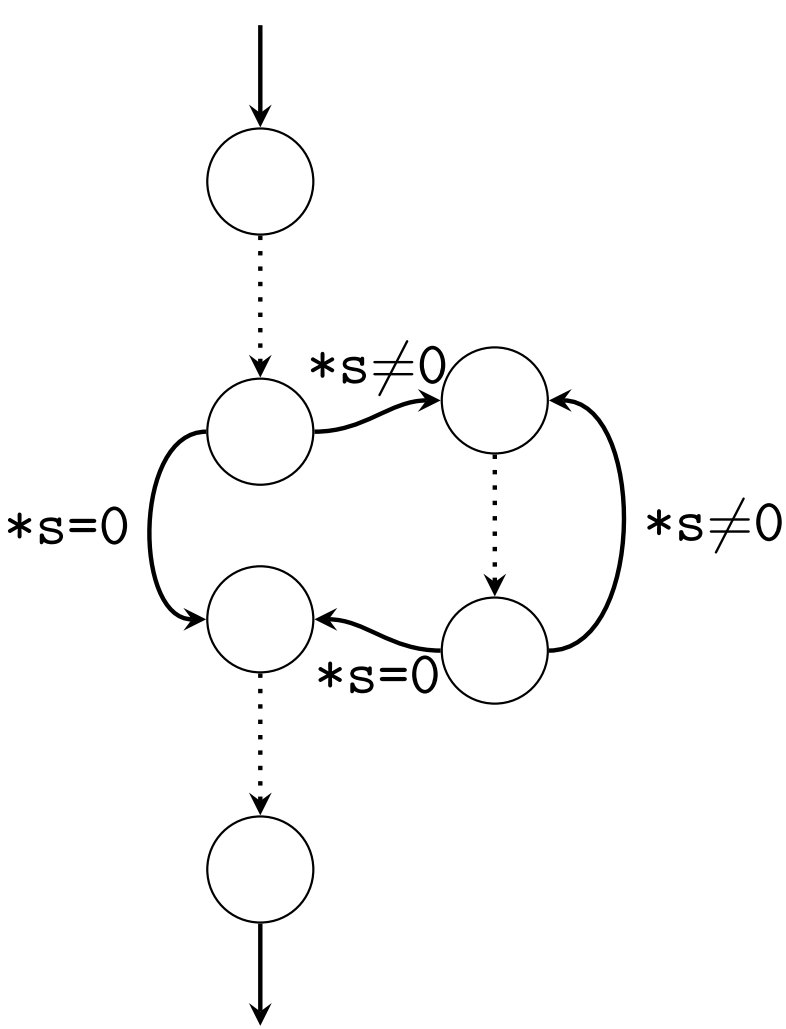



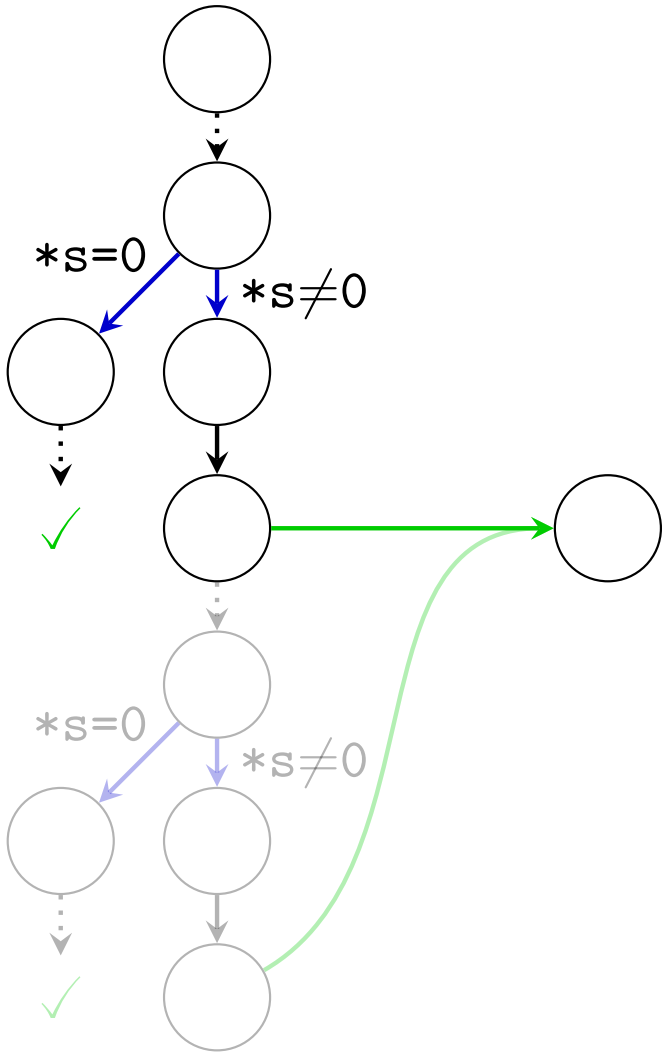
# Merging of Abstract States



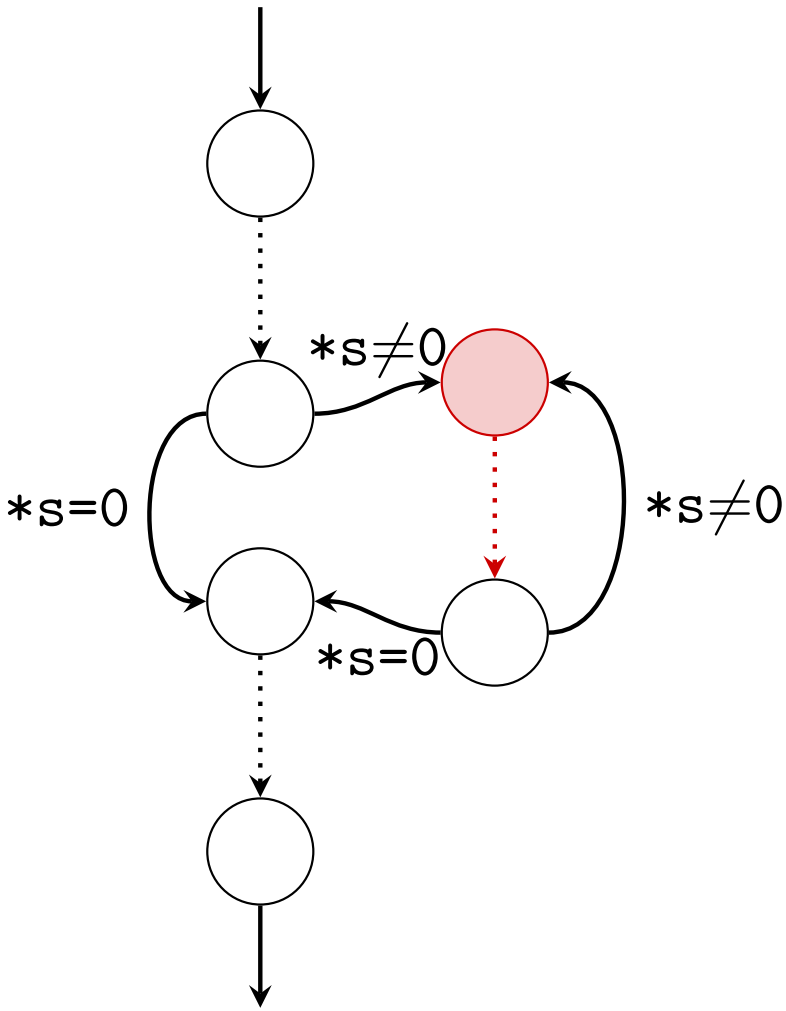



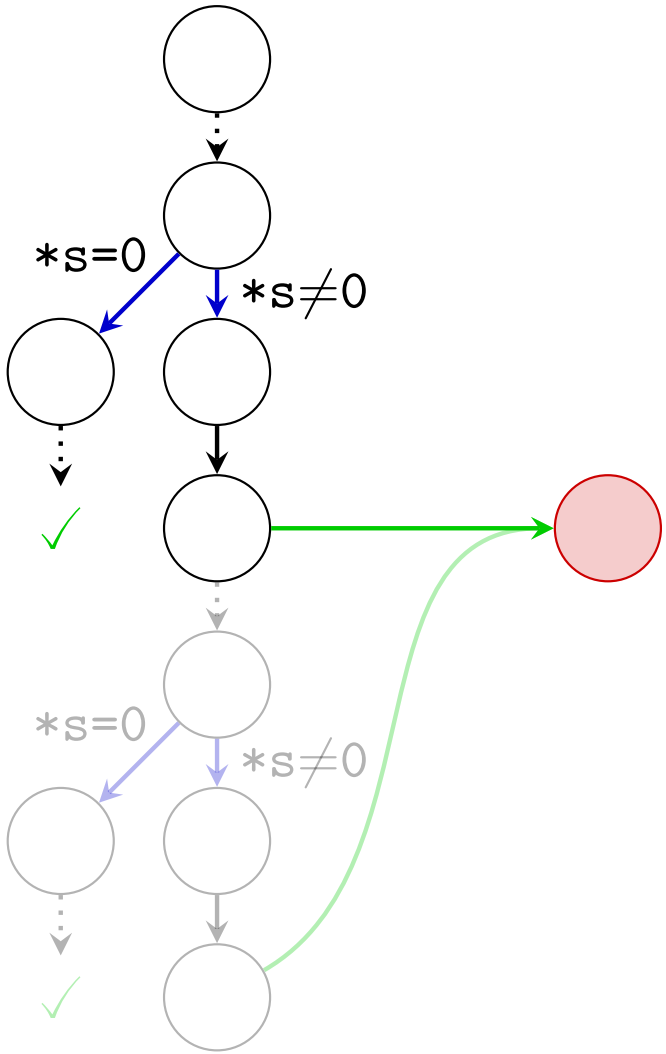
# Merging of Abstract States



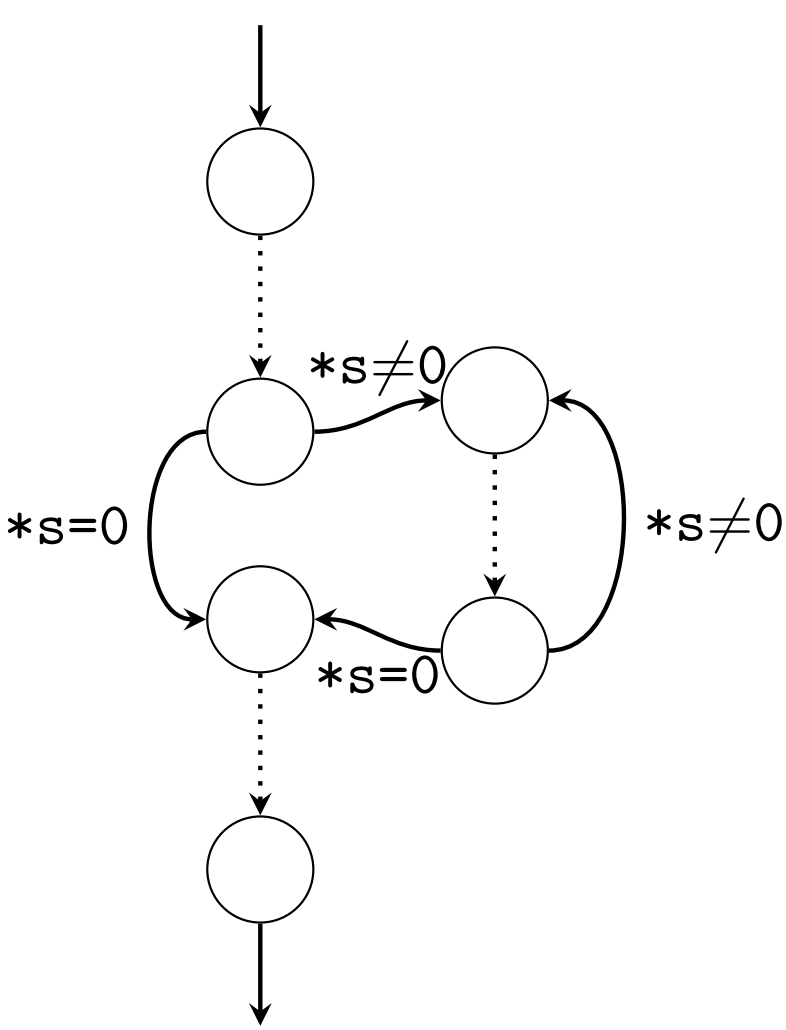



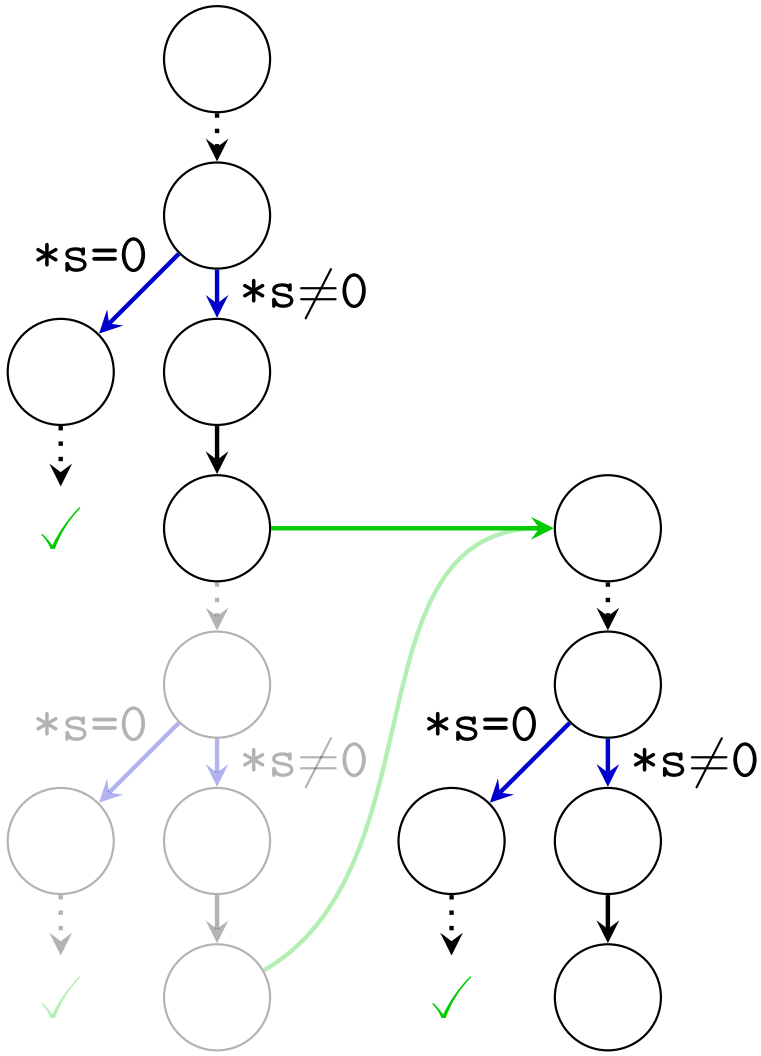
# Merging of Abstract States



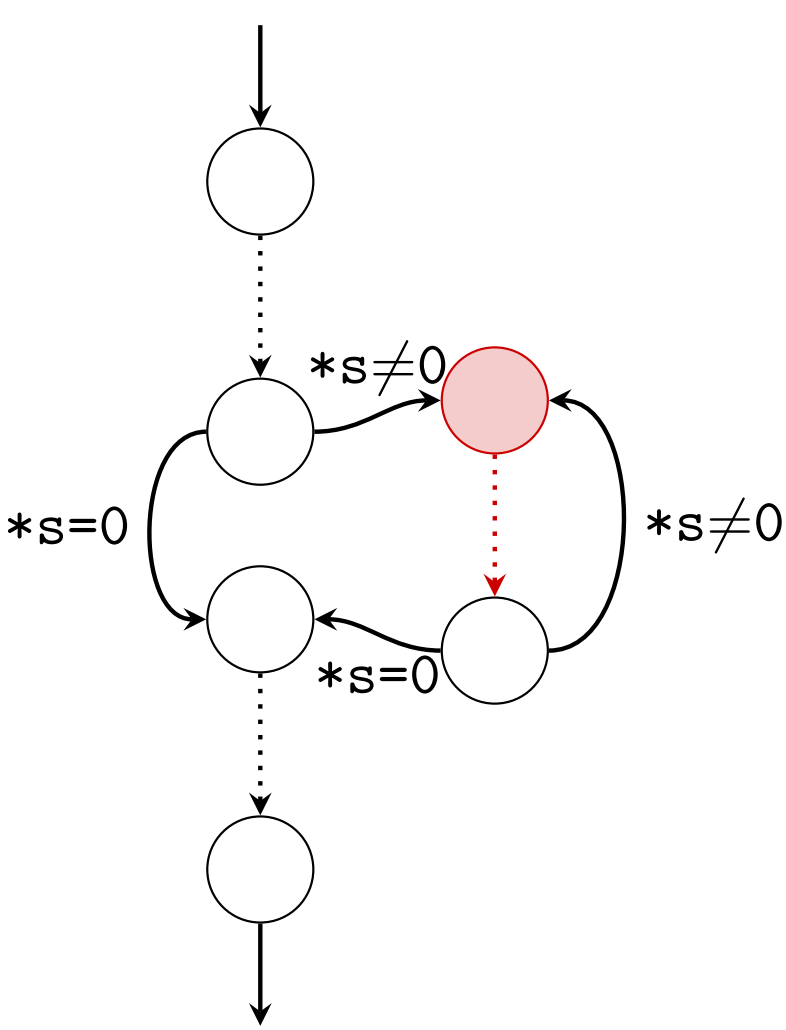



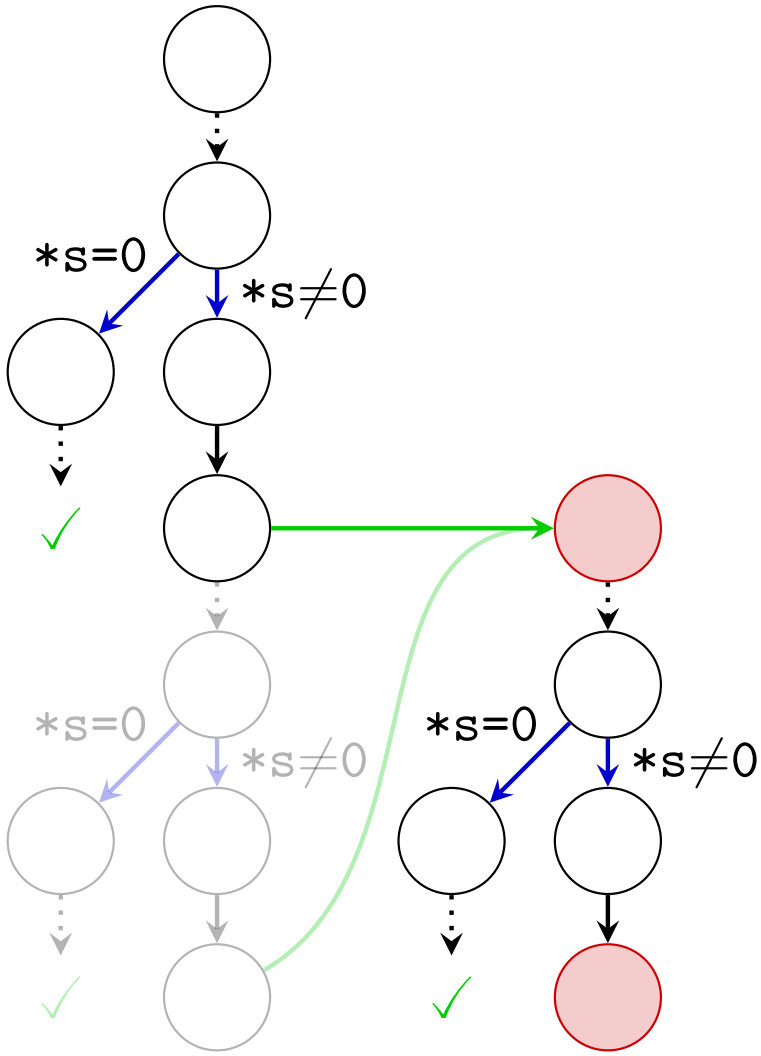
# Merging of Abstract States



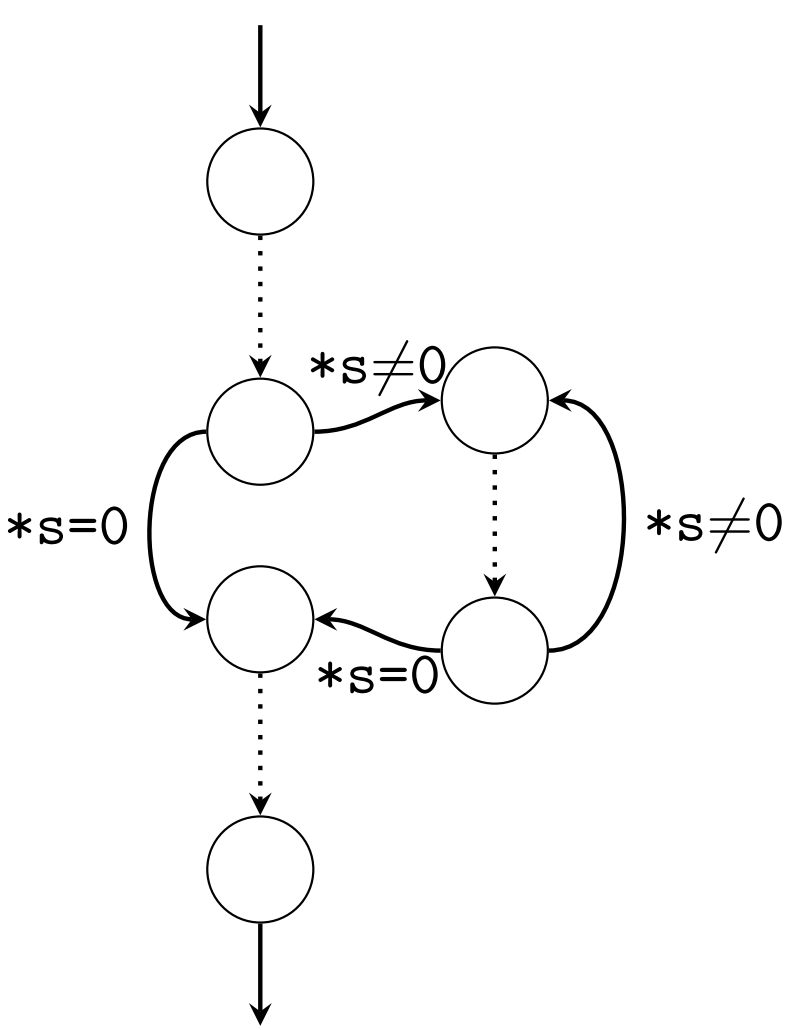



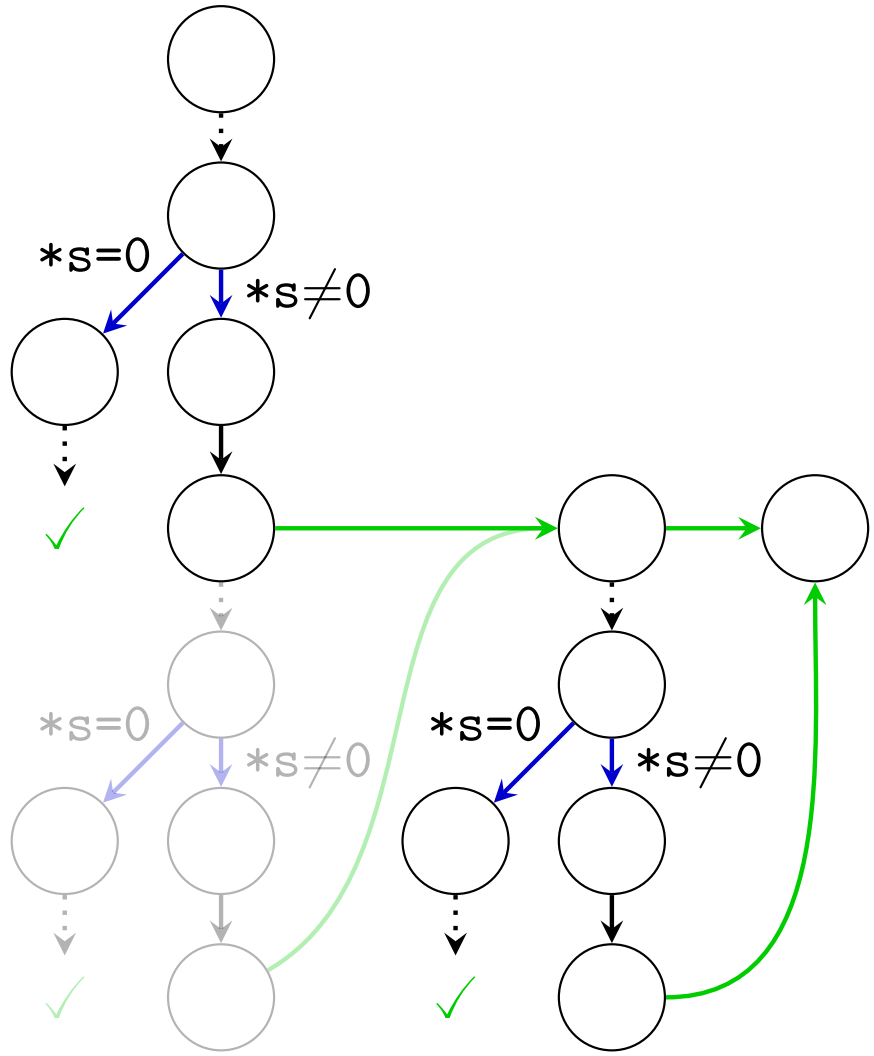
# Merging of Abstract States



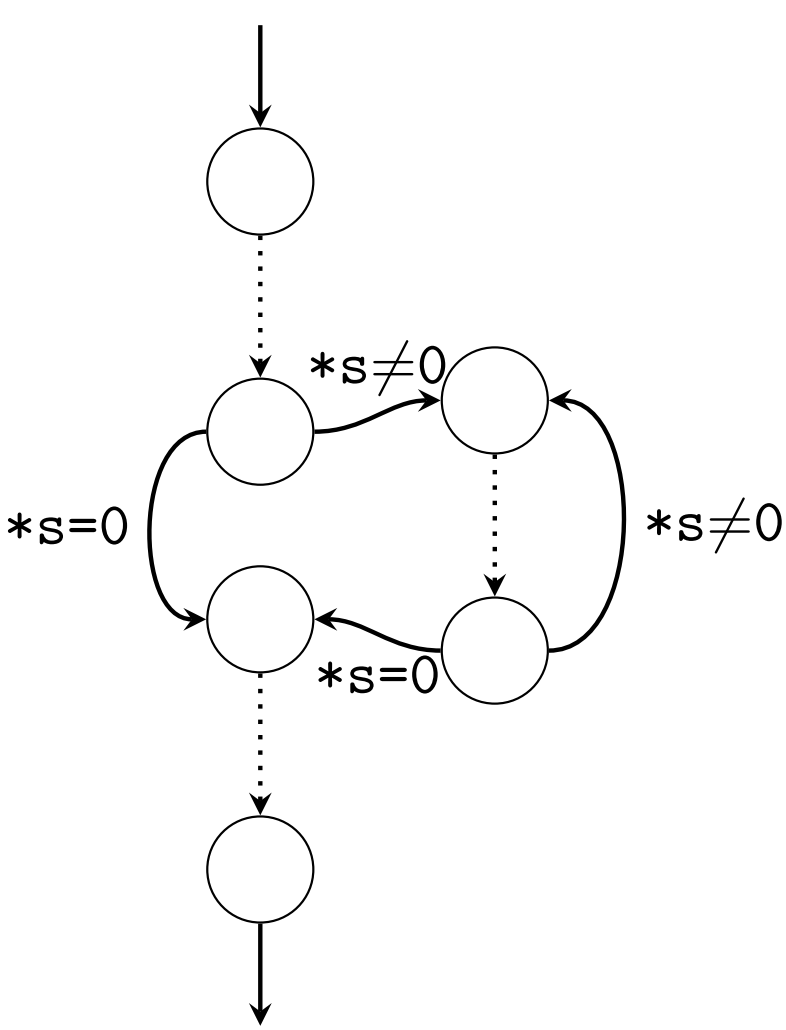



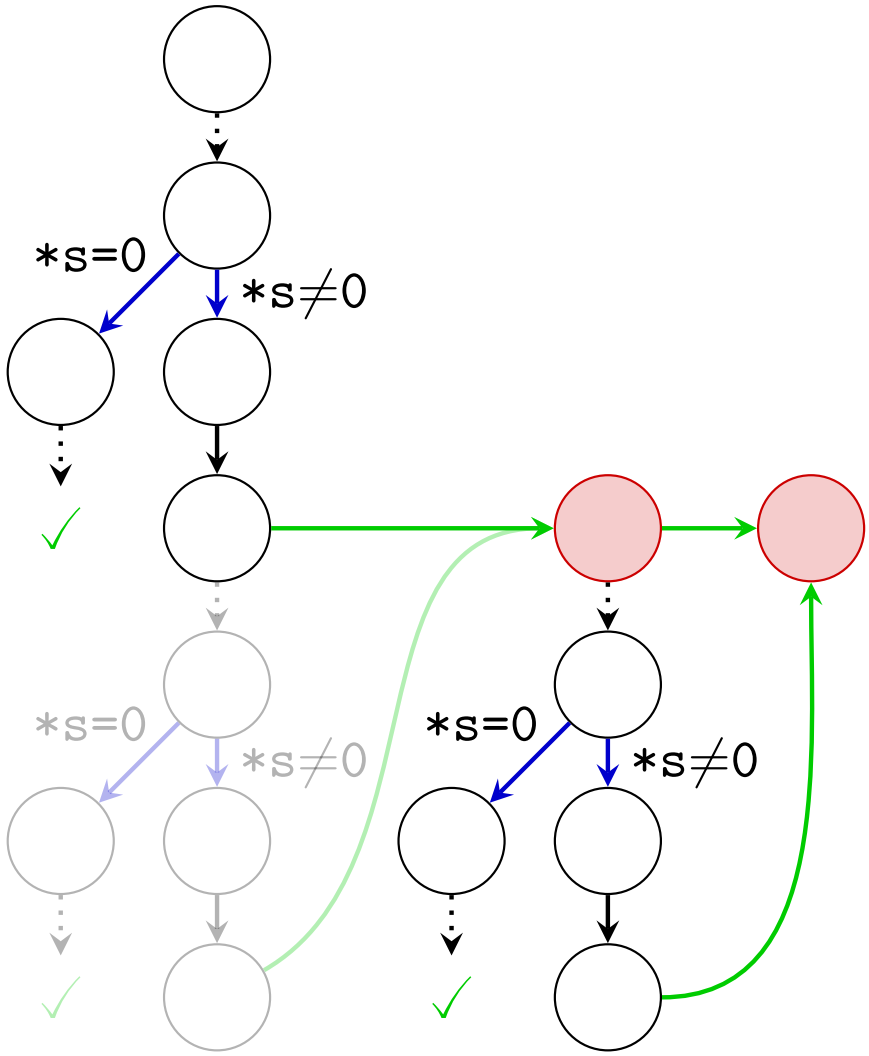
# Merging of Abstract States



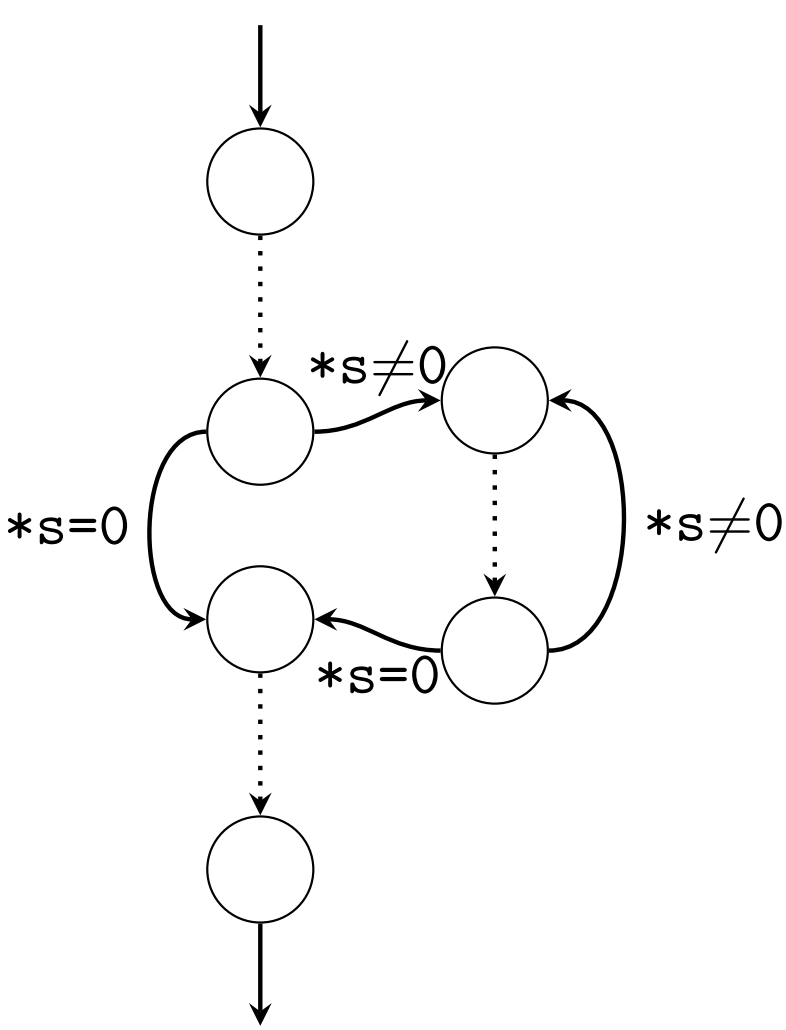



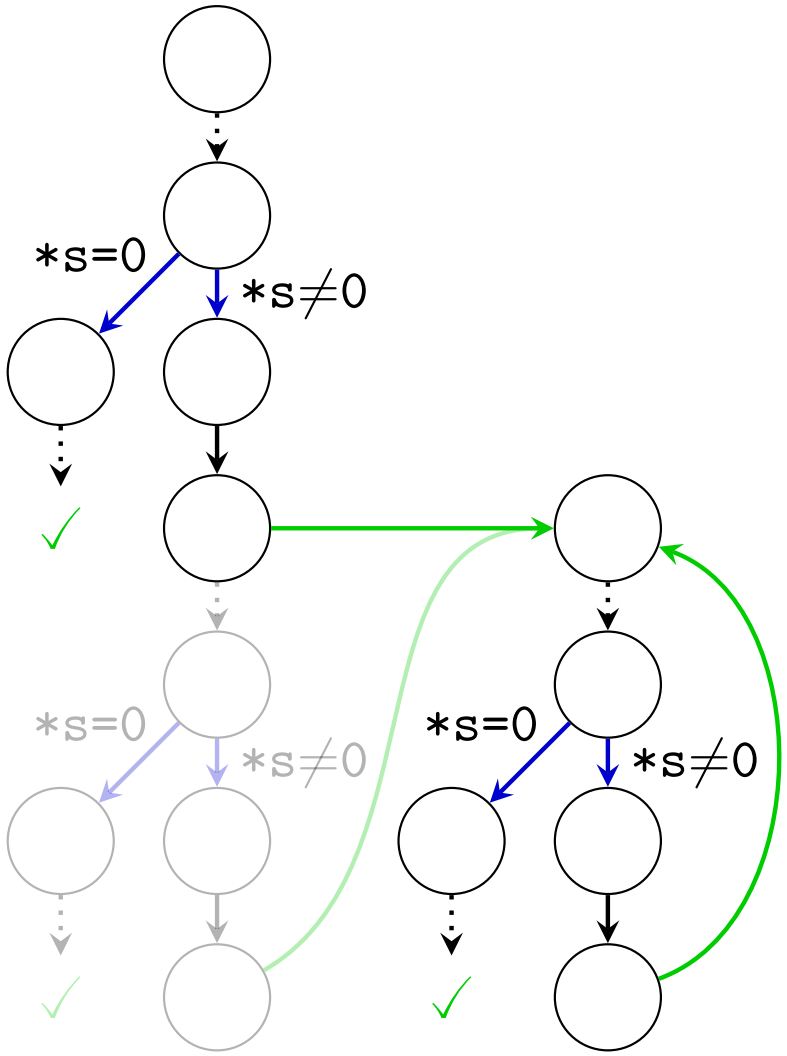
# Merging of Abstract States



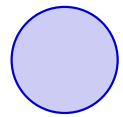



# Merging of Abstract States

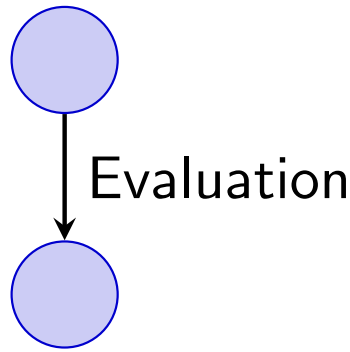


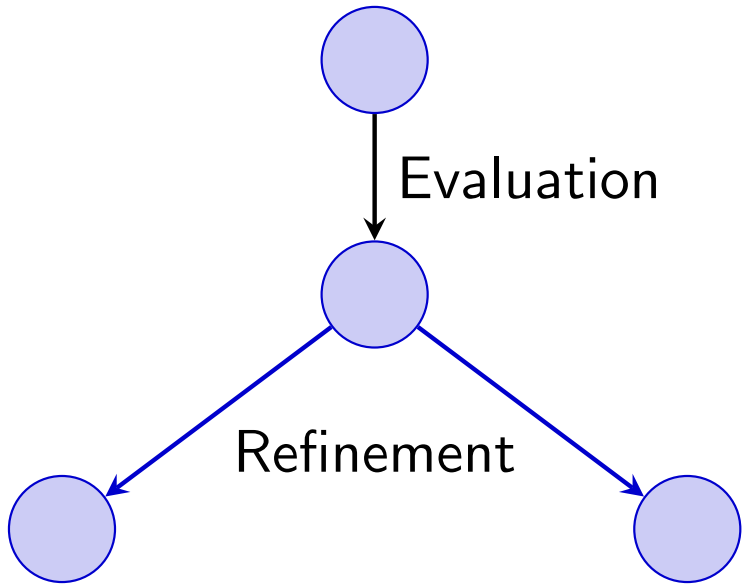
# Summary: Symbolic Execution Graphs



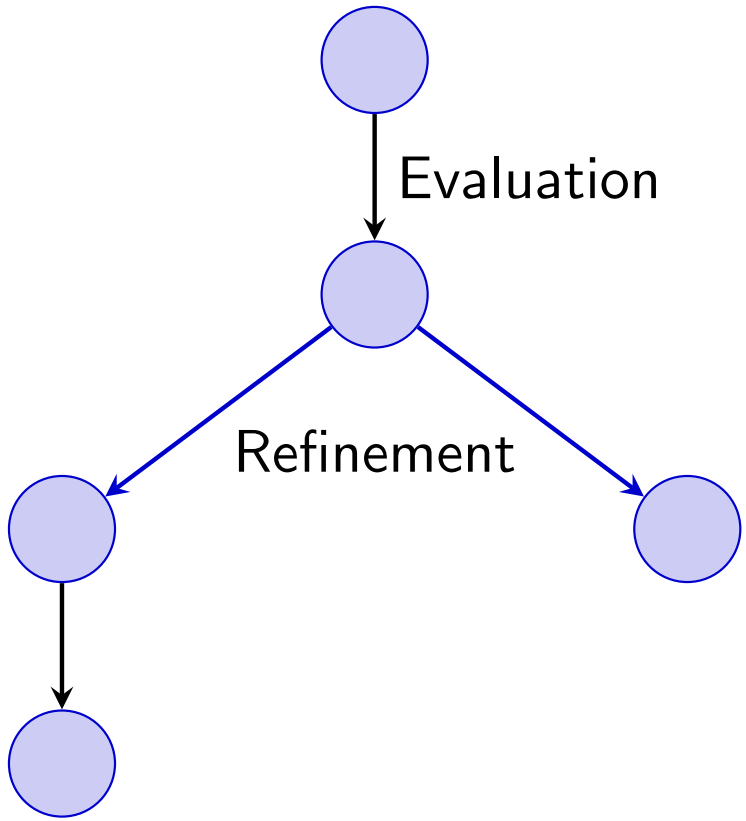
# Summary: Symbolic Execution Graphs



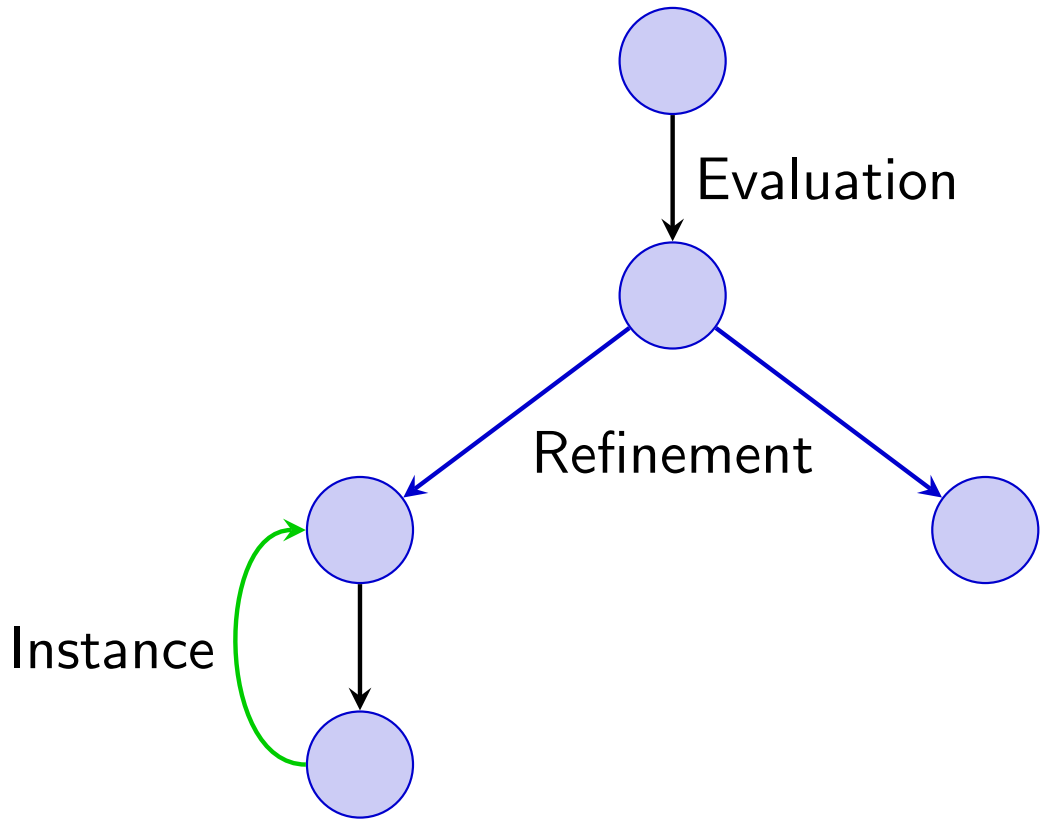
# Summary: Symbolic Execution Graphs



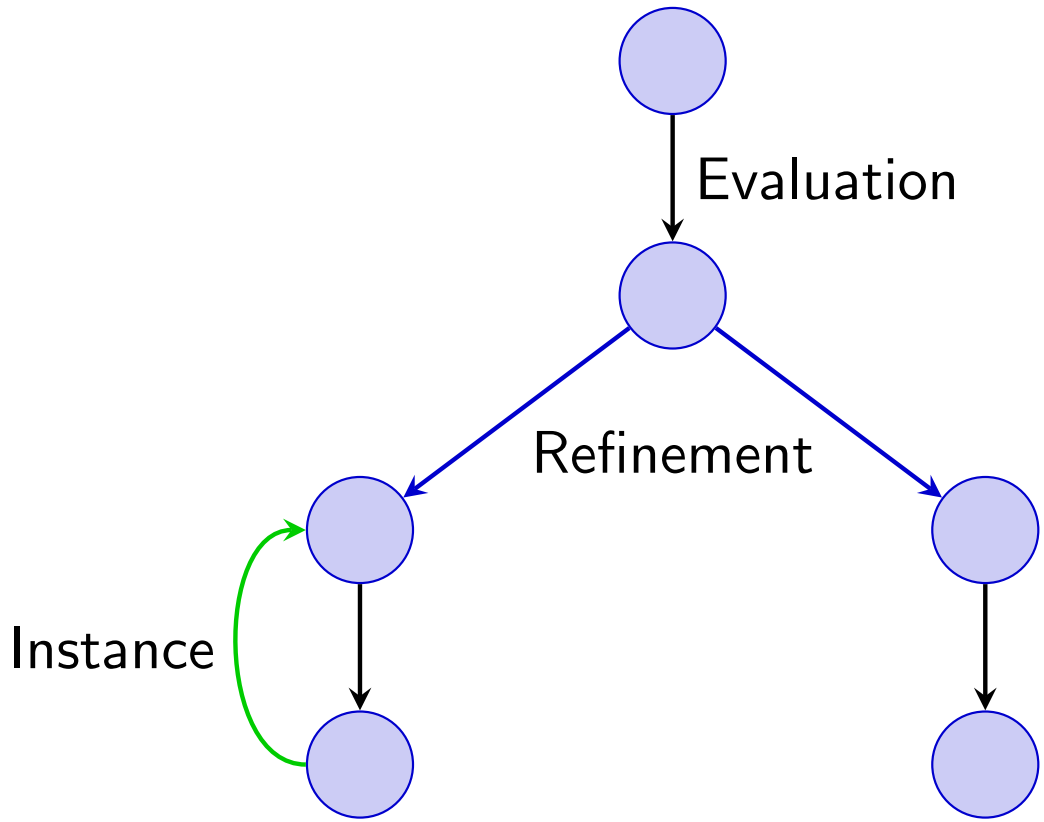
# Summary: Symbolic Execution Graphs



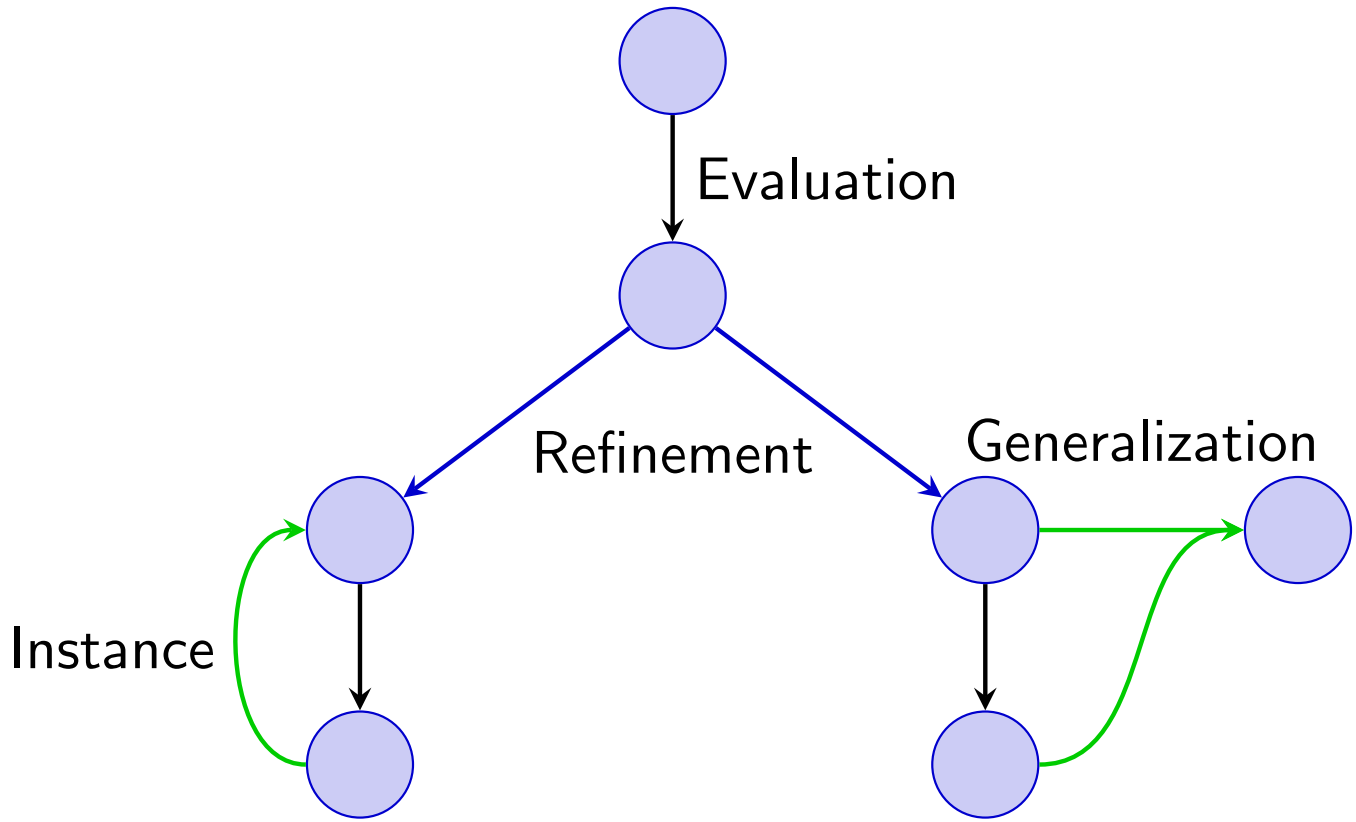
# Summary: Symbolic Execution Graphs



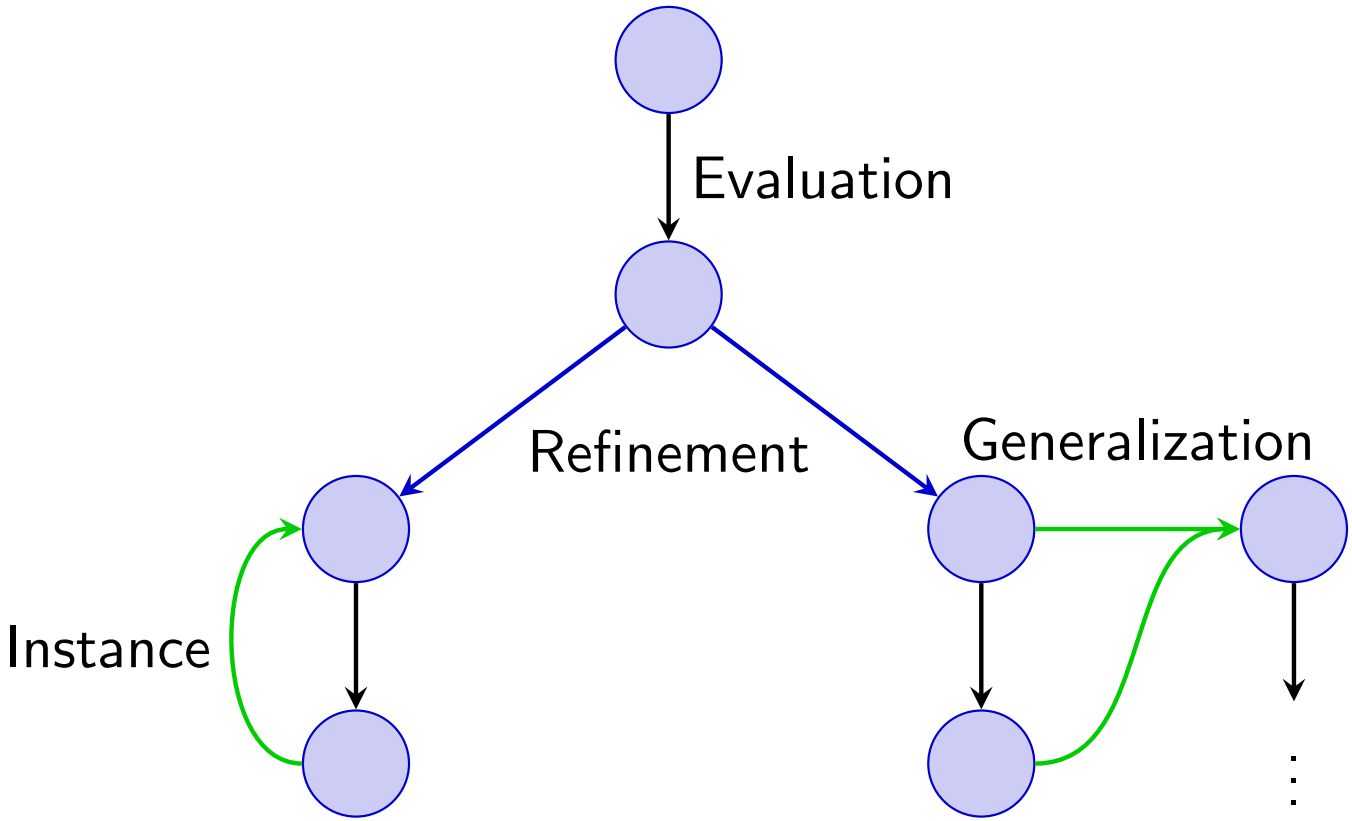
# Summary: Symbolic Execution Graphs



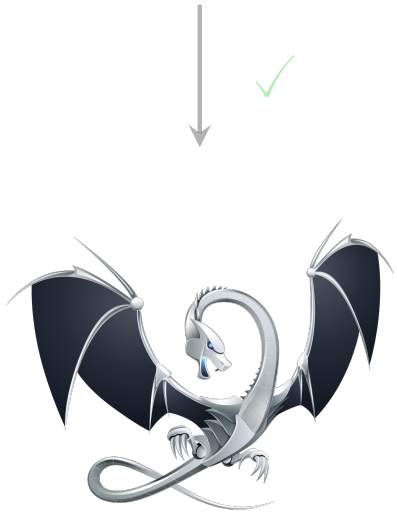
# Summary: Symbolic Execution Graphs



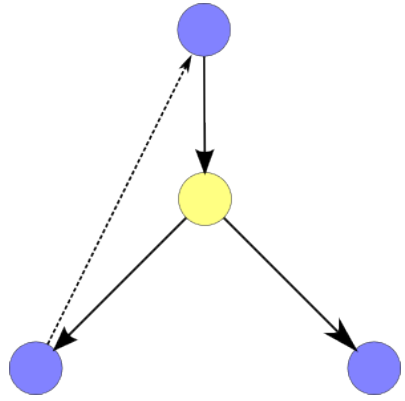
# Summary: Symbolic Execution Graphs



# Symbolic Execution Graph to Integer Transition System

LLVM



Symbolic Execution Graph

?

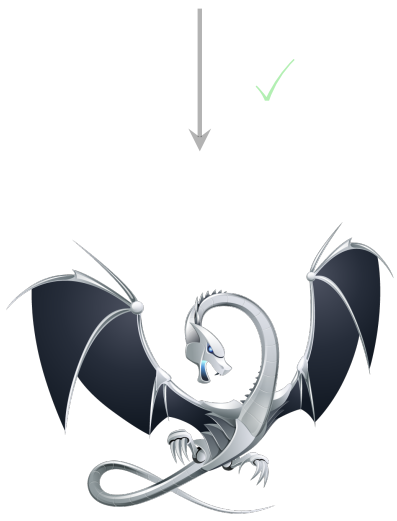
$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

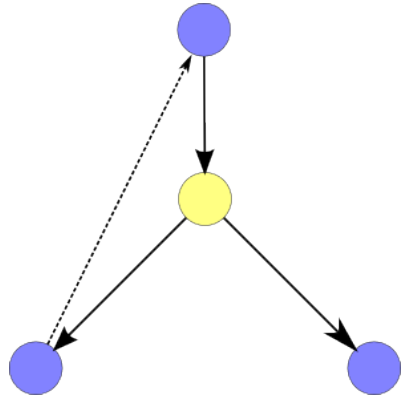
?

YES/NO/
MAYBE

# Symbolic Execution Graph to Integer Transition System

LLVM



Symbolic Execution Graph

YES/NO/
MAYBE

?

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

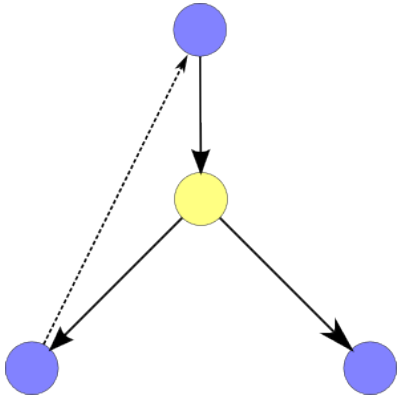
Integer Transition
System

# Symbolic Execution Graph to Integer Transition System


✓

LLVM

✓



Symbolic Execution Graph

YES/NO/
MAYBE

?

$$f(x) \to g(x+1)$$
$$g(x) \to g(x-1)$$

Integer Transition System

# Integer Transition Systems

Term Rewriting System:

$$f(x, y) \to g(y, x)$$

# Integer Transition Systems

Term Rewriting System:

$$f(x, y) \to g(y, x)$$

Integer Transition System:

$$f(x, y) \to g(y + 1, x - 2)$$

# Integer Transition Systems

Term Rewriting System:

$$f(x, y) \to g(y, x)$$

Integer Transition System:

$$f(x, y) \to g(y + 1, x - 2)$$

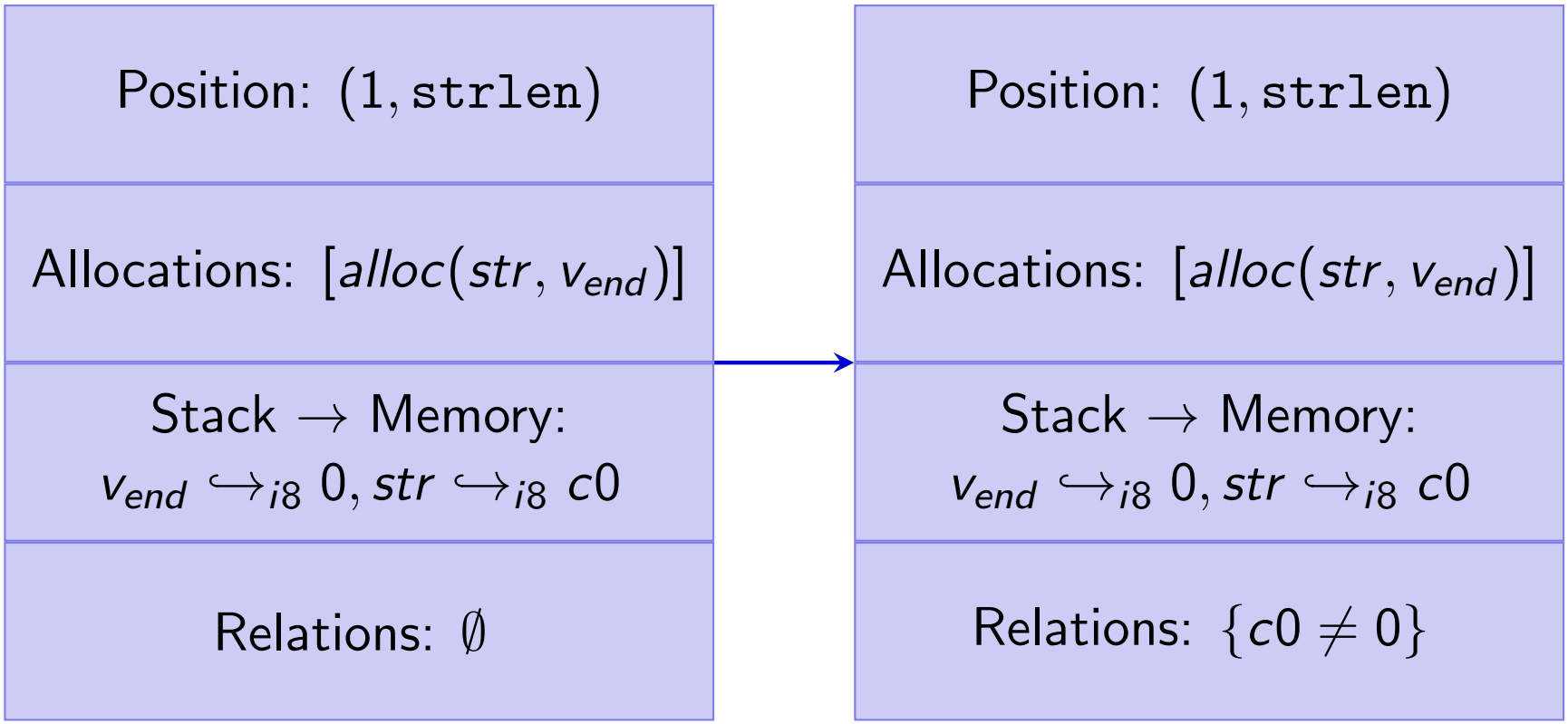
Integer Transition System:

$$f(x, y) \to g(y + 1, x - 2) \quad | \, x > 0$$

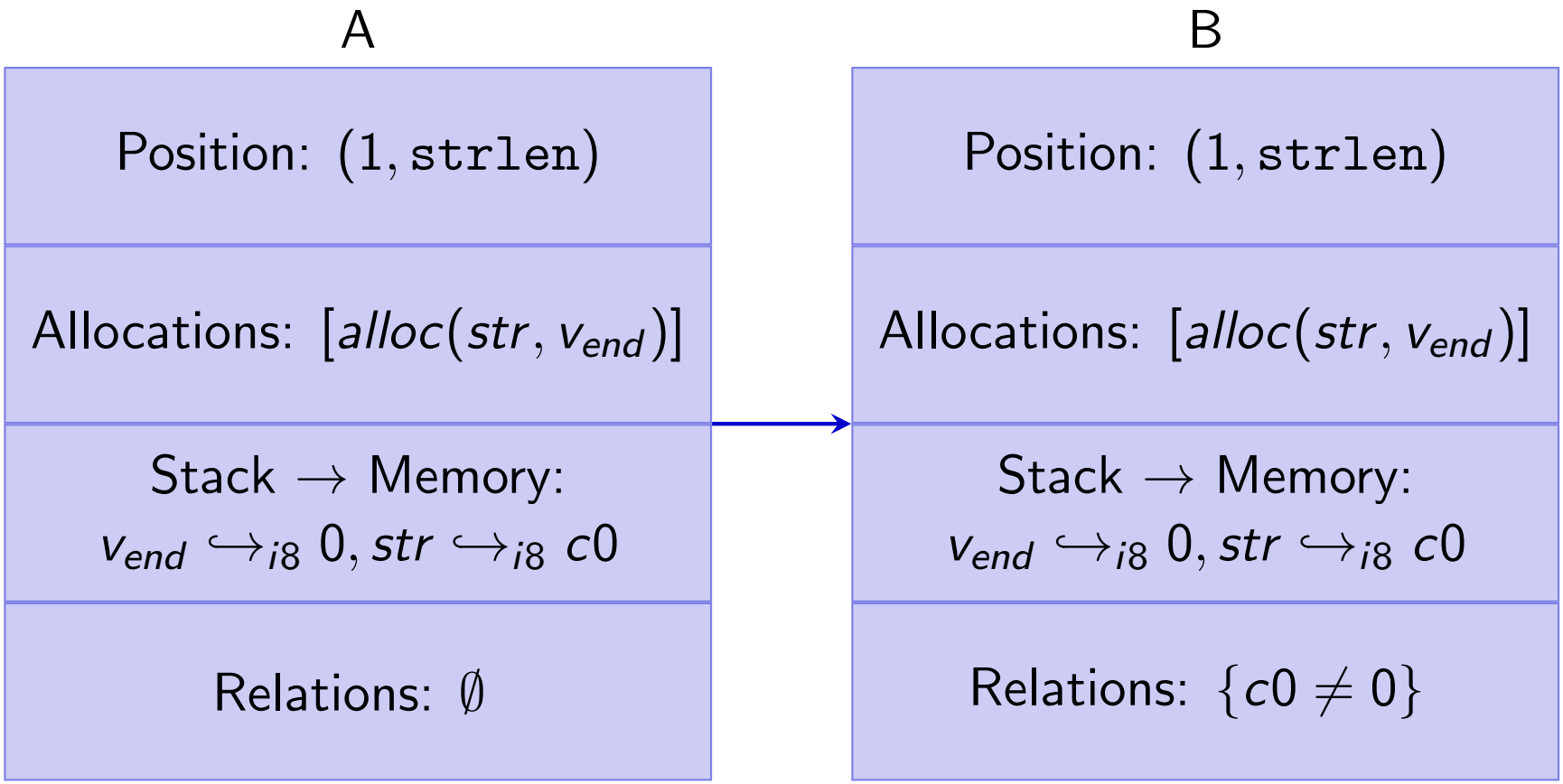
# Symbolic Execution Graph to Integer Transition System

| Position: $(1, \texttt{strlen})$ |
| --- |
| Allocations: $[alloc(str, v_{end})]$ |
| Stack $\rightarrow$ Memory:<br>$v_{end} \hookrightarrow_{i8} 0, str \hookrightarrow_{i8} c0$ |
| Relations: $\emptyset$ |

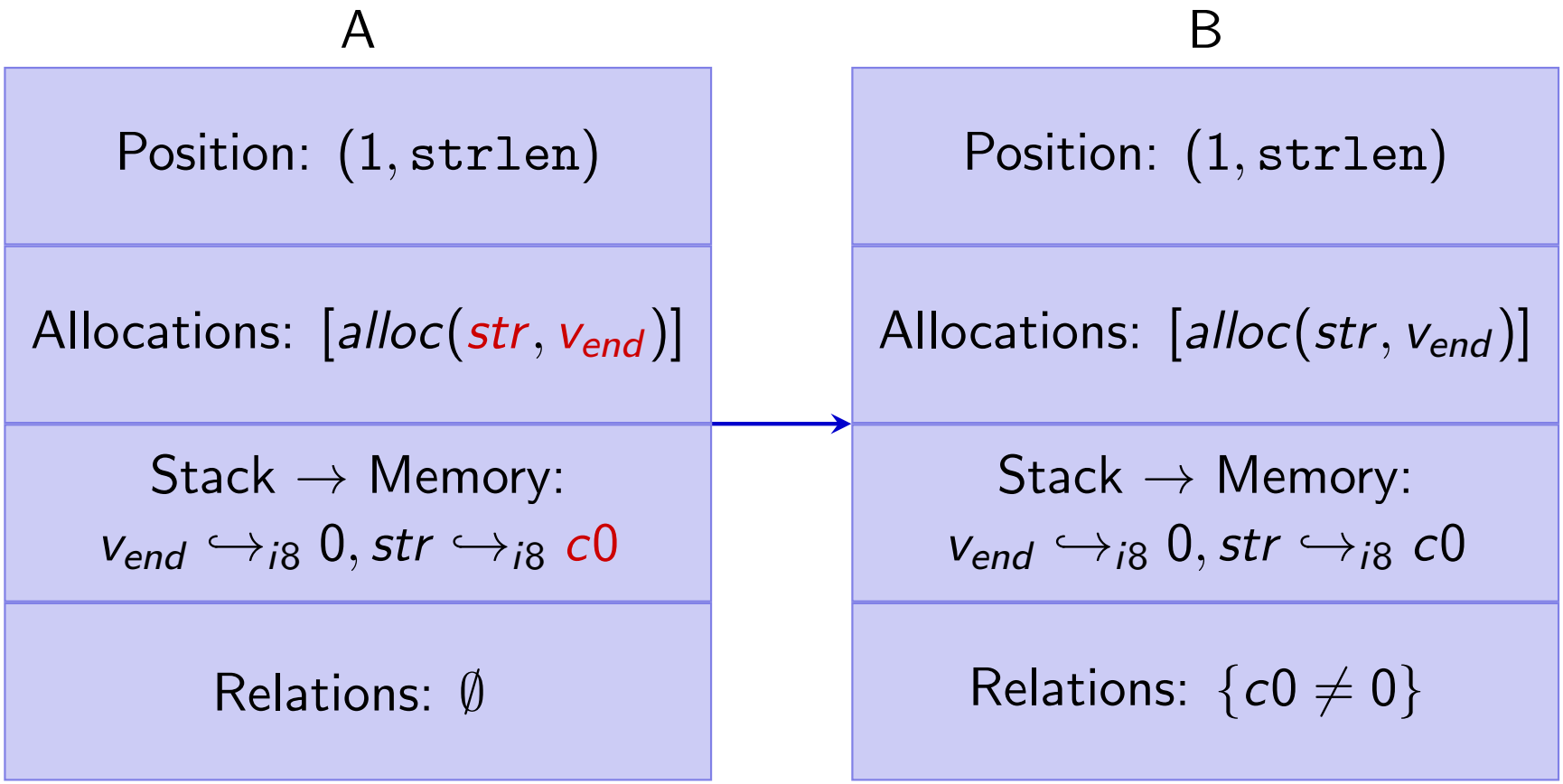
# Symbolic Execution Graph to Integer Transition System



# Symbolic Execution Graph to Integer Transition System

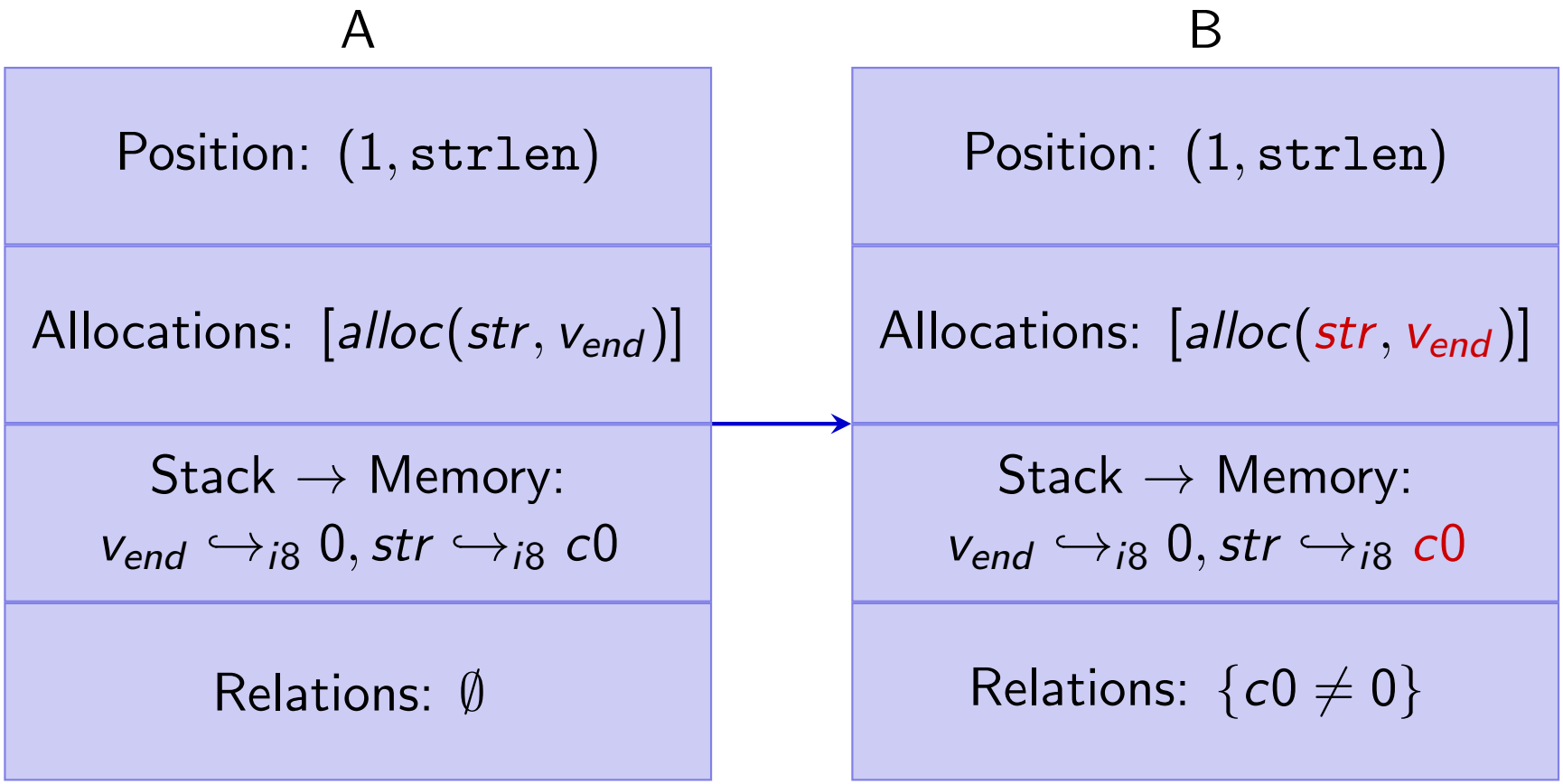


# Symbolic Execution Graph to Integer Transition System



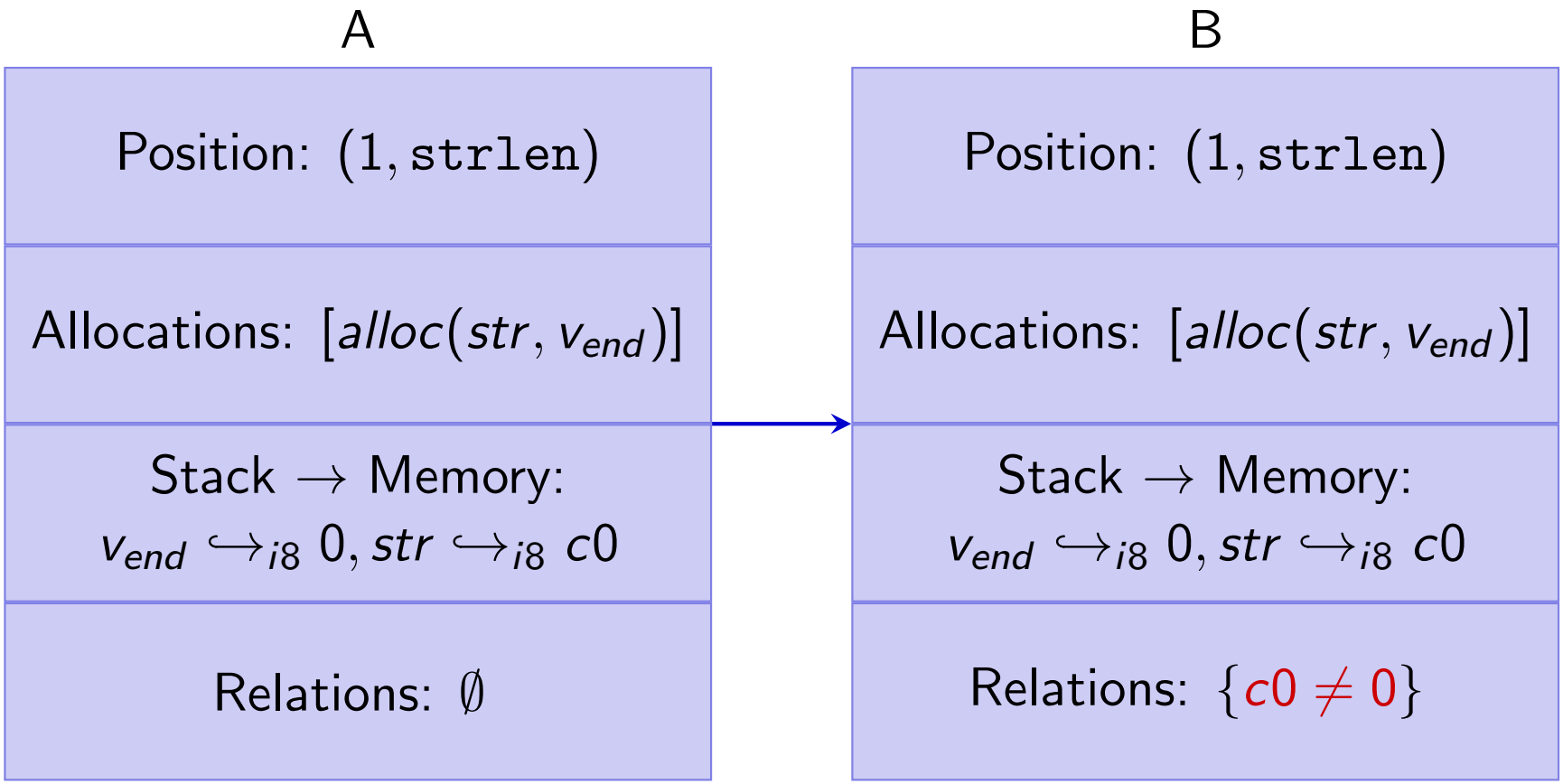

$f_A(str, v_{end}, c0)$

# Symbolic Execution Graph to Integer Transition System




$$f_A(str, v_{end}, c0) \rightarrow f_B(str, v_{end}, c0)$$

# Symbolic Execution Graph to Integer Transition System



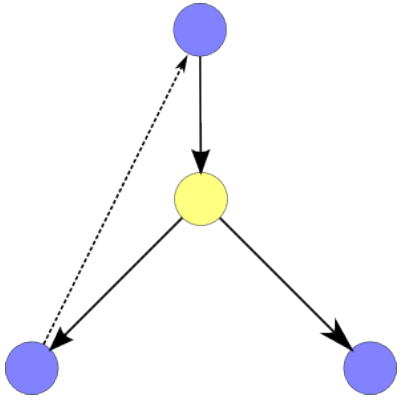

$$f_A(str, v_{end}, c0) \rightarrow f_B(str, v_{end}, c0) \mid c0 \neq 0$$

# Termination of Integer Transition System


✓

LLVM

✓



Symbolic Execution Graph

YES/NO/
MAYBE

?

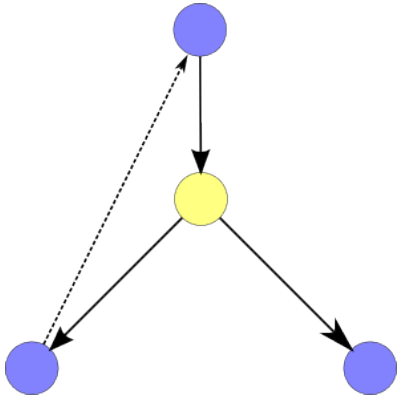
$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

# Termination of Integer Transition System




LLVM



Symbolic Execution Graph

YES/NO/
MAYBE

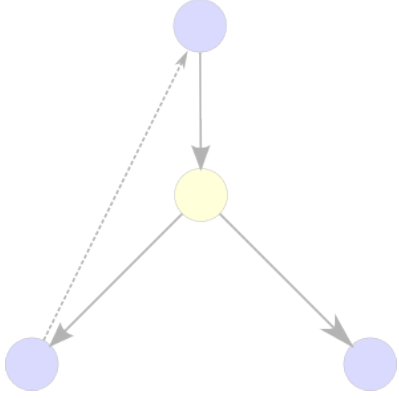
?

$$f(x) \to g(x+1)$$
$$g(x) \to g(x-1)$$

Integer Transition System

# Termination of Integer Transition System


LLVM



Symbolic Execution Graph

YES/NO/
MAYBE

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

# Termination of Integer Transition System

Well-studied problem

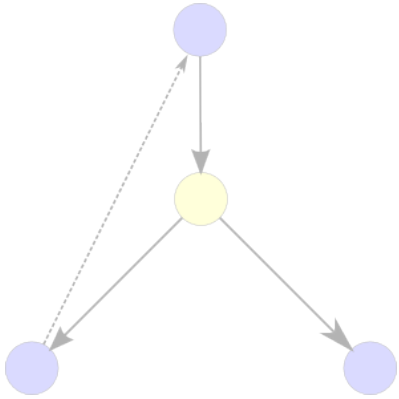
# Termination of Integer Transition System

Well-studied problem

Use known techniques to show termination,
e.g. (Termination of Integer Term Rewriting, Fuhs et al., 2009)

# Termination of Integer Transition System


LLVM



Symbolic Execution Graph

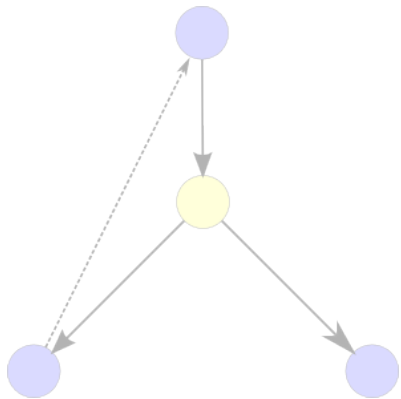
YES/NO/
MAYBE

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

# Termination of Integer Transition System


LLVM



Symbolic Execution Graph

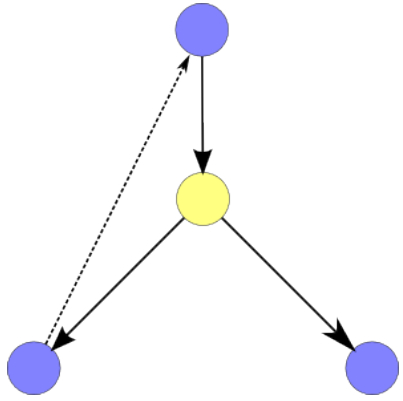
YES/NO/
MAYBE

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

# Overview


LLVM



Symbolic Execution Graph

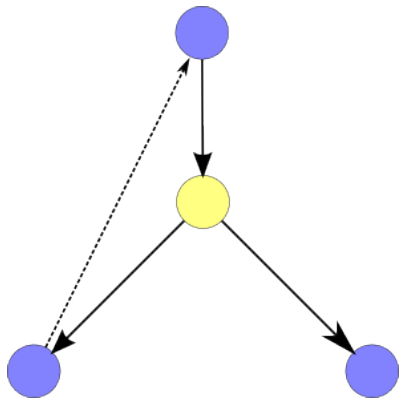
$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

YES/NO/
MAYBE

# My Contributions


LLVM



Symbolic Execution Graph

$$f(x) \to g(x+1)$$
$$g(x) \to g(x-1)$$

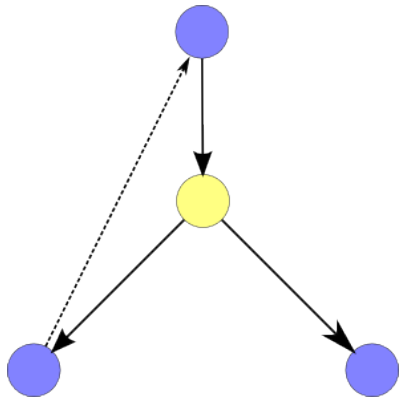
Integer Transition System

YES/NO/
MAYBE

# My Contributions

THE C PROGRAMMING LANGUAGE

LLVM



Symbolic Execution Graph

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

YES/NO/MAYBE

# My Contributions

Given: Abstract State $s$, Integer Relation $r$

Question: Does $s \models r$?

# My Contributions

- Reduction from state to set of arithmetic relations

# My Contributions

- Reduction from state to set of arithmetic relations
- Inference of knowledge from states
  - Formulation of inference in terms of integer relations
  - New framework for inference
  - Parameterization of framework with abstract arithmetic domain
  - Formulation of existing inference in framework

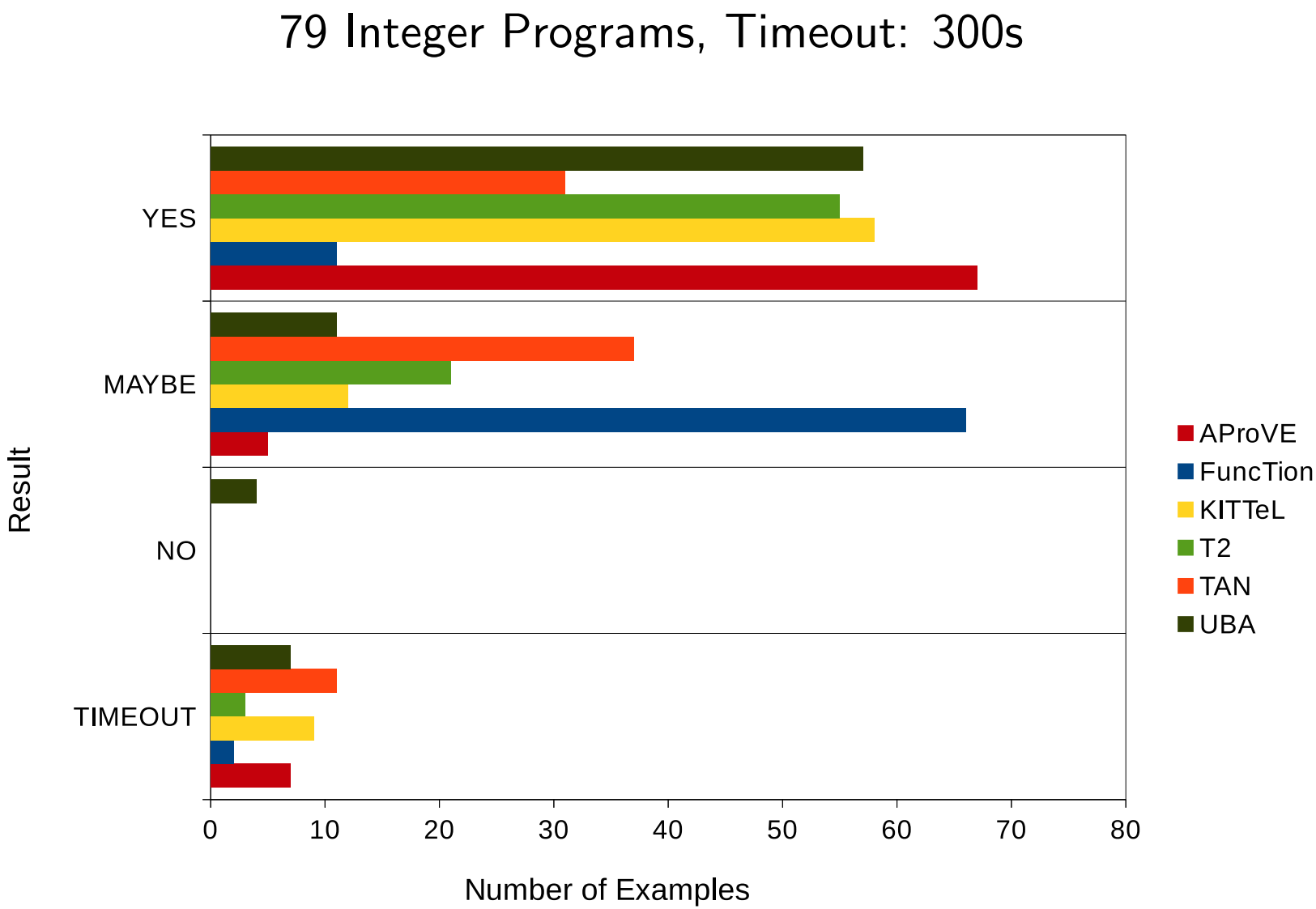
# My Contributions

- Use of Octagon Domain for inference of relations
  - (The Octagon Abstract Domain, Miné, 2006)

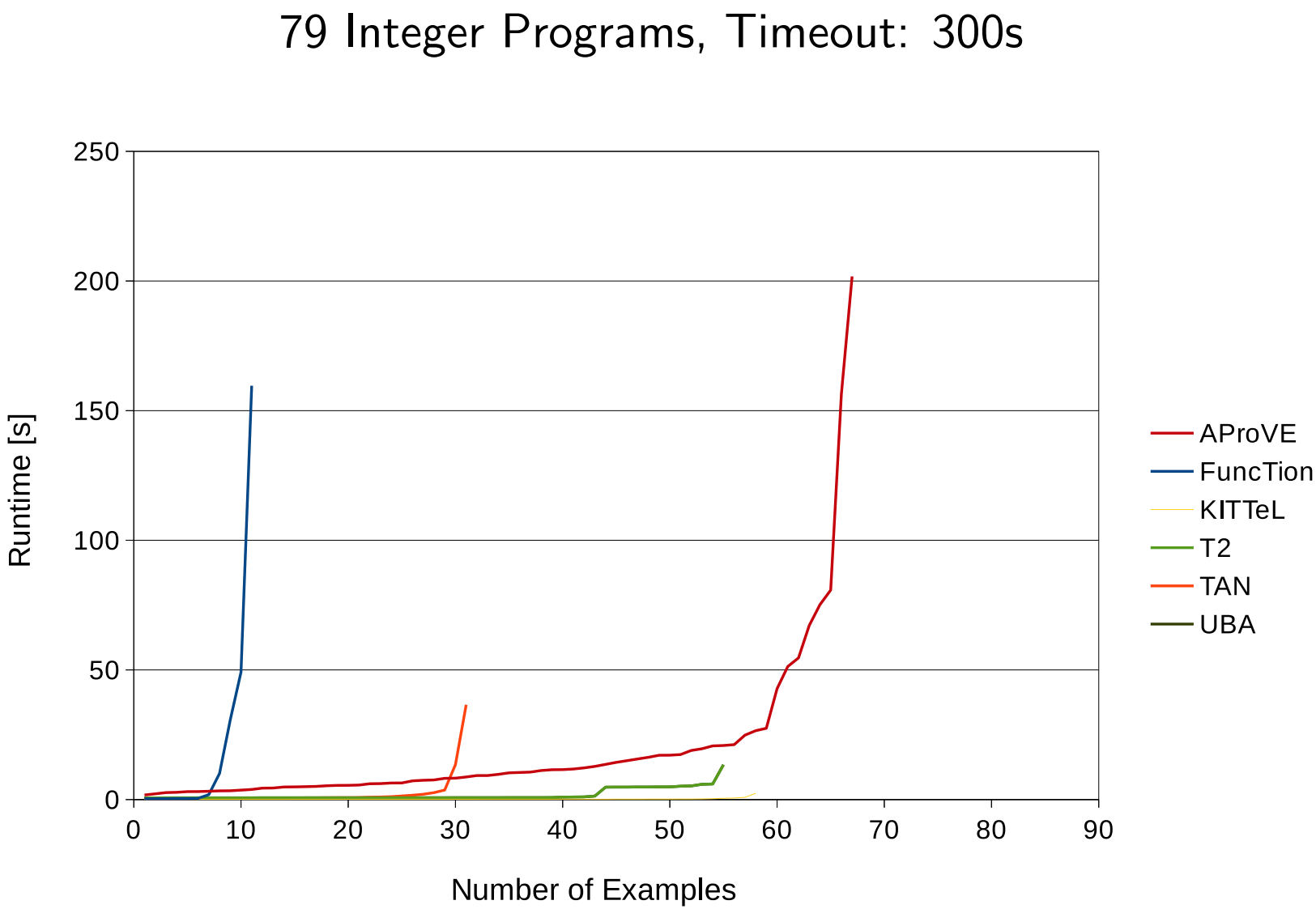
# My Contributions

- Use of Octagon Domain for inference of relations
  - (The Octagon Abstract Domain, Miné, 2006)
- Experimental comparison
  - "Traditional" inference
  - Inference in framework
  - New inference based on Octagons

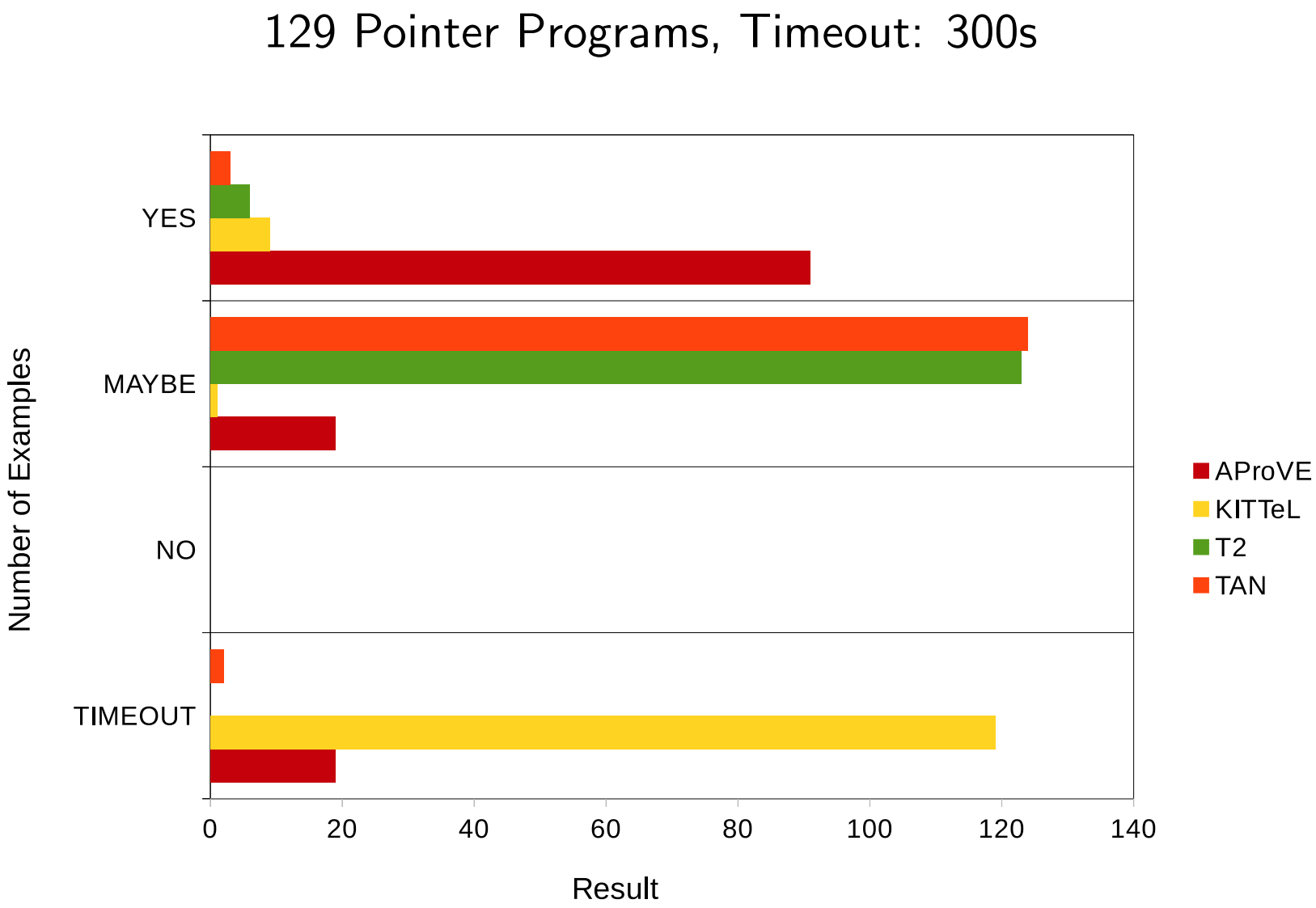
# Evaluation: Empirical Evaluation



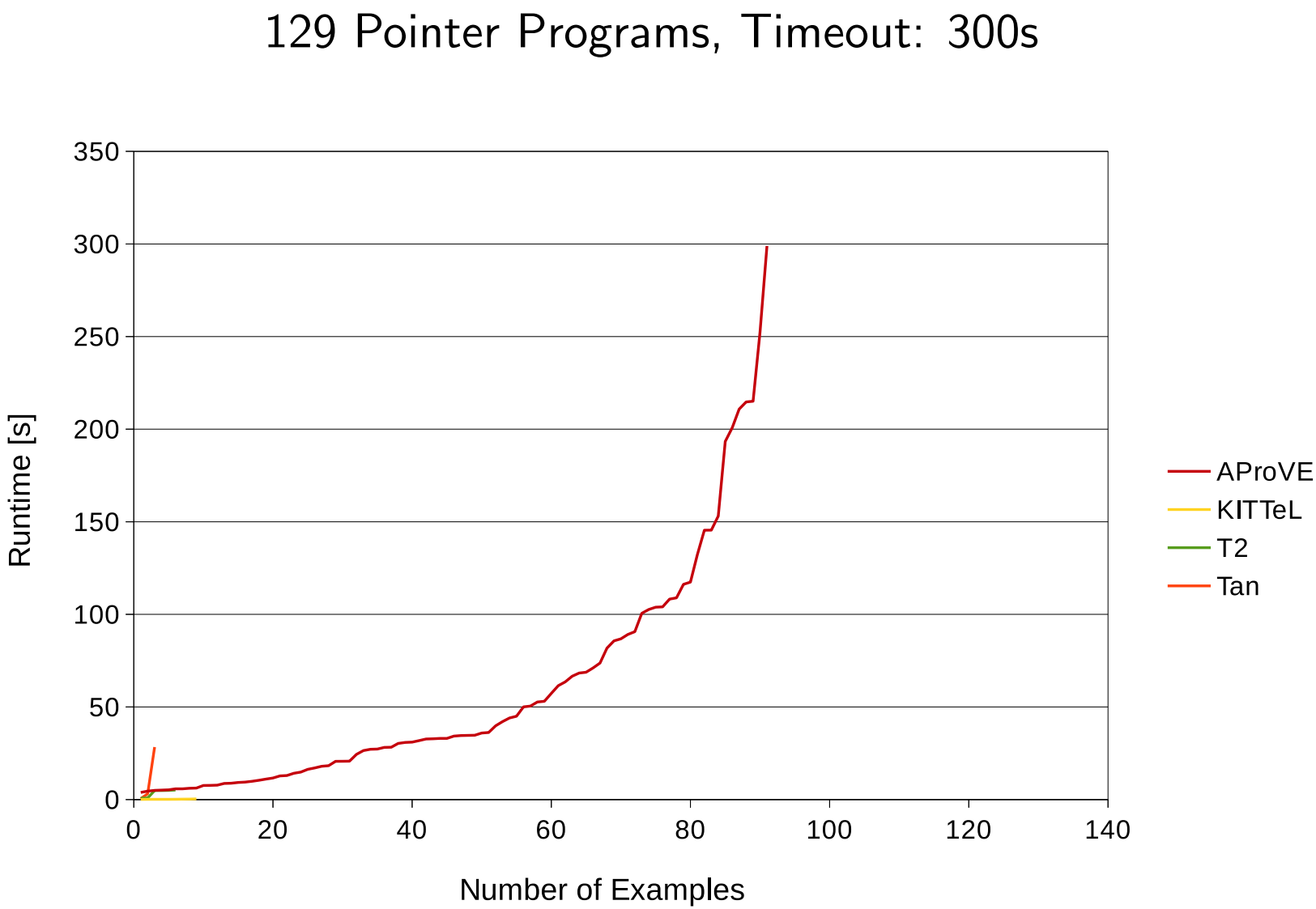
# Evaluation: Empirical Evaluation



# Evaluation: Empirical Evaluation



# Evaluation: Empirical Evaluation



Complete evaluation at

`http://aprove.informatik.rwth-aachen.de/eval/Pointer/`

# Thank you for your attention

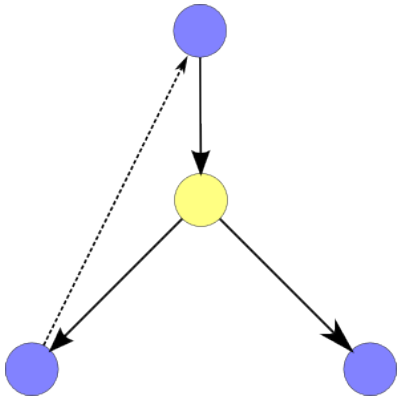
www.alexanderweinert.net
alexander.weinert@rwth-aachen.de

http://alexanderweinert.net/talks
alexander.weinert@rwth-aachen.de


1

LLVM

2



Symbolic Execution Graph

3

$f(x) \to g(x+1)$
$g(x) \to g(x-1)$

Integer Transition System

4

YES/NO/
MAYBE