On Verifying Secret Control Flow Elimination

David Knothe, Oliver Bringmann
FZI Research Center for Information Technology, Karlsruhe, Germany
knothe@fzi.de

Motivation

- 10^{13}€ annual cybercrime damage
- Vulnerabilities (Spectre/Meltdown)

Motivation

- 10^{13}€ annual cybercrime damage
- Vulnerabilities (Spectre/Meltdown)
- Side-Channel Attacks

Image generated by ChatGPT

Timing Side-Channels

```
def cmp_pass(password, input, n):
    for i=1 to n:
        if password[i] <> input[i]:
            return false
    return true
```

Timing Side-Channels

```
def cmp_pass(password, input, n):
    for i=1 to n:
        if password[i] <> input[i]:
            return false
    return true
```

- Execution time depends on secret password

Timing Side-Channels

```
def cmp_pass(password, input, n):
    for i=1 to n:
        if password[i] <> input[i]:
            return false
    return true
```

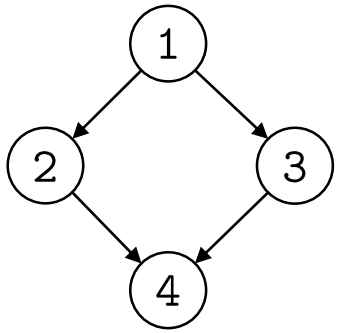
- Execution time depends on secret password
- ~25 timing vulnerabilites on CVE since 2024

Timing Side-Channels

```
def calculate(secret):
    if secret:
        x = pow(cos(y),3)
    else:
        x = x+1
```

Timing Side-Channels

```
def calculate(secret):
    if secret:
        x = pow(cos(y),3)
    else:
        x = x+1
```

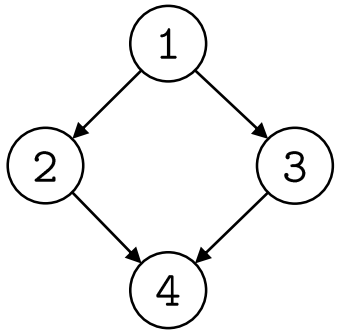


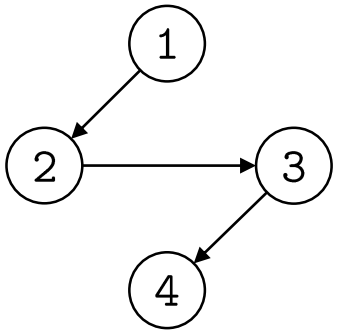
If-Conversion

Ferrante et al., 1987

```
def calculate(secret):
    if secret:
        x = pow(cos(y),3)
    else:
        x = x+1
```

```
def calculate(secret):
    x_1 = pow(cos(y),3)
    x_2 = x+1
    x = secret ? x_1 : x_2
```



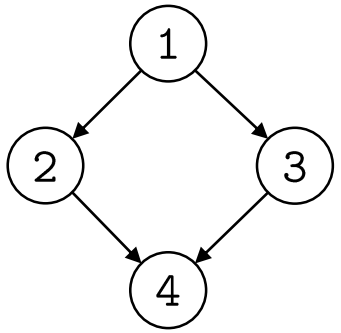


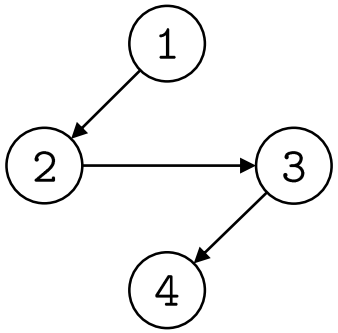
If-Conversion

Ferrante et al., 1987

```
def calculate(secret):
    if secret:
        x = pow(cos(y),3)
    else:
        x = x+1
```

```
def calculate(secret):
    x_1 = pow(cos(y),3)
    x_2 = x+1
    x = secret ? x_1 : x_2
```

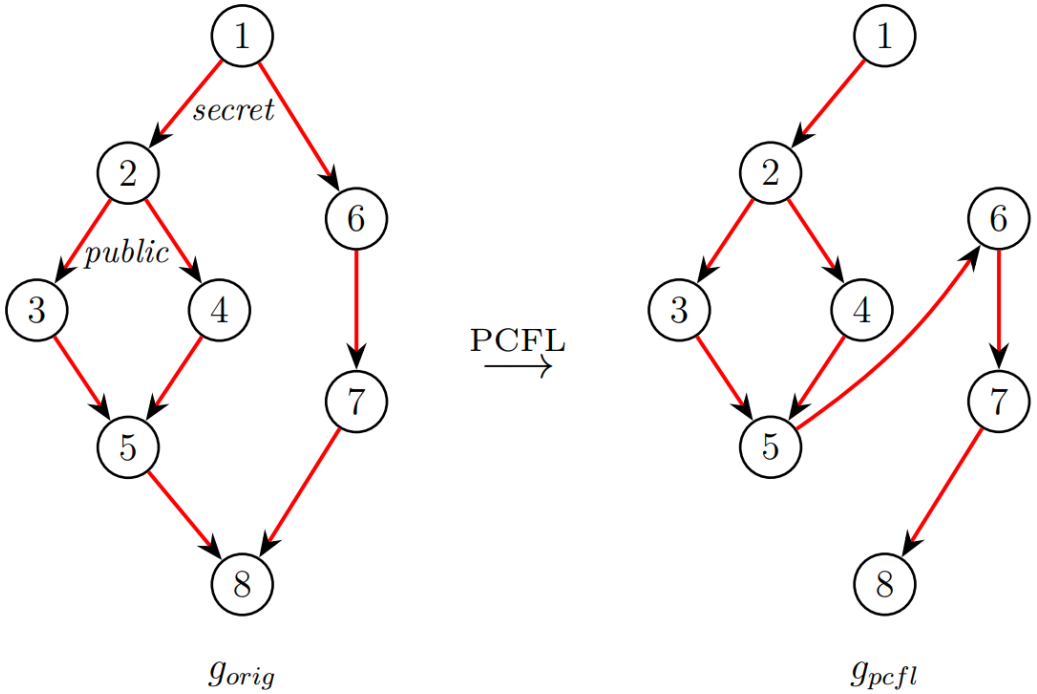




Goal: branch decisions should be independent of any secret parameters

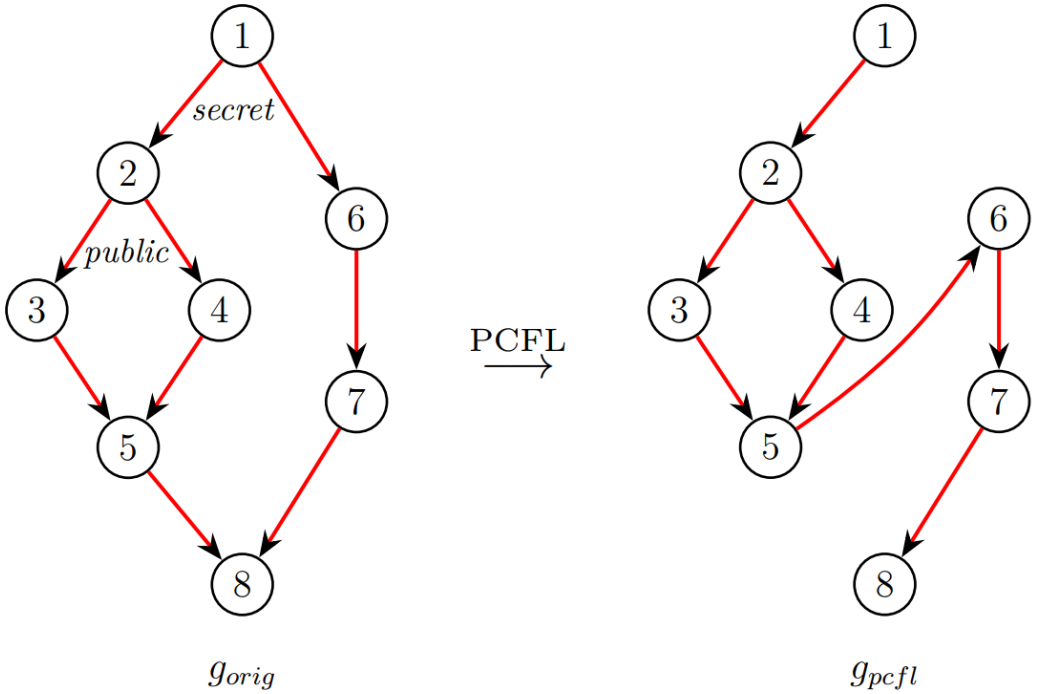
Partial Control-Flow Linearization (PCFL)

Moll and Hack, 2018



Partial Control-Flow Linearization (PCFL)

Moll and Hack, 2018



g_{orig} g_{pcfl}

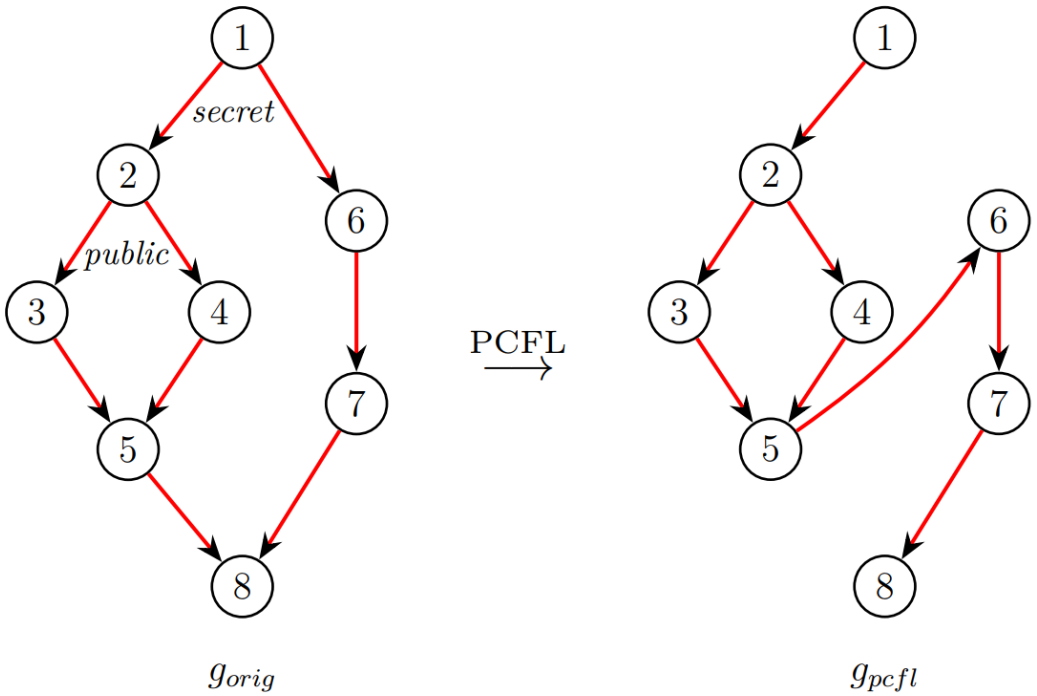
$$\text{detour} : E \to V$$

$$\text{detour}(1, 6) = 2$$

$$\text{detour}(5, 8) = 6$$

Partial Control-Flow Linearization (PCFL)

Moll and Hack, 2018



g_{orig} g_{pcfl}

$$\text{detour} : E \to V$$

$$\text{detour}(1, 6) = 2$$

$$\text{detour}(5, 8) = 6$$

Lemma.

$$\forall (v, w) \in E, w \geq_{PD} \text{detour}(v, w)$$

Constant-Time Compilers

- SC-Eliminator (Wu et al., 2018)
- Constantine (Borello et al., 2021)
- PCFL (Hack and Moll, 2018) (Soares et al., 2023)

Constant-Time Compilers

- SC-Eliminator (Wu et al., 2018)
- Constantine (Borello et al., 2021)
- PCFL (Hack and Moll, 2018) (Soares et al., 2023)

Question: Can we formally verify a constant-time compiler transformation?

CompCert

Leroy, 2009

$$\texttt{Csem} \longrightarrow \ldots \longrightarrow \texttt{RTL} \longrightarrow \ldots \longrightarrow \texttt{Asm}$$

- has multiple intermediate languages

CompCert

Leroy, 2009

$$\texttt{Csem} \longrightarrow \ldots \longrightarrow \texttt{RTL} \longrightarrow \ldots \longrightarrow \texttt{Asm}$$

- has multiple intermediate languages
- small-step semantics
- semantic preservation proven via small-step simulation

CompCert

Leroy, 2009

$$\texttt{Csem} \longrightarrow \ldots \longrightarrow \texttt{RTL} \longrightarrow \ldots \longrightarrow \texttt{Asm}$$

- has multiple intermediate languages
- small-step semantics
- semantic preservation proven via small-step simulation

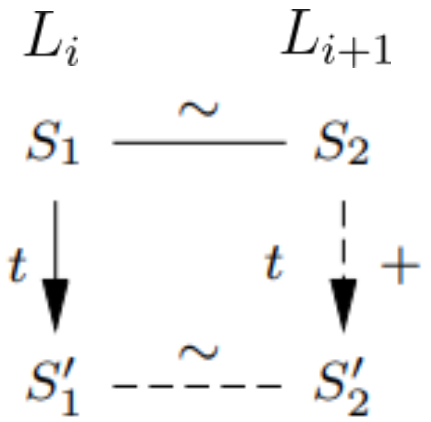


Image from Leroy, 2009

Contributions

- Verify a constant-time compiler transformation (PCFL) in CompCert

Contributions

- Verify a constant-time compiler transformation (PCFL) in CompCert
  - Only for subset of C
  - Eliminate secret *control flow*

Contributions

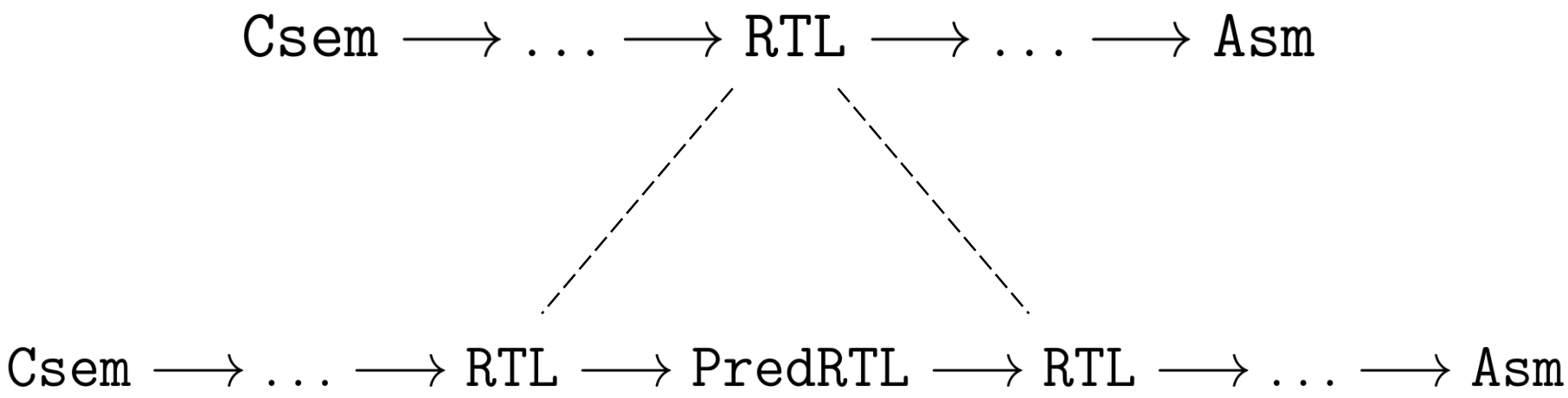
- Verify a constant-time compiler transformation (PCFL) in CompCert
  - Only for subset of C
  - Eliminate secret *control flow*

1. Transformation is correct
2. Transformation removes secret control-flow

Approach

$$\texttt{Csem} \longrightarrow \ldots \longrightarrow \texttt{RTL} \longrightarrow \ldots \longrightarrow \texttt{Asm}$$

Approach



Approach

$$\texttt{Csem} \longrightarrow \ldots \longrightarrow \texttt{RTL} \longrightarrow \ldots \longrightarrow \texttt{Asm}$$

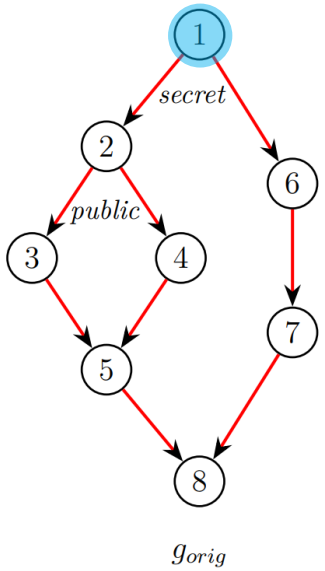
$$\texttt{Csem} \longrightarrow \ldots \longrightarrow \underset{g_{orig}}{\texttt{RTL}} \xrightarrow{\text{PCFL}} \underset{(g_{pcfl}, g_{orig})}{\texttt{PredRTL}} \xrightarrow{\text{predication}} \underset{g_{pcfl}}{\texttt{RTL}} \longrightarrow \ldots \longrightarrow \texttt{Asm}$$

Small-Step Semantics: RTL

RTL state: $(v, regs) \in V \times val^{reg}$

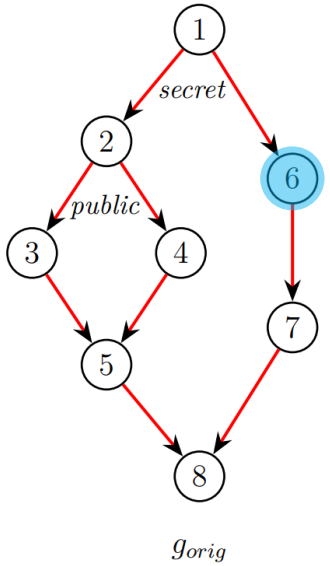
Small-Step Semantics: RTL

RTL state: $(v, regs) \in V \times val^{reg}$



Small-Step Semantics: RTL

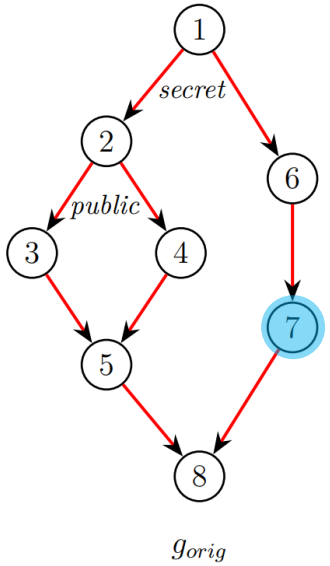
RTL state: $(v, regs) \in V \times val^{reg}$



g_{orig}

Small-Step Semantics: RTL

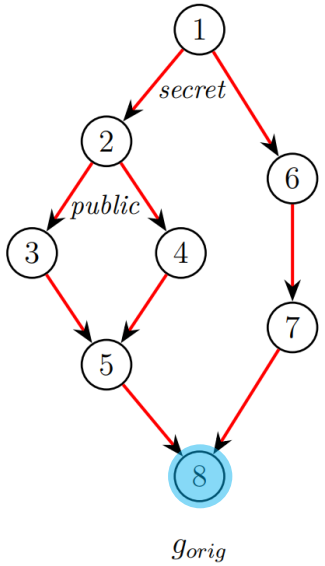
RTL state: $(v, regs) \in V \times val^{reg}$



g_{orig}

Small-Step Semantics: RTL

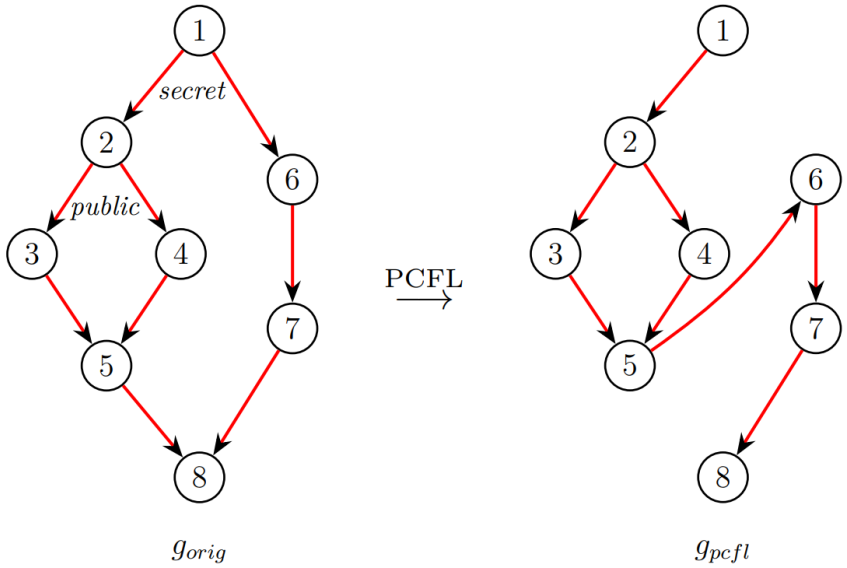
RTL state: $(v, regs) \in V \times val^{reg}$



Small-Step Semantics: PredRTL

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

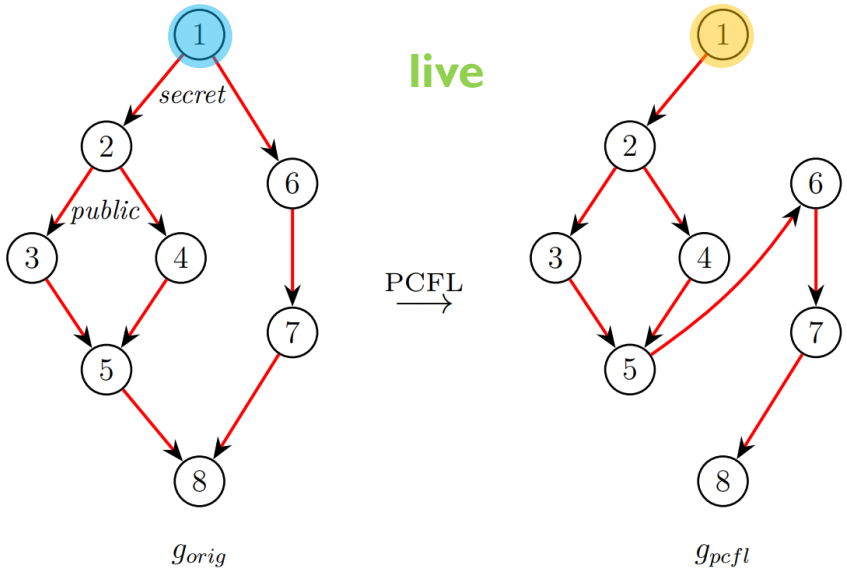


g_{orig} g_{pcfl}

Small-Step Semantics: PredRTL

RTL state: $(v, regs) \in V \times val^{reg}$

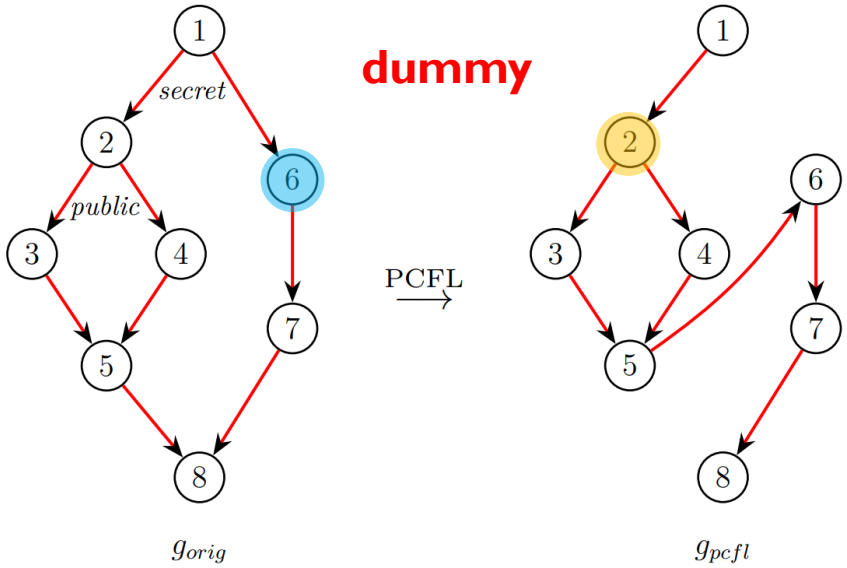
PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$



Small-Step Semantics: PredRTL

RTL state: $(v, regs) \in V \times val^{reg}$

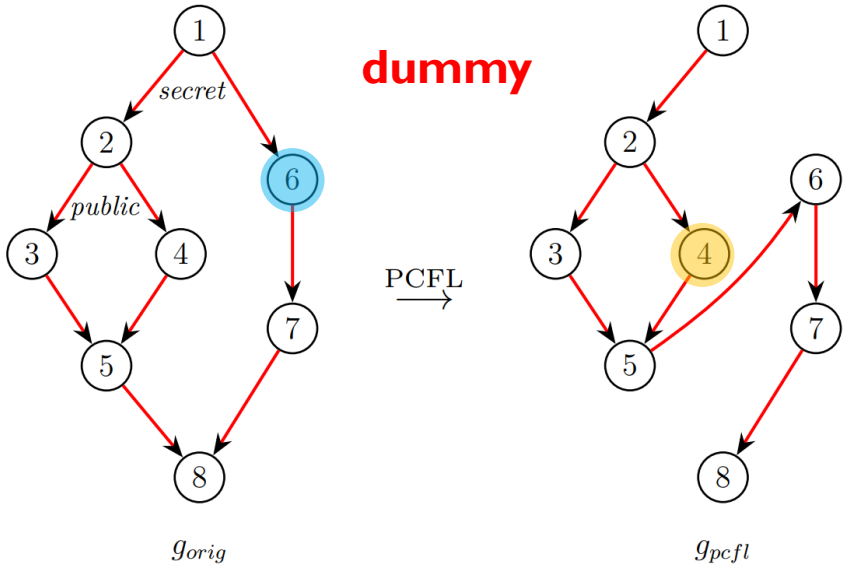
PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$



Small-Step Semantics: PredRTL

RTL state: $(v, regs) \in V \times val^{reg}$

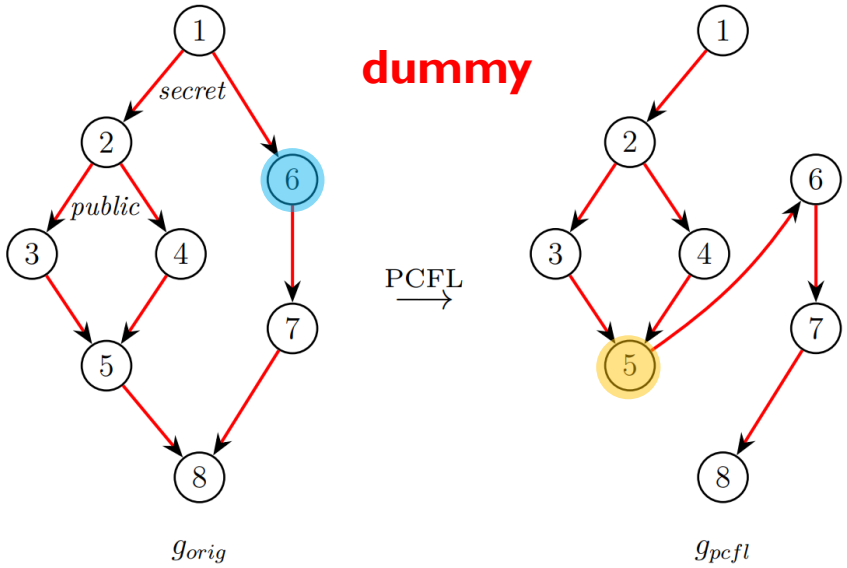
PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$



Small-Step Semantics: PredRTL

RTL state: $(v, regs) \in V \times val^{reg}$

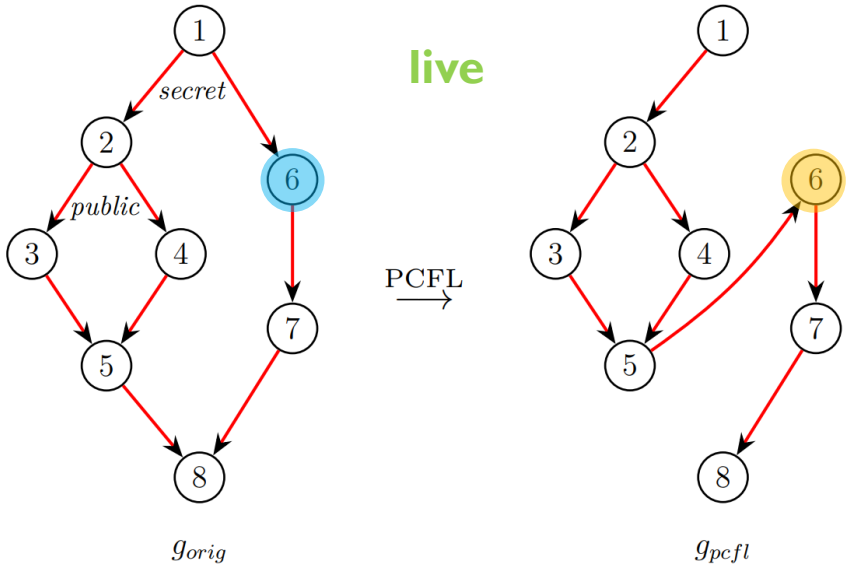
PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$



Small-Step Semantics: PredRTL

RTL state: $(v, regs) \in V \times val^{reg}$

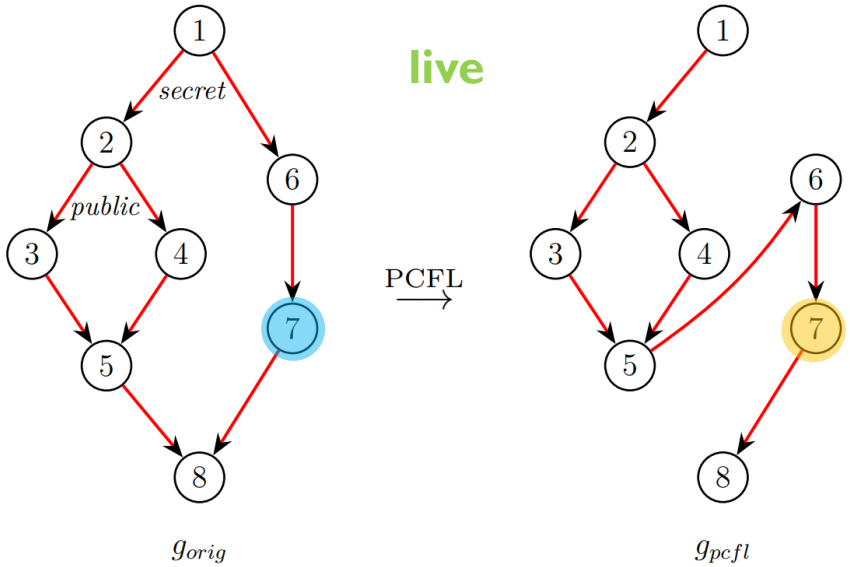
PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$



Small-Step Semantics: PredRTL

RTL state: $(v, regs) \in V \times val^{reg}$

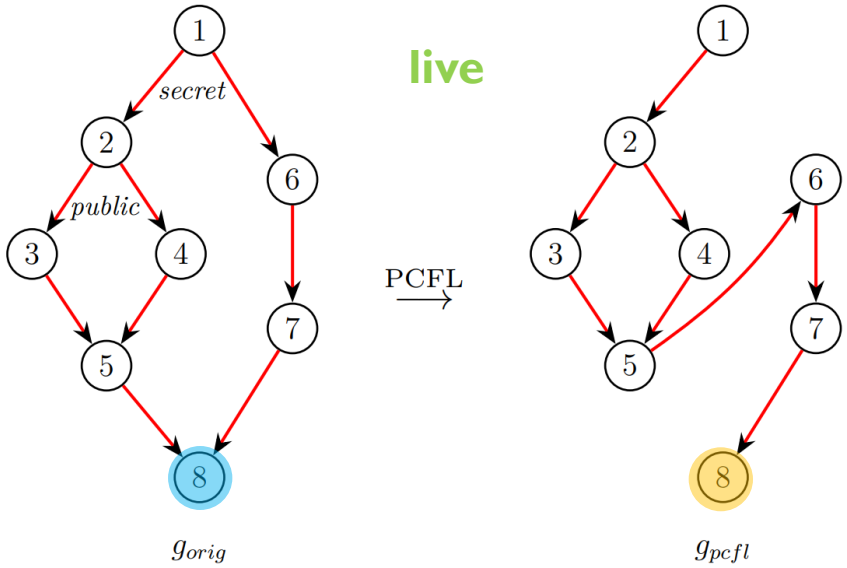
PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$



Small-Step Semantics: PredRTL

RTL state: $(v, regs) \in V \times val^{reg}$

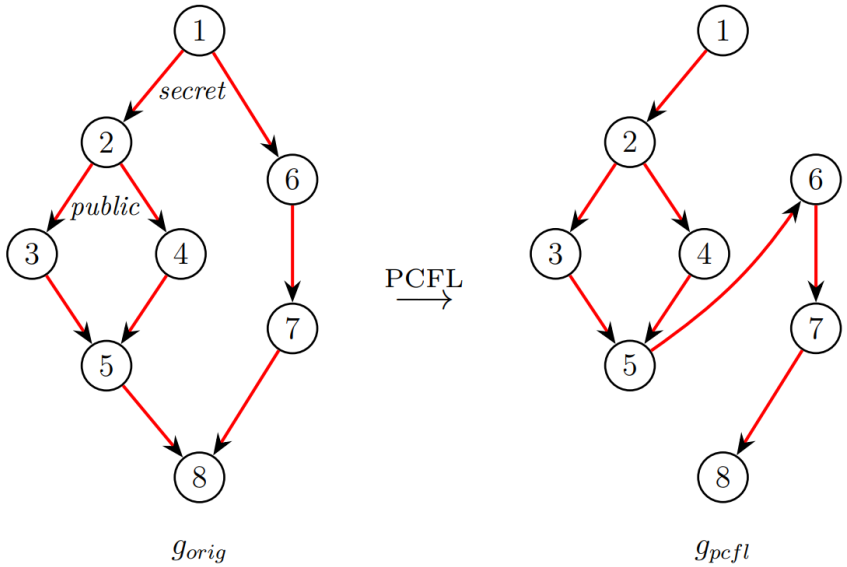
PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$



RTL->PredRTL Matching

RTL state: $(v, regs) \in V \times val^{reg}$

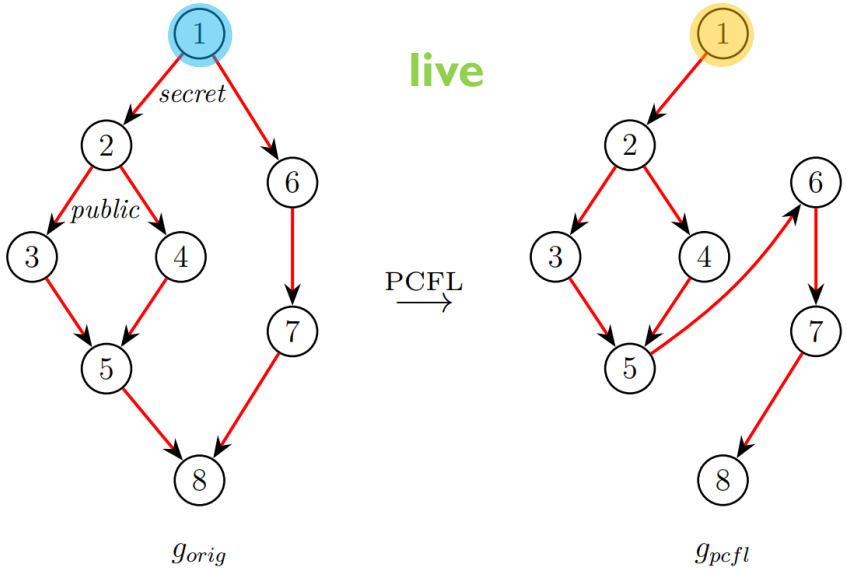
PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$



RTL->PredRTL Matching

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

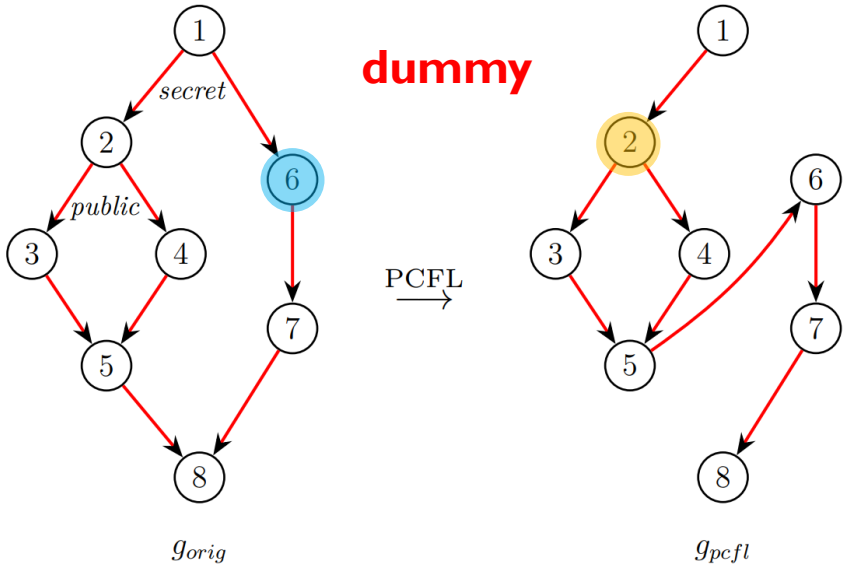


$$(1, r) \sim (1, 1, r)$$

RTL->PredRTL Matching

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

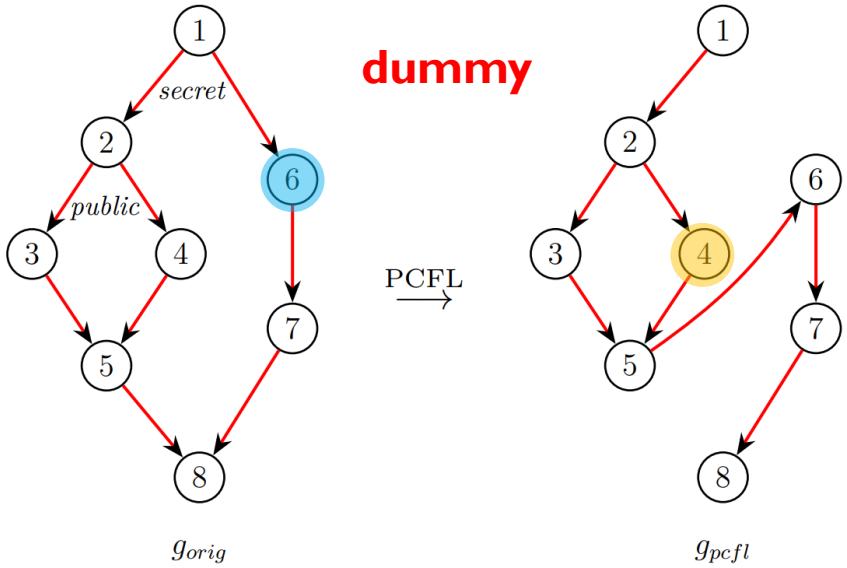


$$(1, r) \sim (1, 1, r)$$

RTL->PredRTL Matching

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

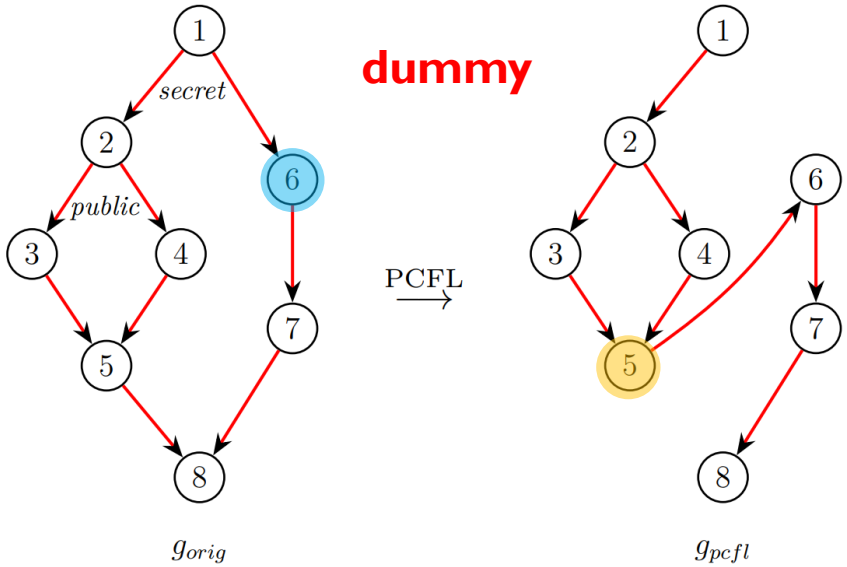


$$(1, r) \sim (1, 1, r)$$

RTL->PredRTL Matching

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

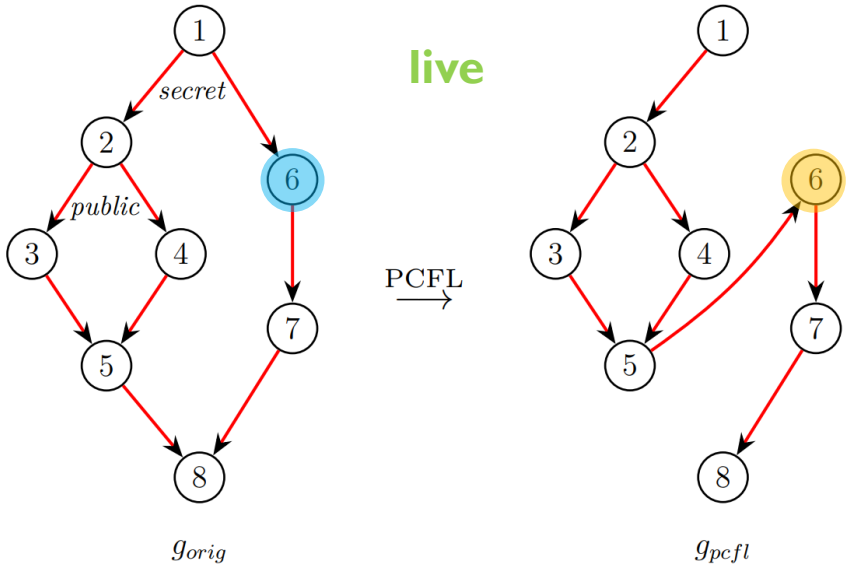


$(1, r) \sim (1, 1, r)$

RTL->PredRTL Matching

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

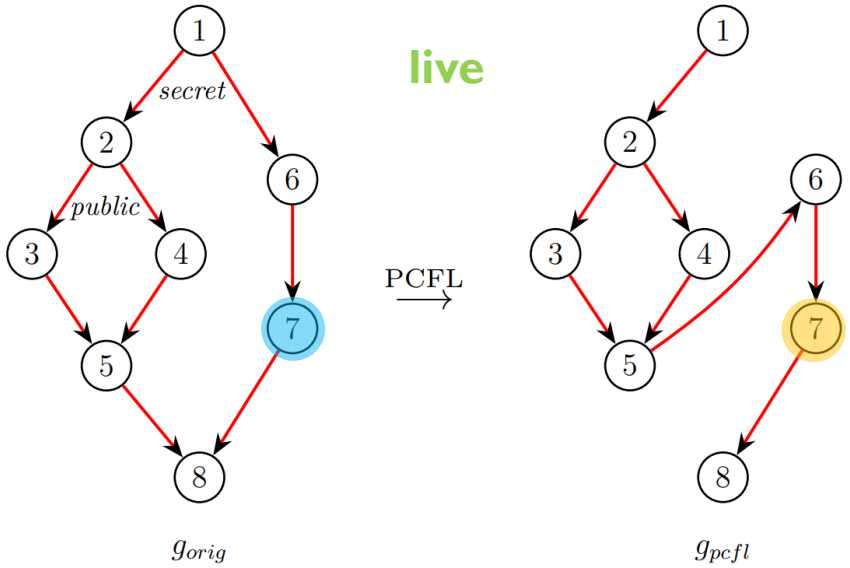


$$(1, r) \sim (1, 1, r)$$
$$(6, r') \sim (6, 6, r')$$

RTL->PredRTL Matching

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

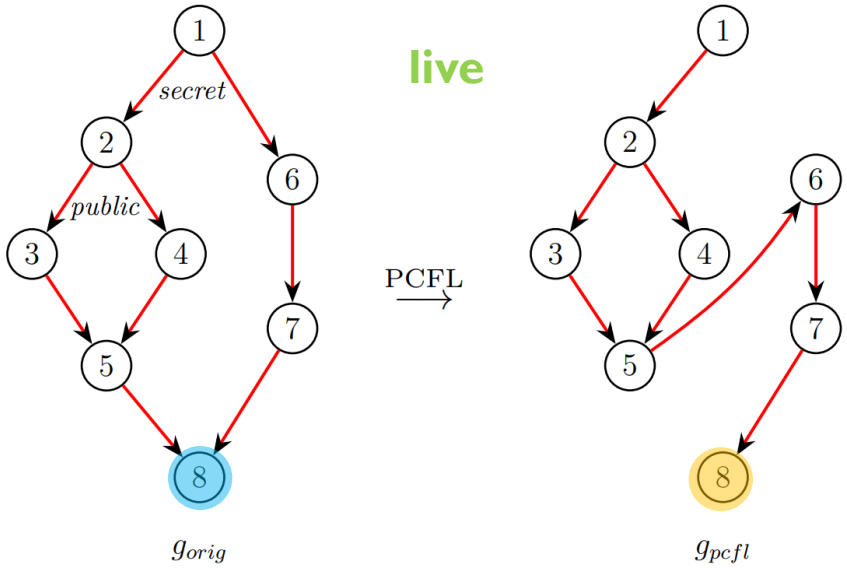


$$(1, r) \sim (1, 1, r)$$
$$(6, r') \sim (6, 6, r')$$
$$(v, r_v) \sim (v, v, r_v)$$

RTL->PredRTL Matching

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$



$$(1, r) \sim (1, 1, r)$$
$$(6, r') \sim (6, 6, r')$$
$$(v, r_v) \sim (v, v, r_v)$$

RTL->PredRTL Simulation

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

RTL->PredRTL Simulation

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

1. Dummy states must not change registers

RTL->PredRTL Simulation

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

1. Dummy states must not change registers
   - Via predication

RTL->PredRTL Simulation

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

1. Dummy states must not change registers
   - Via predication
2. Dummy states must always return

RTL->PredRTL Simulation

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

1. Dummy states must not change registers
   - Via predication
2. Dummy states must always return

Lemma.
$\forall (v, w) \in E, w \geq_{PD} \text{detour}(v, w)$

RTL->PredRTL Simulation

RTL state: $(v, regs) \in V \times val^{reg}$

PredRTL state: $(v_{orig}, v_{pcfl}, regs) \in V \times V \times val^{reg}$

1. Dummy states must not change registers
   - Via predication
2. Dummy states must always return
   - must not crash

Lemma.
$\forall (v, w) \in E, w \geq_{PD} \text{detour}(v, w)$

Unsafe Operations

```
if (t > 0):
  x = 1 / t
else:
  x = 0
```

(a) Original code

```
x1 = 1 / t
x2 = 0
x = (t > 0) ? x1 : x2
```

(b) Linearized code with t being secret

Unsafe Operations

```
if (t > 0):
  x = 1 / t
else:
  x = 0
```

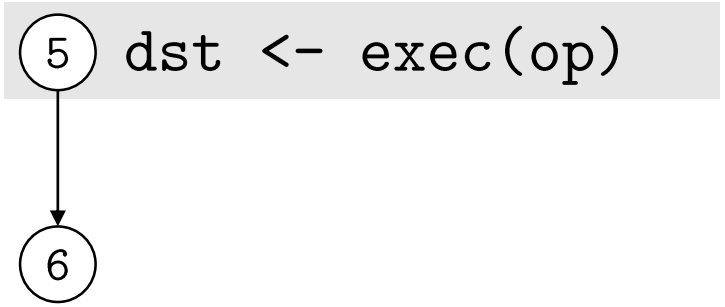
(a) Original code

```
x1 = 1 / t
x2 = 0
x = (t > 0) ? x1 : x2
```

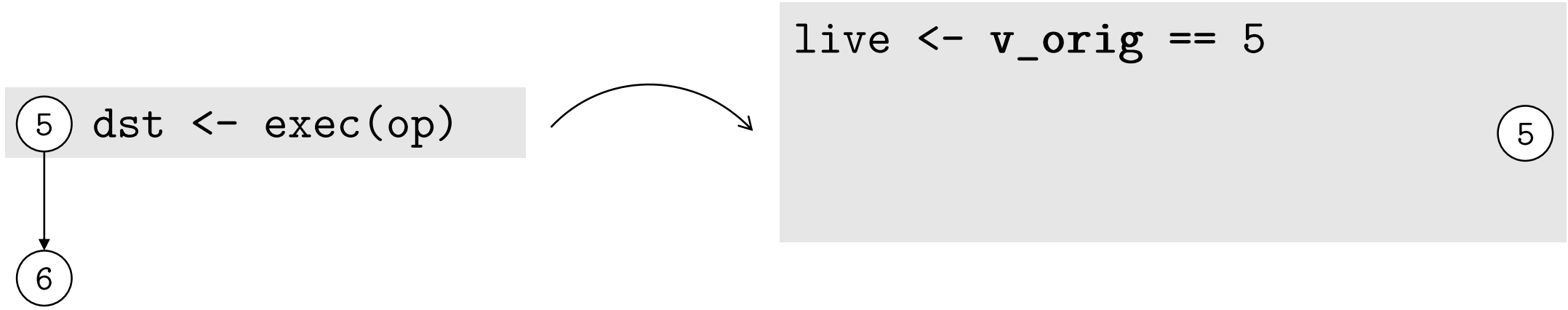
(b) Linearized code with t being secret

- Solution: restrict to *safe* operations
- can be relaxed to *non-uniform* vertices

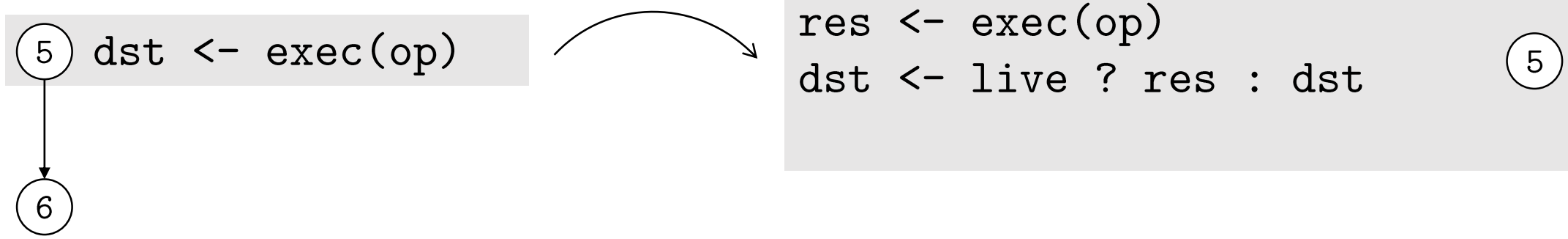
Predication (PredRTL -> RTL)



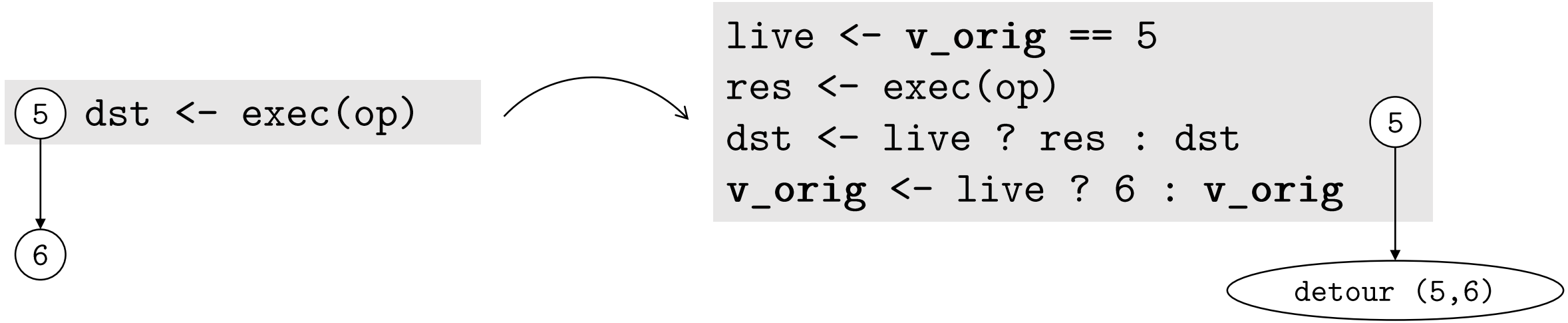
Predication (PredRTL -> RTL)



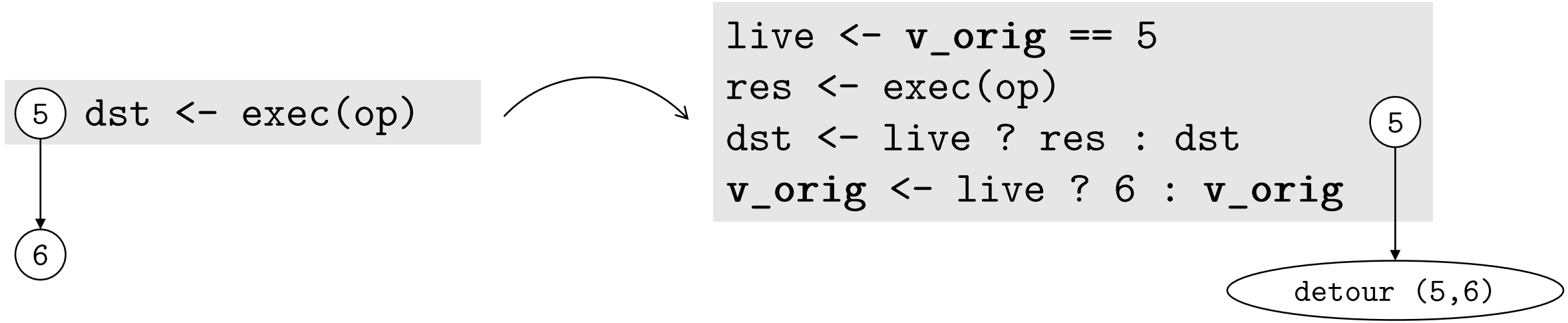
Predication (PredRTL -> RTL)



Predication (PredRTL -> RTL)

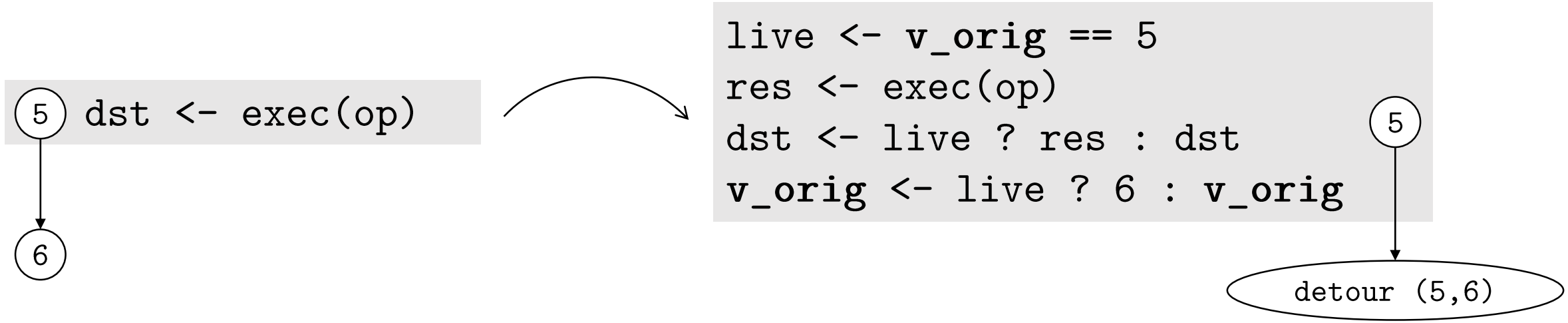


Predication (PredRTL -> RTL)



- `v_orig` register contains v_{orig} position
  - → structure of original graph baked into code
  - → no need for boolean predicate

Predication (PredRTL -> RTL)



- `v_orig` register contains v_{orig} position
  - → structure of original graph baked into code
  - → no need for boolean predicate

Transformation is correct ✓

Control-Flow Security

Goal: branch decisions should be independent of any secret parameters

$$S_{init}(pub, sec_1) \xrightarrow{tr_1}{}^{*} S^1_{fin}$$

$$S_{init}(pub, sec_2) \xrightarrow{tr_2}{}^{*} S^2_{fin}$$

Control-Flow Security

Goal: branch decisions should be independent of any secret parameters

$$S_{init}(pub, sec_1) \xrightarrow{tr_1}{}^{*} S^1_{fin}$$
$$S_{init}(pub, sec_2) \xrightarrow{tr_2}{}^{*} S^2_{fin}$$
$$\Longrightarrow tr_1 = tr_2$$

Control-Flow Security

Goal: branch decisions should be independent of any secret parameters

$$S_{init}(pub, sec_1) \xrightarrow{tr_1}{}^{*} S^1_{fin}$$
$$S_{init}(pub, sec_2) \xrightarrow{tr_2}{}^{*} S^2_{fin}$$
$$\Longrightarrow tr_1 = tr_2$$

- Proven for PredRTL
- Future: down to Asm

Conclusions

- Possible to verify a constant-time transformation for a subset of C
- However, much effort (9000 LOC)

Conclusions

- Possible to verify a constant-time transformation for a subset of C
- However, much effort (9000 LOC)
- Future: tackle remaining challenges
  - big-step semantics better suited?

Thank You!

Image generated by ChatGPT