Automated Analysis of Weak Memory Models

Artem Yushkovskiy[1,2]

MSc Candidate

Supervisors: **Assoc. Prof. Keijo Heljanko**[1]
Docent Igor I. Komarov[2]

[1]**Aalto University** (Espoo, Finland)
Department of Computer Science,
School of Science

[2]**ITMO University** (Saint Petersburg, Russia)
Faculty of Information Security
and Computer Technologies

Espoo, Saint Petersburg, 2018

Problem statement *(Цель работы)*

To rework the proof-of-concept memory model-aware analysis tool `Porthos` [**Porthos17a**] by:

- extending the C-like input language,
- revising its architecture and
- re-implementing the tool in order to enhance performance, extensibility, reliability and maintainability

Task specification *(Задачи работы)*

- Study the general framework for memory model-aware analysis of concurrent programs [**alglave2010shared**];
- Review the existing tools for memory model-aware analysis;
- Examine the existing architecture of `Porthos`, its strengths and weaknesses;
- Design a new architecture for `PorthosC` that allow to extend the input language to the (large subset of) C language, be robust, transparent, efficient and extensible.

Verification of concurrent software

Motivating example: Write-write reordering (compiler relaxations)

{ x=0; y=0; }	
P	Q
$p_0: \quad x \leftarrow 1$	$q_0: \quad y \leftarrow 1$
$p_1: \quad r_p \leftarrow y$	$q_1: \quad r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	

Verification of concurrent software

Motivating example: Write-write reordering (compiler relaxations)

{ x=0; y=0; }	
P	Q
$p_0: \quad x \leftarrow 1$	$q_0: \quad y \leftarrow 1$
$p_1: \quad r_p \leftarrow y$	$q_1: \quad r_q \leftarrow x$
exists $(r_p = 0 \land r_q = 0)$	

Single thread
(no concurrency)

$p_0, p_1, q_0, q_1 \quad (0; 1)$
$q_0, q_1, p_0, p_1 \quad (1; 0)$

Verification of concurrent software

Motivating example: Write-write reordering (compiler relaxations)

{ x=0; y=0; }	
P	Q
$p_0: \quad x \leftarrow 1$	$q_0: \quad y \leftarrow 1$
$p_1: \quad r_p \leftarrow y$	$q_1: \quad r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	

Sequential Consistency
(classical concurrency)

$p_0, p_1, q_0, q_1 \quad (0; 1)$
$q_0, q_1, p_0, p_1 \quad (1; 0)$
$p_0, q_0, p_1, q_1 \quad (1; 1)$
$p_0, q_0, q_1, p_1 \quad (1; 1)$
$q_0, p_0, p_1, q_1 \quad (1; 1)$
$q_0, p_0, q_1, p_1 \quad (1; 1)$

Verification of concurrent software

Motivating example: Write-write reordering (compiler relaxations)

`{ x=0; y=0; }`	
`P`	`Q`
$p_0 :\quad x \leftarrow 1$	$q_0 :\quad y \leftarrow 1$
$p_1 :\quad r_p \leftarrow y$	$q_1 :\quad r_q \leftarrow x$
`exists` $(r_p = 0 \wedge r_q = 0)$	

Total Store Order
(x86, Alpha, ...)

p_0, p_1, q_0, q_1	$(0; 1)$	$\underline{p_1}, \underline{p_0}, q_0, q_1$	$(0; 1)$	$p_0, p_1, \underline{q_1}, \underline{q_0}$	$(0; 1)$	$\underline{p_1}, \underline{p_0}, \underline{q_1}, \underline{q_0}$	$(0; 1)$
q_0, q_1, p_0, p_1	$(1; 0)$	$q_0, q_1, \underline{p_1}, \underline{p_0}$	$(1; 0)$	$\underline{q_1}, \underline{q_0}, p_0, p_1$	$(1; 0)$	$\underline{q_1}, \underline{q_0}, \underline{p_1}, \underline{p_0}$	$(1; 0)$
p_0, q_0, p_1, q_1	$(1; 1)$	$\underline{p_1}, q_0, \underline{p_0}, q_1$	$(0; 1)$	$p_0, \underline{q_1}, p_1, \underline{q_0}$	$(0; 1)$	$\underline{p_1}, \underline{q_1}, \underline{p_0}, \underline{q_0}$	$(0; 0)$
p_0, q_0, q_1, p_1	$(1; 1)$	$\underline{p_1}, q_0, q_1, \underline{p_0}$	$(0; 1)$	$p_0, \underline{q_1}, \underline{q_0}, p_1$	$(1; 1)$	$\underline{p_1}, \underline{q_1}, \underline{q_0}, \underline{p_0}$	$(0; 0)$
q_0, p_0, p_1, q_1	$(1; 1)$	$q_0, \underline{p_1}, \underline{p_0}, q_1$	$(1; 1)$	$\underline{q_1}, p_0, p_1, \underline{q_0}$	$(0; 0)$	$\underline{q_1}, \underline{p_1}, \underline{p_0}, \underline{q_0}$	$(0; 0)$
q_0, p_0, q_1, p_1	$(1; 1)$	$q_0, \underline{p_1}, \underline{q_1}, \underline{p_0}$	$(1; 0)$	$\underline{q_1}, p_0, \underline{q_0}, p_1$	$(1; 0)$	$\underline{q_1}, \underline{p_1}, \underline{q_0}, \underline{p_0}$	$(0; 0)$

Verification of concurrent software

Motivating example: Write-write reordering (compiler relaxations)

{ x=0; y=0; }	
P	Q
$p_0: \quad x \leftarrow 1$	$q_0: \quad y \leftarrow 1$
$p_1: \quad r_p \leftarrow y$	$q_1: \quad r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	

Total Store Order
(x86, Alpha, ...)

p_0, p_1, q_0, q_1	$(0;1)$	$\underline{p_1}, \underline{p_0}, q_0, q_1$	$(0;1)$	$p_0, p_1, \underline{q_1}, \underline{q_0}$	$(0;1)$	$\underline{p_1}, \underline{p_0}, \underline{q_1}, \underline{q_0}$	$(0;1)$
q_0, q_1, p_0, p_1	$(1;0)$	$q_0, q_1, \underline{p_1}, \underline{p_0}$	$(1;0)$	$\underline{q_1}, \underline{q_0}, p_0, p_1$	$(1;0)$	$\underline{q_1}, \underline{q_0}, \underline{p_1}, \underline{p_0}$	$(1;0)$
p_0, q_0, p_1, q_1	$(1;1)$	$\underline{p_1}, q_0, \underline{p_0}, q_1$	$(0;1)$	$p_0, \underline{q_1}, p_1, \underline{q_0}$	$(0;1)$	$\underline{p_1}, \underline{q_1}, \underline{p_0}, \underline{q_0}$	$(0;0)$
p_0, q_0, q_1, p_1	$(1;1)$	$\underline{p_1}, q_0, q_1, \underline{p_0}$	$(0;1)$	$p_0, \underline{q_1}, \underline{q_0}, p_1$	$(1;1)$	$\underline{p_1}, \underline{q_1}, \underline{q_0}, \underline{p_0}$	$(0;0)$
q_0, p_0, p_1, q_1	$(1;1)$	$q_0, \underline{p_1}, \underline{p_0}, q_1$	$(1;1)$	$\underline{q_1}, p_0, p_1, \underline{q_0}$	$(0;0)$	$\underline{q_1}, \underline{p_1}, \underline{p_0}, \underline{q_0}$	$(0;0)$
q_0, p_0, q_1, p_1	$(1;1)$	$q_0, \underline{p_1}, q_1, \underline{p_0}$	$(1;0)$	$\underline{q_1}, p_0, \underline{q_0}, p_1$	$(1;0)$	$\underline{q_1}, \underline{p_1}, \underline{q_0}, \underline{p_0}$	$(0;0)$

Verification of concurrent software

Motivating example: Store buffering (hardware relaxations)

{ x=0; y=0; }	
P	Q
p_0 : $x \leftarrow 1$	q_0 : $y \leftarrow 1$
p_1 : $r_p \leftarrow y$	q_1 : $r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	

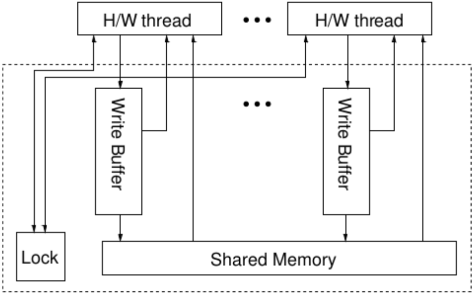
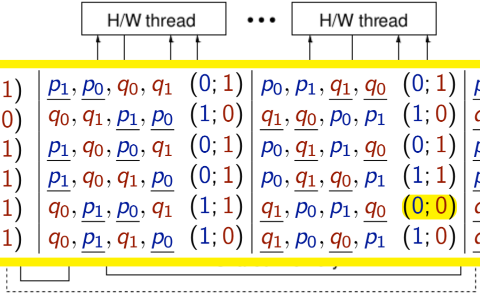


Figure: An x86-TSO abstract machine [**sewell2010x86**]

Verification of concurrent software

Motivating example: Store buffering (hardware relaxations)

{ x=0; y=0; }	
P	Q
p_0 : $x \leftarrow 1$	q_0 : $y \leftarrow 1$
p_1 : $r_p \leftarrow y$	q_1 : $r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	



p_0, p_1, q_0, q_1	$(0;1)$	$\underline{p_1}, \underline{p_0}, q_0, q_1$	$(0;1)$	$p_0, p_1, \underline{q_1}, \underline{q_0}$	$(0;1)$	$\underline{p_1}, \underline{p_0}, \underline{q_1}, \underline{q_0}$	$(0;1)$
q_0, q_1, p_0, p_1	$(1;0)$	$q_0, q_1, \underline{p_1}, \underline{p_0}$	$(1;0)$	$\underline{q_1}, \underline{q_0}, p_0, p_1$	$(1;0)$	$\underline{q_1}, \underline{q_0}, \underline{p_1}, \underline{p_0}$	$(1;0)$
p_0, q_0, p_1, q_1	$(1;1)$	$\underline{p_1}, q_0, \underline{p_0}, q_1$	$(0;1)$	$p_0, \underline{q_1}, p_1, \underline{q_0}$	$(0;1)$	$\underline{p_1}, \underline{q_1}, \underline{p_0}, \underline{q_0}$	$(0;0)$
p_0, q_0, q_1, p_1	$(1;1)$	$\underline{p_1}, q_0, q_1, \underline{p_0}$	$(0;1)$	$p_0, \underline{q_1}, \underline{q_0}, p_1$	$(1;1)$	$\underline{p_1}, \underline{q_1}, \underline{q_0}, \underline{p_0}$	$(0;0)$
q_0, p_0, p_1, q_1	$(1;1)$	$q_0, \underline{p_1}, \underline{p_0}, q_1$	$(1;1)$	$\underline{q_1}, p_0, p_1, \underline{q_0}$	$(0;0)$	$\underline{q_1}, \underline{p_1}, \underline{p_0}, \underline{q_0}$	$(0;0)$
q_0, p_0, q_1, p_1	$(1;1)$	$q_0, \underline{p_1}, q_1, \underline{p_0}$	$(1;0)$	$\underline{q_1}, p_0, \underline{q_0}, p_1$	$(1;0)$	$\underline{q_1}, \underline{p_1}, \underline{q_0}, \underline{p_0}$	$(0;0)$

Figure: An x86-TSO abstract machine [**sewell2010x86**]

Weak memory model-aware analysis

Event-based program representation

- **Event** $\in \mathbb{E}$, a low-level primitive operation:
  - *memory event* $\in \mathbb{M} = \mathbb{R} \cup \mathbb{W}$: access to a local/shared memory,
  - *computational event* $\in \mathbb{C}$: computation over local memory, and
  - *barrier event* $\in \mathbb{B}$: synchronisation fences;
- **Relation** $\subseteq \mathbb{E} \times \mathbb{E}$:
  - *basic relations*:
    - *program-order* relation $\mathtt{po} \subseteq \mathbb{E} \times \mathbb{E}$: (control-flow),
    - *read-from* relation $\mathtt{rf} \subseteq \mathbb{W} \times \mathbb{R}$: (data-flow), and
    - *coherence-order* relation $\mathtt{co} \subseteq \mathbb{W} \times \mathbb{W}$: (data-flow);
  - *derived relations*:
    - *union* $\mathtt{r1} \mid \mathtt{r2}$,
    - *sequence* $\mathtt{r1}\ ;\ \mathtt{r2}$,
    - *transitive closure* $\mathtt{r+}$,
    - $\ldots$;
- **Assertion** over relations or sets of events:
  - *acyclicity*, *irreflexivity* or *emptiness*

Weak memory model-aware analysis

Testing candidate executions

{ x=0; y=0; }	
P	Q
p_0 : $x \leftarrow 1$	q_0 : $y \leftarrow 1$
p_1 : $r_p \leftarrow y$	q_1 : $r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	

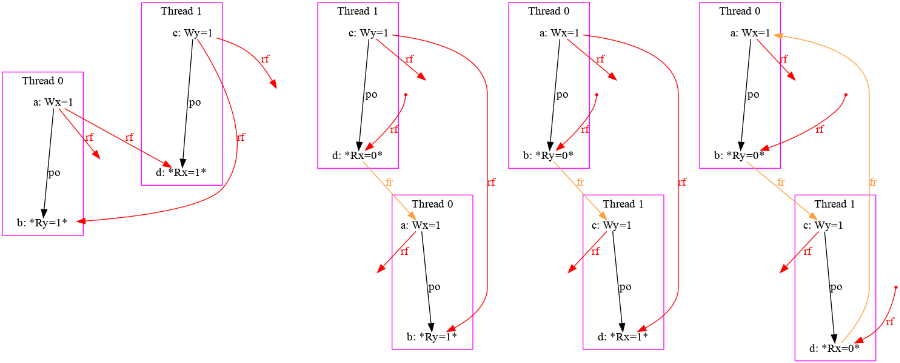


Figure: The four candidate executions allowed under x86-TSO

Weak memory model-aware analysis

Testing candidate executions

{ x=0; y=0; }	
P	Q
p_0 : $x \leftarrow 1$	q_0 : $y \leftarrow 1$
p_1 : $r_p \leftarrow y$	q_1 : $r_q \leftarrow x$
exists $(r_p = 0 \wedge r_q = 0)$	

SC model:

...

```
fr = (rf^-1; co)
acyclic(fr ∪ po)
```

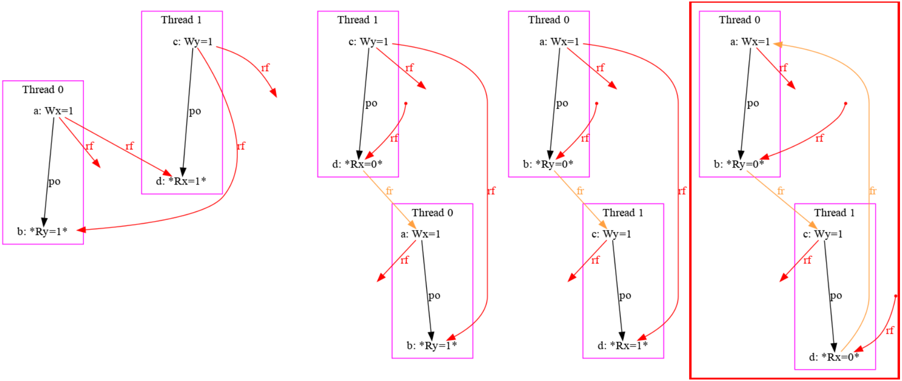


Figure: The four candidate executions allowed under x86-TSO

Portability analysis

The Porthos tool

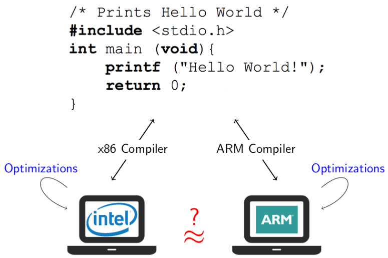


Figure: The illustration of the portability problem **[Porthos17slides]**

Portability analysis

The Porthos tool

```
/* Prints Hello World */
#include <stdio.h>
int main (void){
    printf ("Hello World!");
    return 0;
}
```

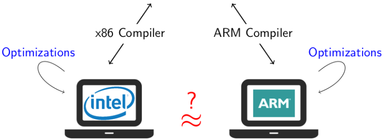


Figure: The illustration of the portability problem [**Porthos17slides**]

Definition (Portability [Porthos17a])

The program P is portable from the source memory model $\mathcal{M}_{\mathcal{S}}$ to the target one $\mathcal{M}_{\mathcal{T}}$ if $cons_{\mathcal{M}_{\mathcal{T}}}(P) \subseteq cons_{\mathcal{M}_{\mathcal{S}}}(P)$

- Portability as an SMT-based bounded reachability problem: $\phi = \phi_{CF} \wedge \phi_{DF} \wedge \phi_{\mathcal{M}_{\mathcal{T}}} \wedge \phi_{\neg \mathcal{M}_{\mathcal{S}}}$
- $\mathtt{SAT}(\phi) \implies$ portability bug

Portability analysis

The Porthos tool

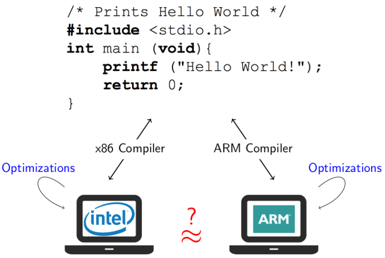


Figure: The illustration of the portability problem **[Porthos17slides]**

Definition (Portability **[Porthos17a]**)

The program P is portable from the source memory model $\mathcal{M}_{\mathcal{S}}$ to the target one $\mathcal{M}_{\mathcal{T}}$ if $cons_{\mathcal{M}_{\mathcal{T}}}(P) \subseteq cons_{\mathcal{M}_{\mathcal{S}}}(P)$

- Portability as an SMT-based bounded reachability problem: $\phi = \phi_{CF} \wedge \phi_{DF} \wedge \phi_{\mathcal{M}_{\mathcal{T}}} \wedge \phi_{\neg \mathcal{M}_{\mathcal{S}}}$
- $\mathtt{SAT}(\phi) \implies$ portability bug

Portability analysis

Porthos v1: An input example

```
{ flag0, flag1, turn }

thread t0 {
  while true {
    a <- 1;
    b <- 0;
    flag0.store(_rx, a);
    f1 = flag1.load(_rx);
    while (f1 == 1) {
      t1 = turn.load(_rx);
      if (t1 != 0) {
        flag0.store(_rx,b);
        t1 = turn.load(_rx);
        while (t1 != 0) {
          t1 = turn.load(_rx)
        };
        flag0.store(_rx,a)
      }
    }
  }
}

thread t1 {
  ...
}

exists turn=10,
```

Portability analysis

Porthos v1: An input example

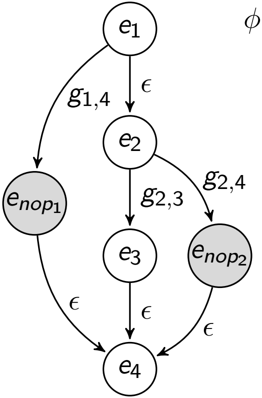
```
{ flag0, flag1, turn }

thread t0 {
  while true {
    a <- 1;
    b <- 0;
    flag0.store(_rx, a);
    f1 = flag1.load(_rx);
    while (f1 == 1) {
      t1 = turn.load(_rx);
      if (t1 != 0) {
        flag0.store(_rx,b);
        t1 = turn.load(_rx);
        while (t1 != 0) {
          t1 = turn.load(_rx)
        };
        flag0.store(_rx,a)
      }
    }
  }
}

thread t1 {
  ...
}

exists turn=10,
```

Encoding for the control-flow: An example



$$
\begin{aligned}
\phi_{CF} = & [\mathsf{x}(e_2) \Rightarrow \mathsf{x}(e_1)] \\
& \wedge [\mathsf{x}(e_3) \Rightarrow \mathsf{x}(e_2)] \\
& \wedge [\mathsf{x}(e_{nop_1}) \Rightarrow \mathsf{x}(e_1)] \\
& \wedge [\mathsf{x}(e_{nop_2}) \Rightarrow \mathsf{x}(e_2)] \\
& \wedge [\mathsf{x}(e_4) \Rightarrow (\mathsf{x}(e_{nop_1}) \vee \mathsf{x}(e_3) \vee \mathsf{x}(e_{nop_2}))] \\
& \wedge [\mathsf{x}(e_{nop_1}) \wedge \mathsf{x}(e_1) \Rightarrow g_{1,4}] \\
& \wedge [(\mathsf{x}(e_3) \wedge \mathsf{x}(e_2)) \Rightarrow g_{2,3}] \\
& \wedge [(\mathsf{x}(e_{nop_2}) \wedge \mathsf{x}(e_2)) \Rightarrow g_{2,4}] \\
& \wedge \neg[\mathsf{x}(e_2) \wedge \mathsf{x}(e_{nop_1})] \\
& \wedge \neg[\mathsf{x}(e_3) \wedge \mathsf{x}(e_{nop_2})]
\end{aligned}
$$

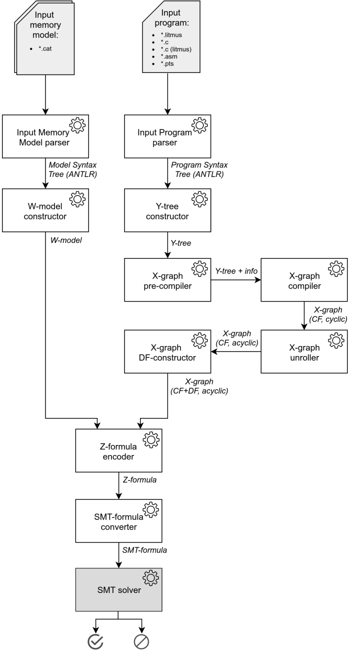
Figure: Example of encoding for the control-flow of the event-flow graph

The input language

The input language parser used by `Porthos` suffered from several disadvantages:

- it contained the parser code inlined directly into the grammar (hardly maintainable);
- the semantics of operations and kinds of variables (global or shared) were determined syntactically (4 different types of assignment:'`=`', '`:=`', '`<-`' and '`<:-`', each for different kinds of arguments);
- restricted syntax for expressions.
- In contrast, `PorthosC` uses the full C language grammar of proposed in the C11 standard **[jtc2011sc22]** and the visitor that converts the ANTLR grammar to the AST (`Y-tree`).

Architecture



The X-graph internal representation

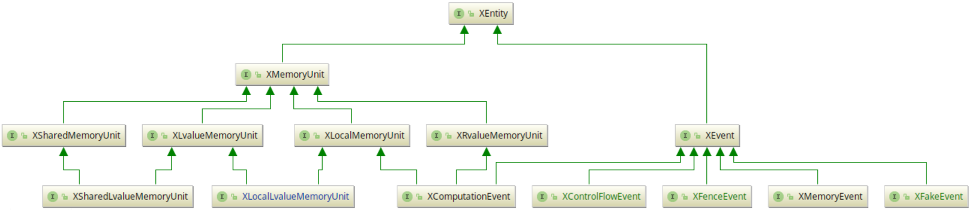


Figure: The inheritance tree of main X-graph interfaces

The X-graph compiler

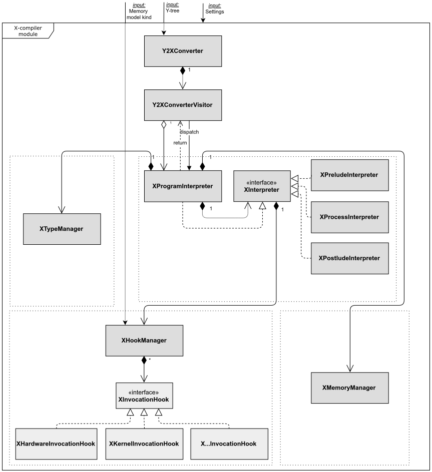


Figure: Main components of the X-compilation processing unit

X-graph unrolling

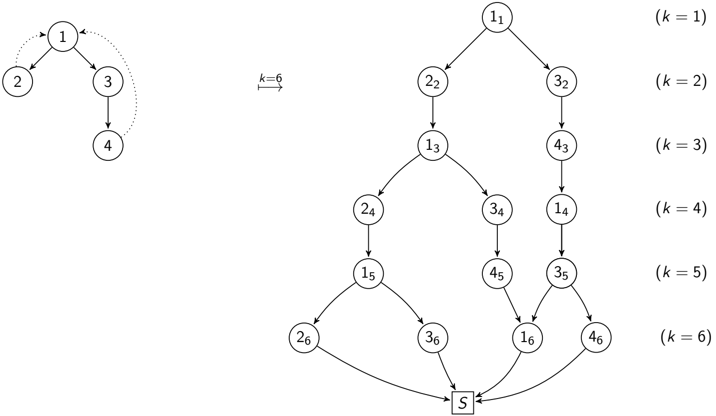


Figure: Example of the flow graph unrolling up to bound $k = 6$

Evaluation

Much better.
[to be done]

Summary

- The general framework for memory model-aware analysis was implemented in `PorthosC`;
- The input language has been extended;
- The old architecture of `Porthos` has been analysed and considered while designing the new architecture for `PorthosC`;
- to be done: more

Bibliography I