A Deep Dive into

seL4’s Binary Verification Story

Nick Spinale <nick@nickspinale.com>
seL4 Summit
September 5th, 2025

Introduction

Completed in 2013

Translation Validation for a Verified OS Kernel

Thomas Sewell
NICTA & UNSW, Sydney, Australia
thomas.sewell@nicta.com.au

Magnus Myreen
Cambridge University, UK
magnus.myreen@cl.cam.ac.uk

Gerwin Klein
NICTA & UNSW, Sydney, Australia
gerwin.klein@nicta.com.au

https://sel4.systems/Research/pdfs/translation-validation-verified-os-kernel.pdf

Thomas Sewell's 2017 PhD thesis

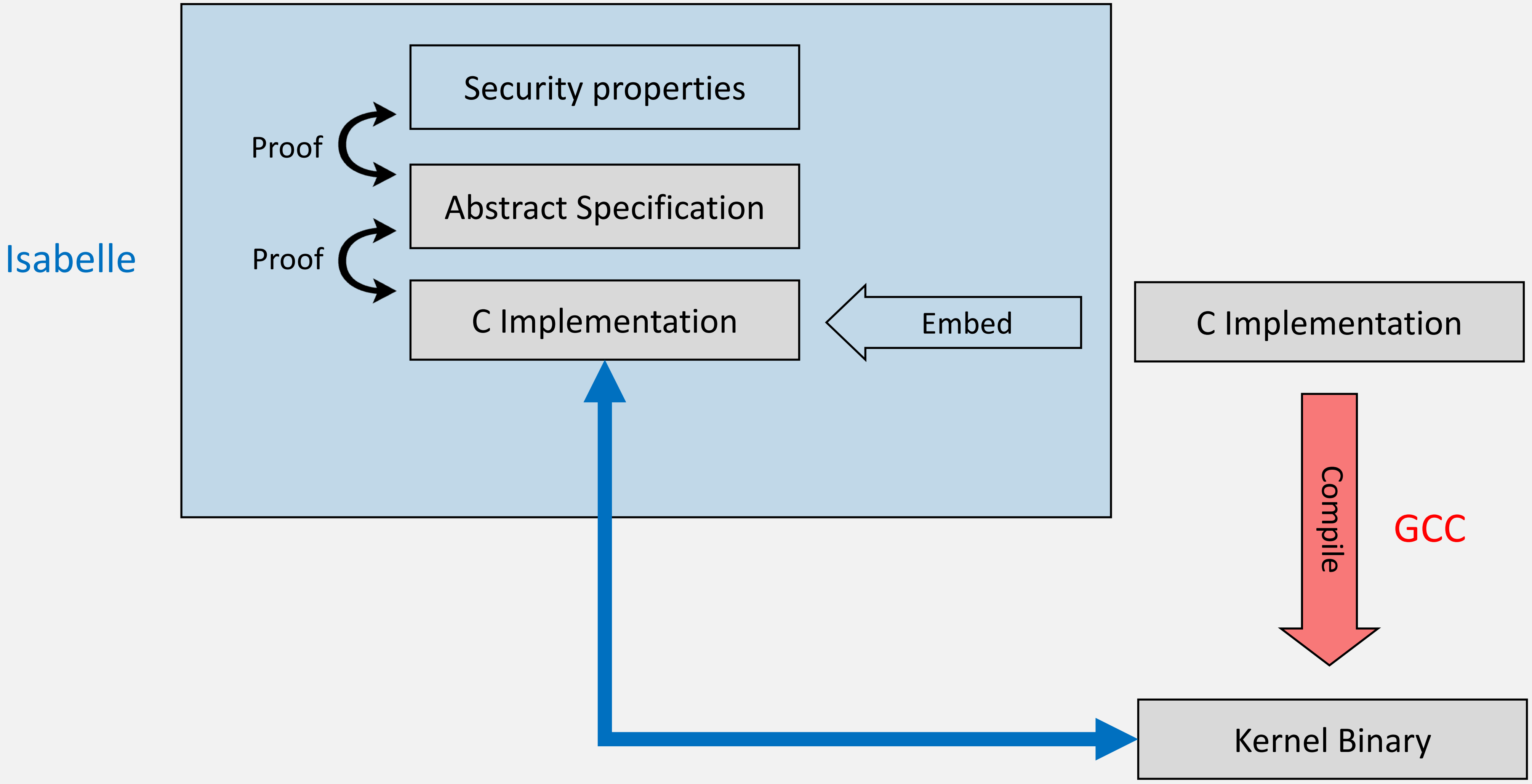
TRANSLATION VALIDATION FOR VERIFIED, EFFICIENT AND TIMELY OPERATING SYSTEMS

Thomas Sewell

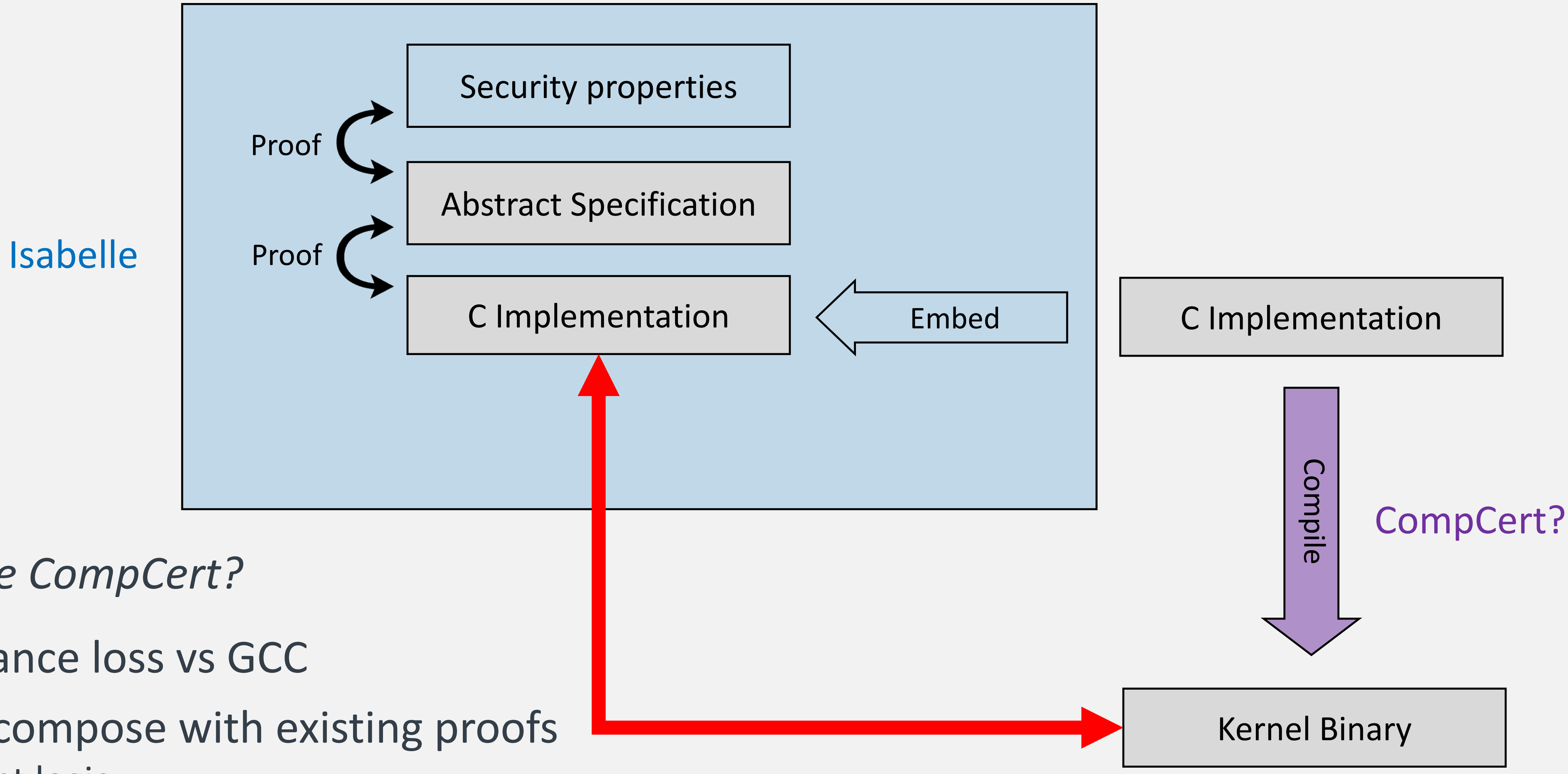
https://trustworthy.systems/publications/papers/Sewell%3Aphd.pdf

Colias Group

Context: The seL4 proofs



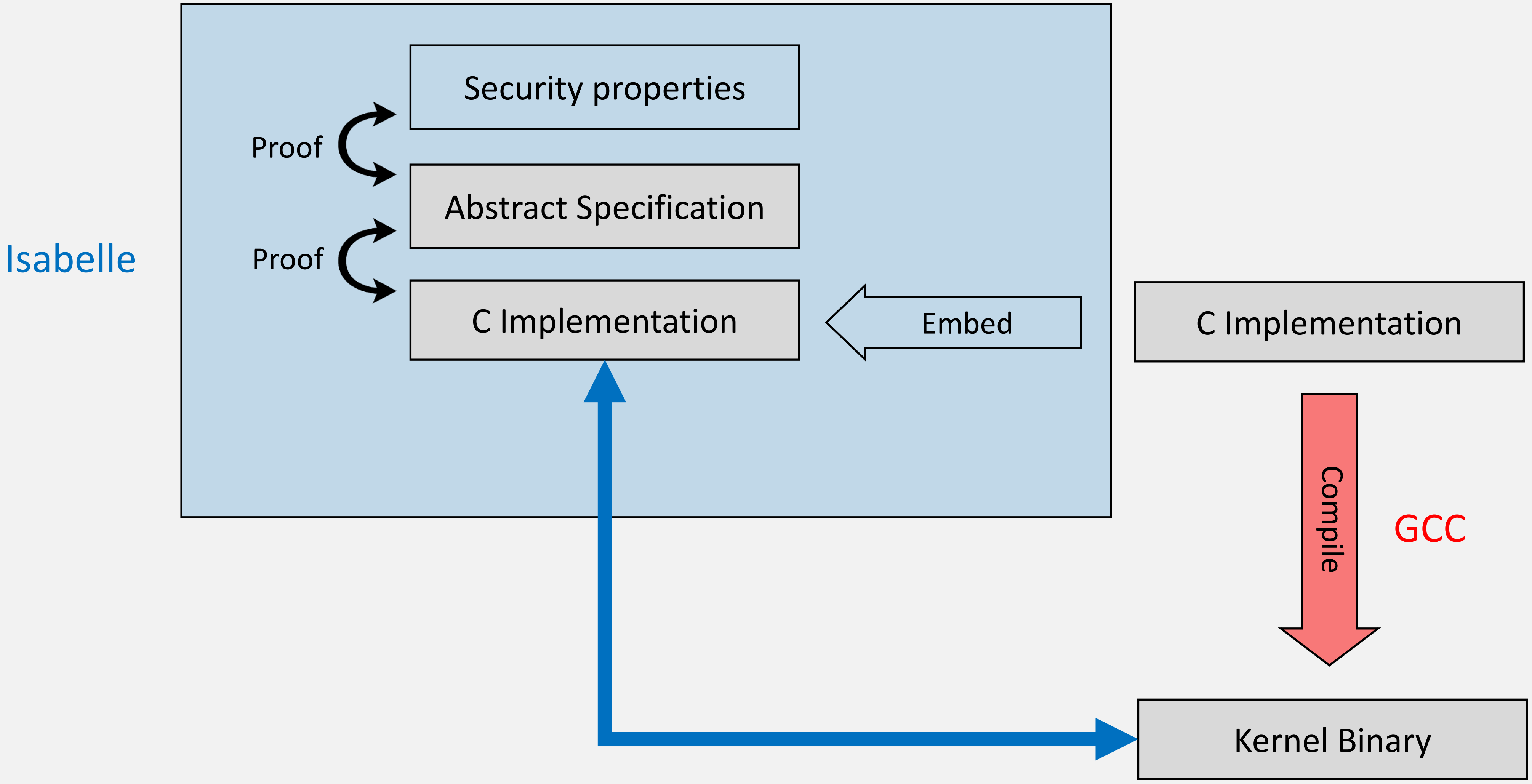
Verified compilation?



Why not use CompCert?

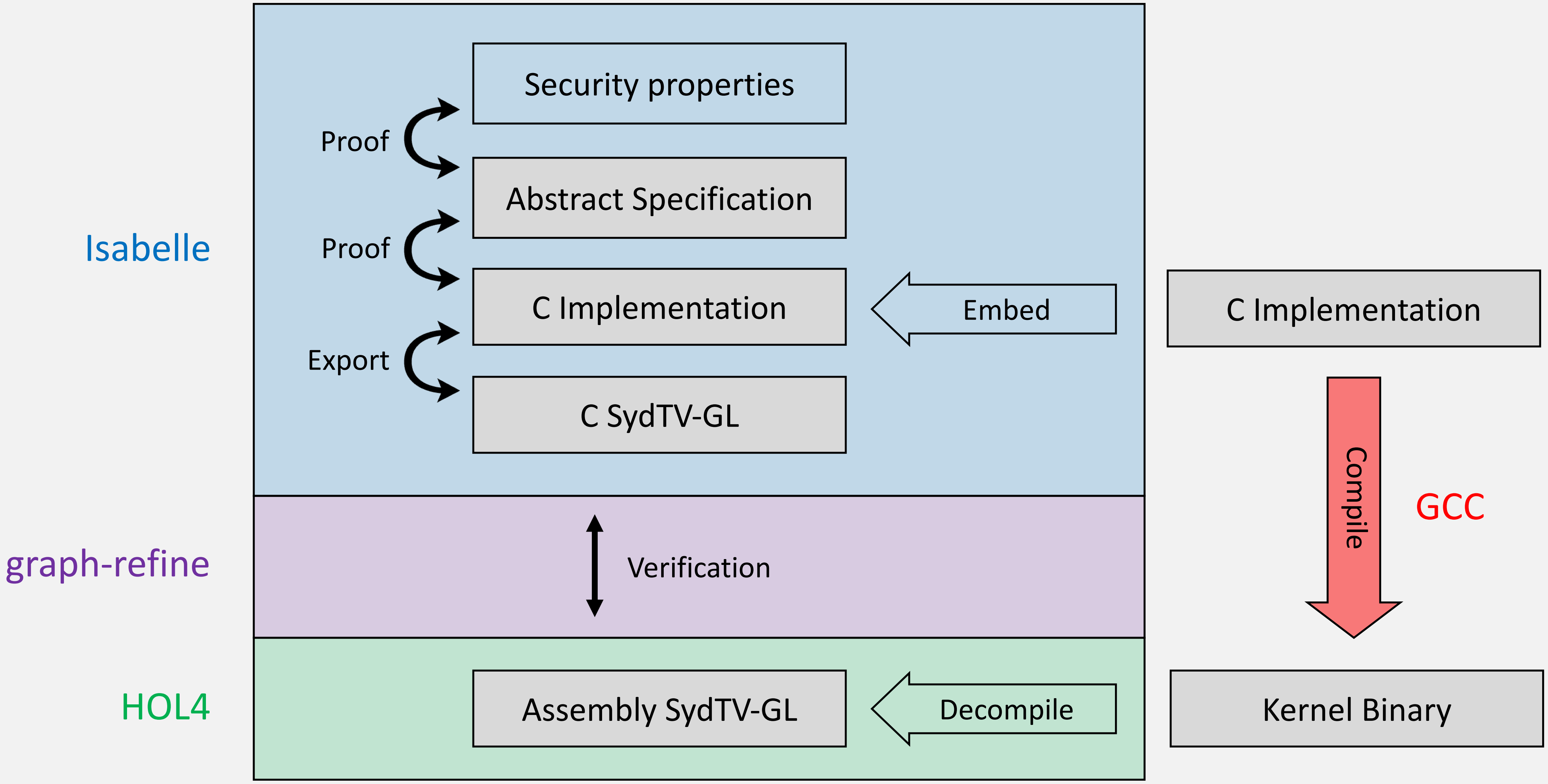
- Performance loss vs GCC
- Doesn't compose with existing proofs
  - Different logic
  - Different C semantics

~~Verified compilation~~ Translation validation



Translation validation using SydTV-GL

(Sydney Translation Validation
(graph-lang)
Graph Language)



SydTV-GL

SydTV-GL program:

```
Function <name> <input vars> <output vars>
    ...

Function <name> <input vars> <output vars>
    ...

Function <name> <input vars> <output vars>
    ...

...
```

SydTV-GL

SydTV-GL function:

```
Function <name> <input vars> <output vars>
    1 <node...>
    2 <node...>
    3 <node...>
    ...
    EntryPoint <entrypoint node id>
```

Control flow graph where nodes can:

- Assign expressions to variables
- Branch according expressions
- Call other functions

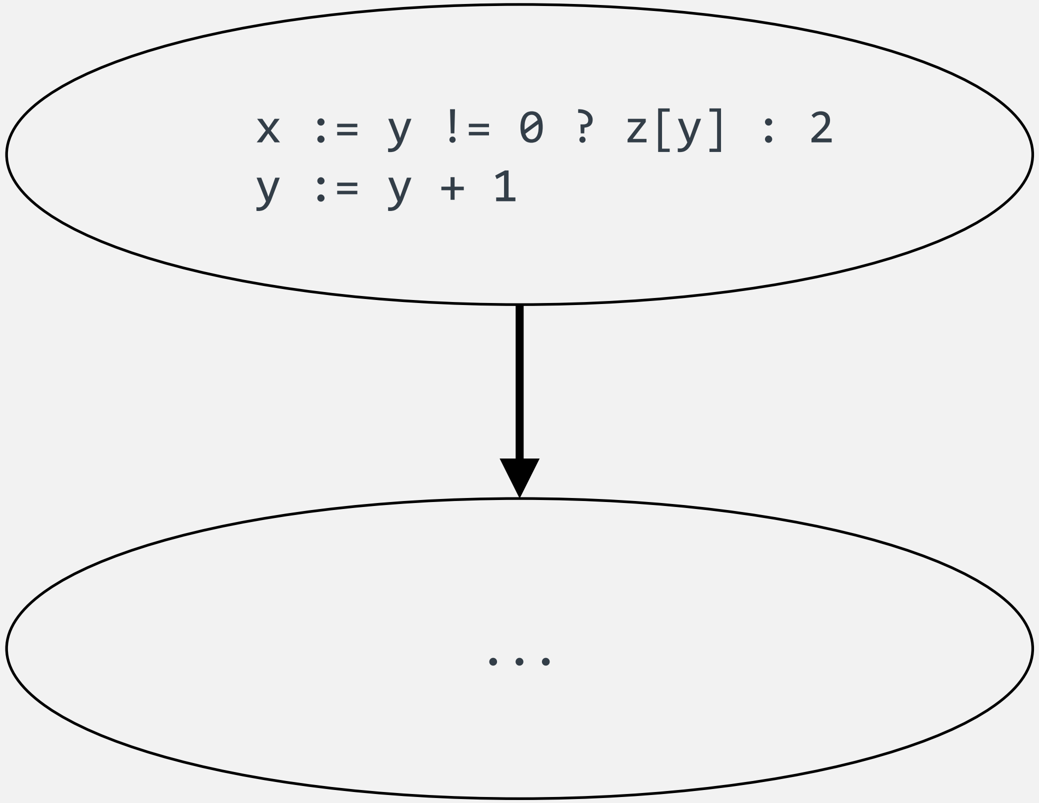
Special nodes: `Ret` and `Err`

Colias Group

SydTV-GL

SydTV-GL nodes:

Basic node

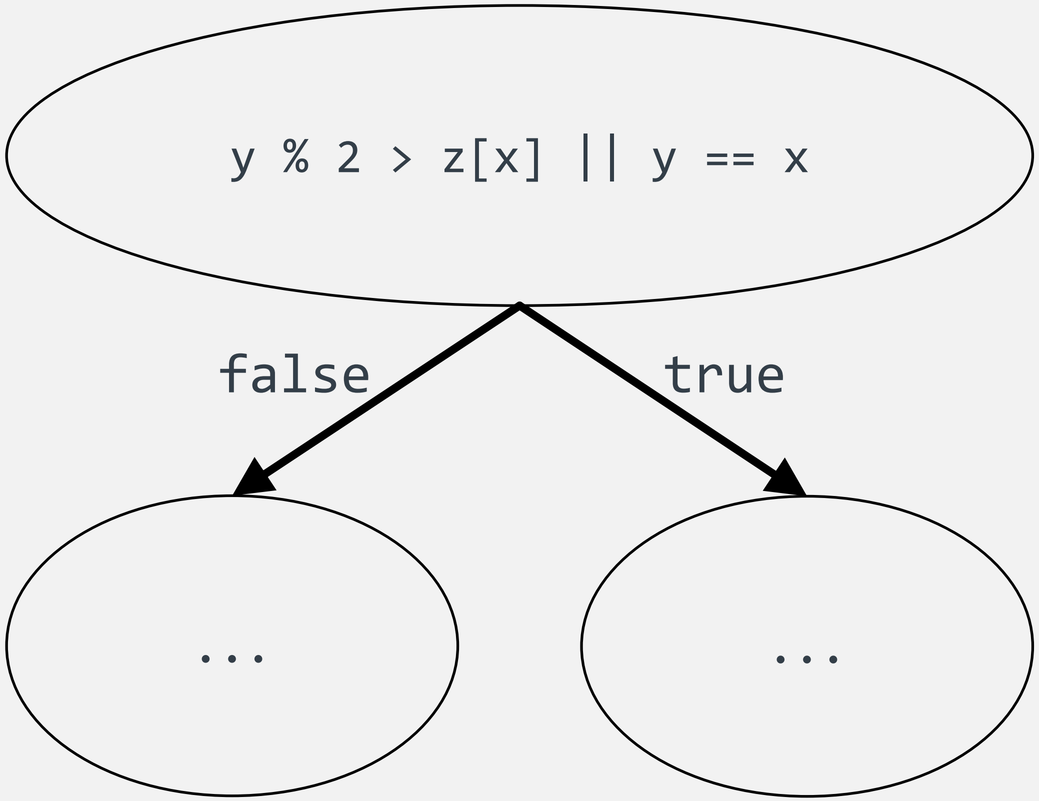


Colias Group

SydTV-GL

SydTV-GL nodes:

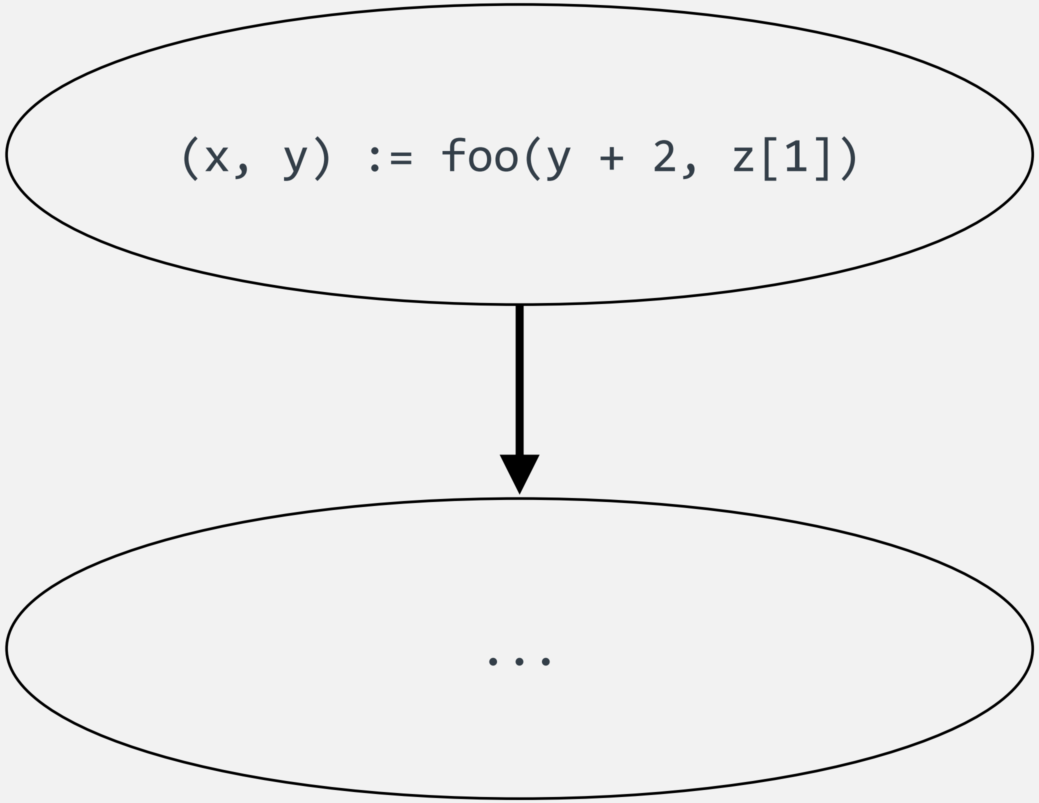
Cond node



SydTV-GL

SydTV-GL nodes:

Call node



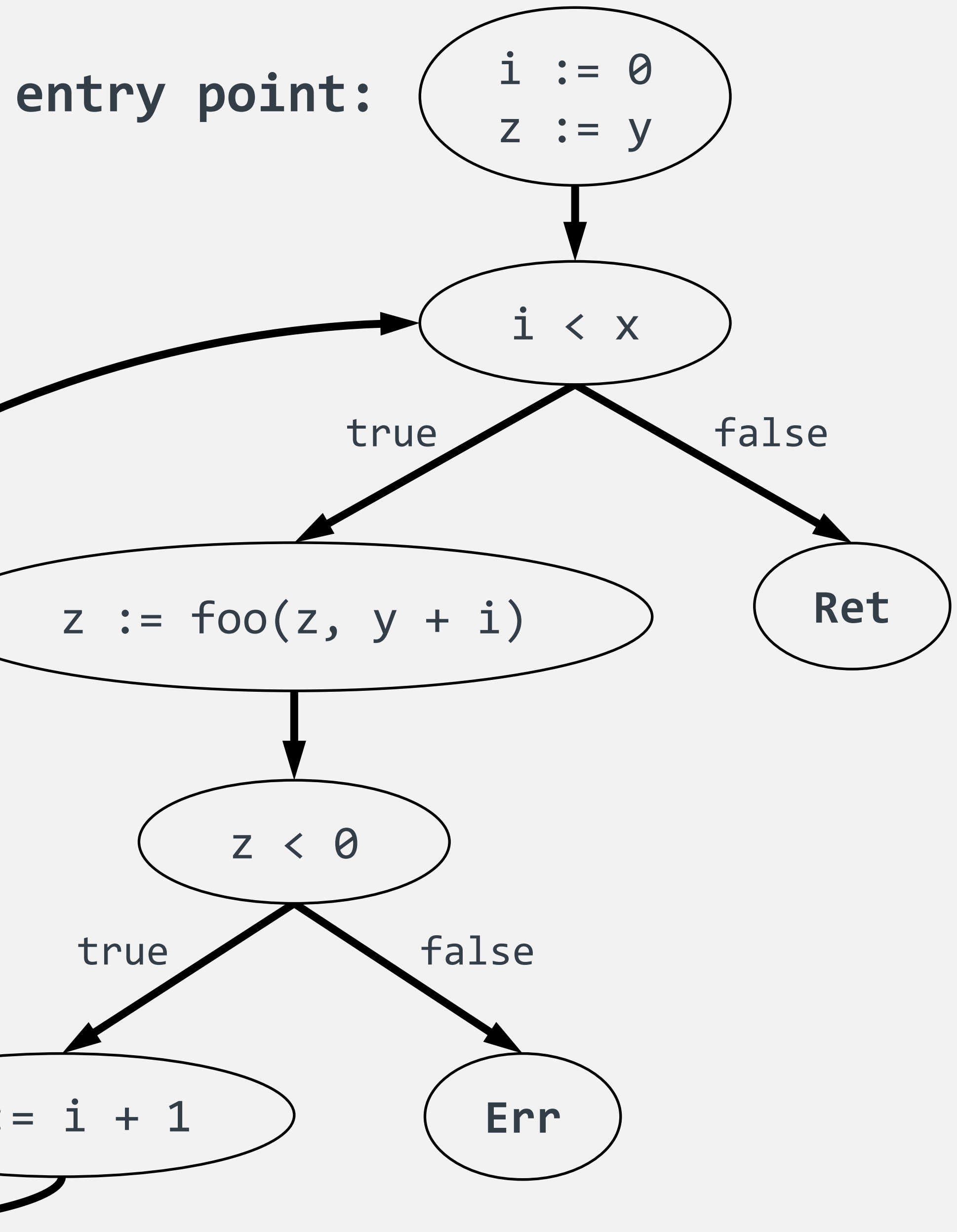
Colias Group

SydTV-GL

Example function:

inputs: x, y

outputs: z



Colias Group

Lowering C to SydTV-GL

For each lowered C function:

`inputs:` `C signature inputs`
`mem[]`

`outputs:` `C signature output (if present)`
`mem[]`

Straightforward translation of statements into SydTV-GL nodes

Lowering C to SydTV-GL

For each lowered C function:

```
 inputs: C signature inputs
         mem[]

outputs: C signature output (if present)
         mem[]
```

Straightforward translation of statements into SydTV-GL nodes, with simplifications:

- `if`/`switch`/`while`/`for` expressed in terms of `cond` nodes
- Local structs decomposed into fields
- Pointers to struct fields translated into offsets
- Global variable symbols translated into addresses

Lowering C to SydTV-GL

For each lowered C function:

`inputs:` `C signature inputs`
`mem[]`

`outputs:` `C signature output (if present)`
`mem[]`

Straightforward translation of statements into SydTV-GL nodes, with simplifications

Compiler assumptions expressed as assertions using `Err` node

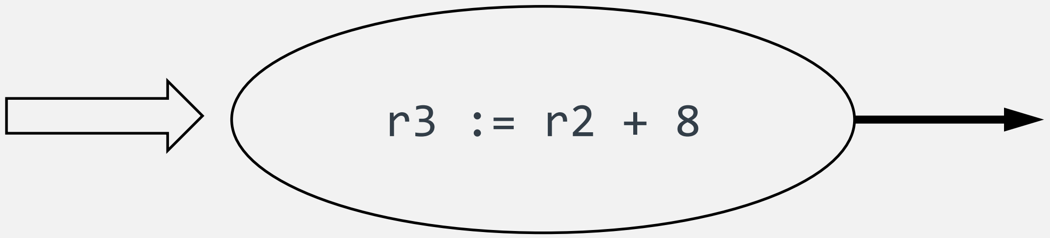
- Integer operations must not overflow
- Pointers accesses must be aligned and non-null
- Strict-aliasing rules

Lifting assembly to SydTV-GL

For each lifted assembly function: **`inputs/outputs:`** `r0, r1, r2, ..., r31`
`n, z, c, v`
`stack[], mem[]`

Straightforward translation of opcodes into SydTV-GL subgraphs

```
add r3, r2, #8
```

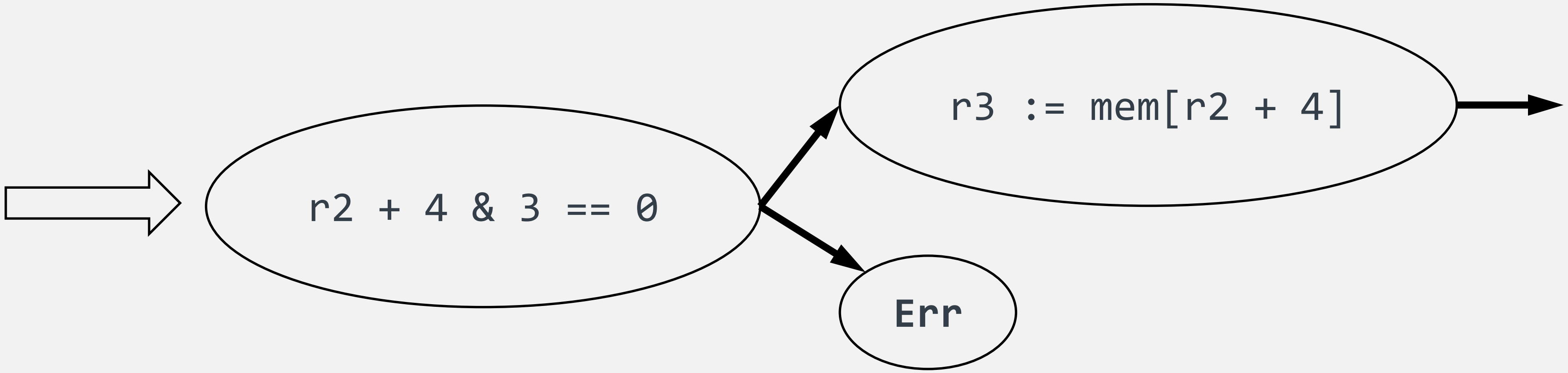


Lifting assembly to SydTV-GL

For each lifted assembly function: **`inputs/outputs:`** `r0, r1, r2, ..., r31`
`n, z, c, v`
`stack[], mem[]`

Straightforward translation of opcodes into SydTV-GL subgraphs

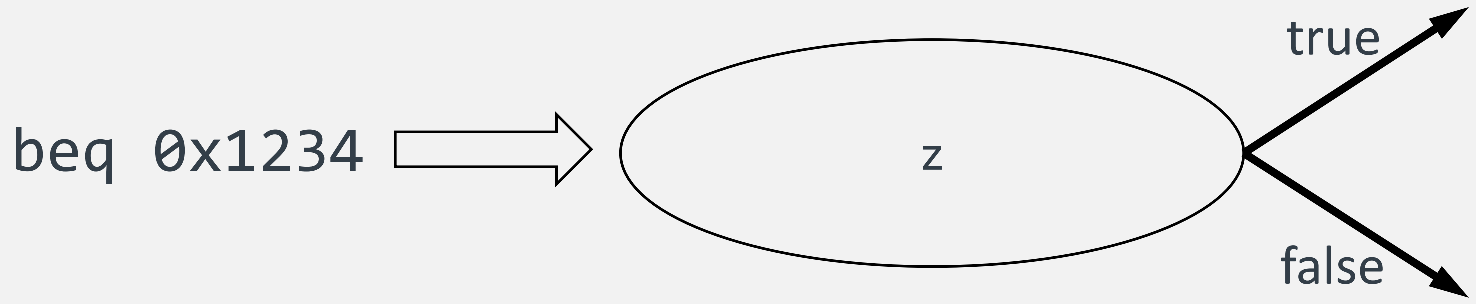
```
ldr r1, [r2, #4]
```



Lifting assembly to SydTV-GL

For each lifted assembly function: **`inputs/outputs:`** `r0, r1, r2, ..., r31`
`n, z, c, v`
`stack[], mem[]`

Straightforward translation of opcodes into SydTV-GL subgraphs



Lifting assembly to SydTV-GL

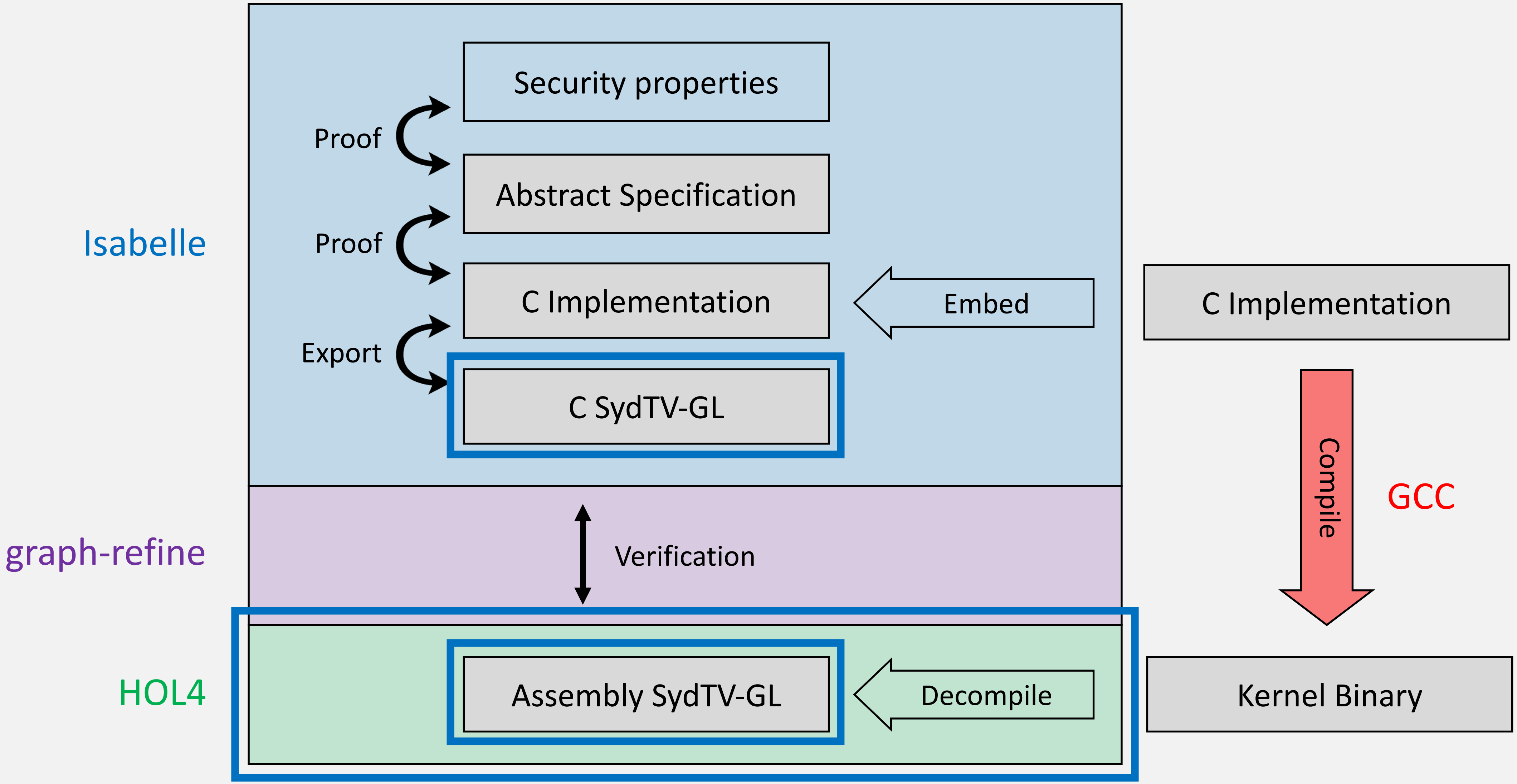
For each lifted assembly function: **inputs/outputs:** `r0, r1, r2, ..., r31`
`n, z, c, v`
`stack[], mem[]`

Straightforward translation of opcodes into SydTV-GL subgraphs

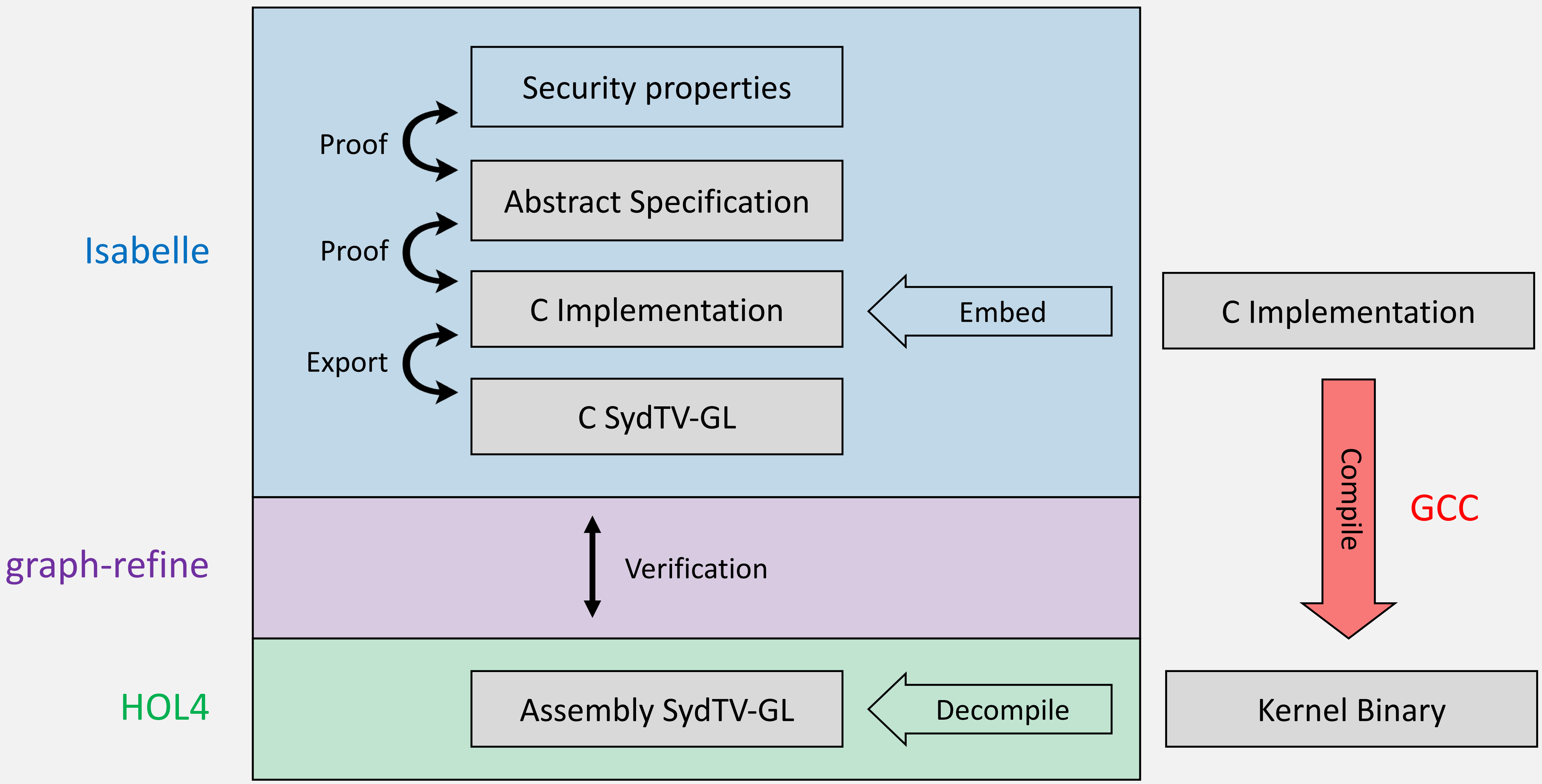
```
0x1234: bl 0x5678 <setThreadState>
```

```
(r0, r1, …, stack, mem) :=
setThreadState(r0, r1, …, r12, 0x1238, r14, …, stack, mem)
```

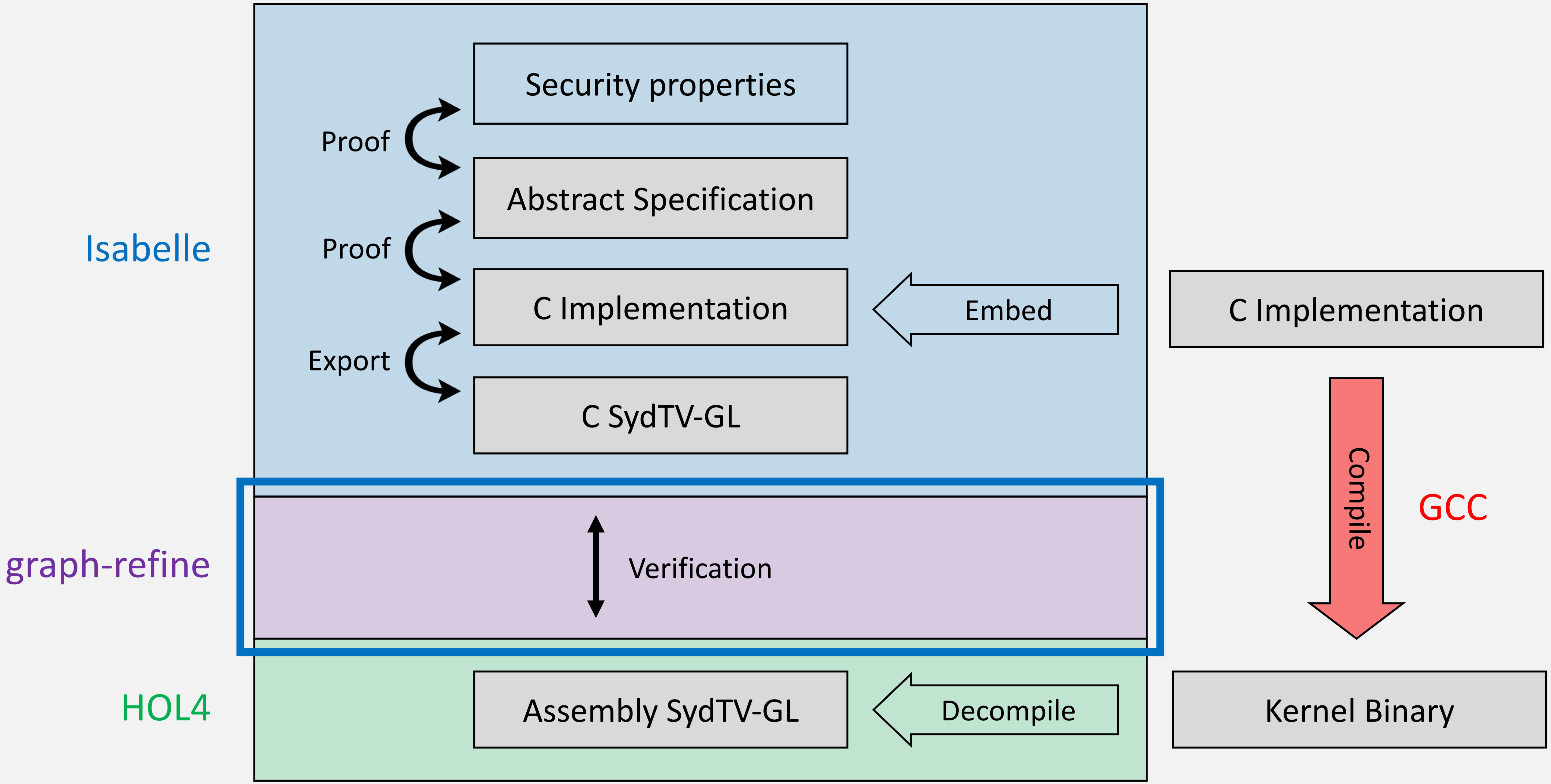
Translation validation using SydTV-GL



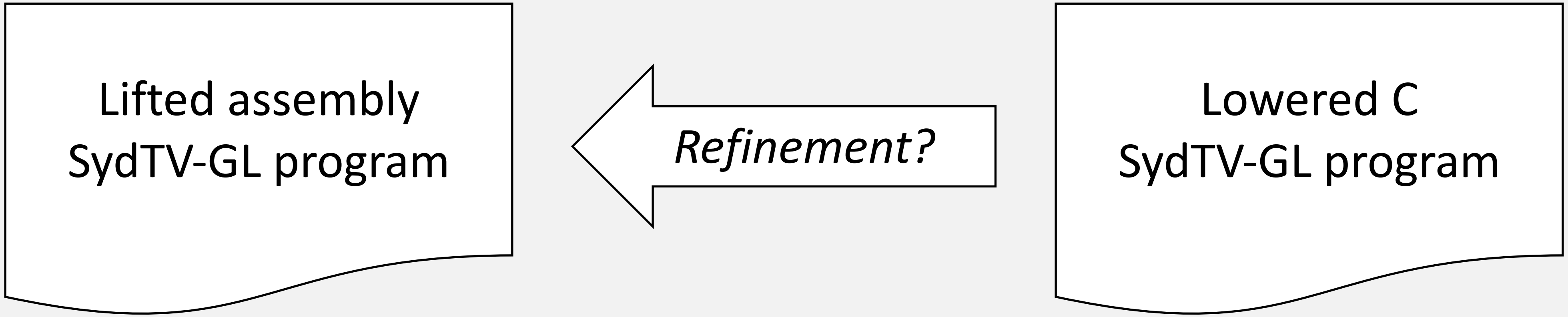
Translation validation using SydTV-GL



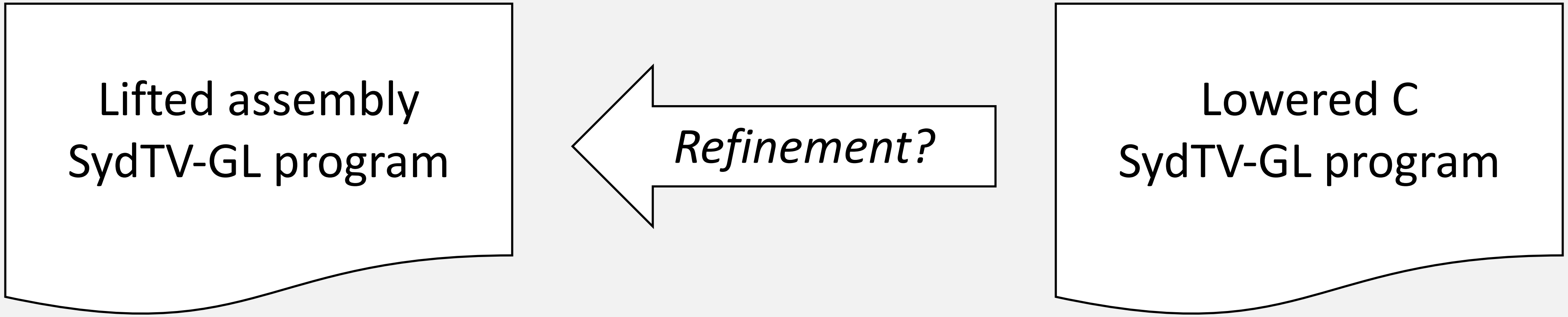
Comparing the two SydTV-GL programs



Comparing the two SydTV-GL programs

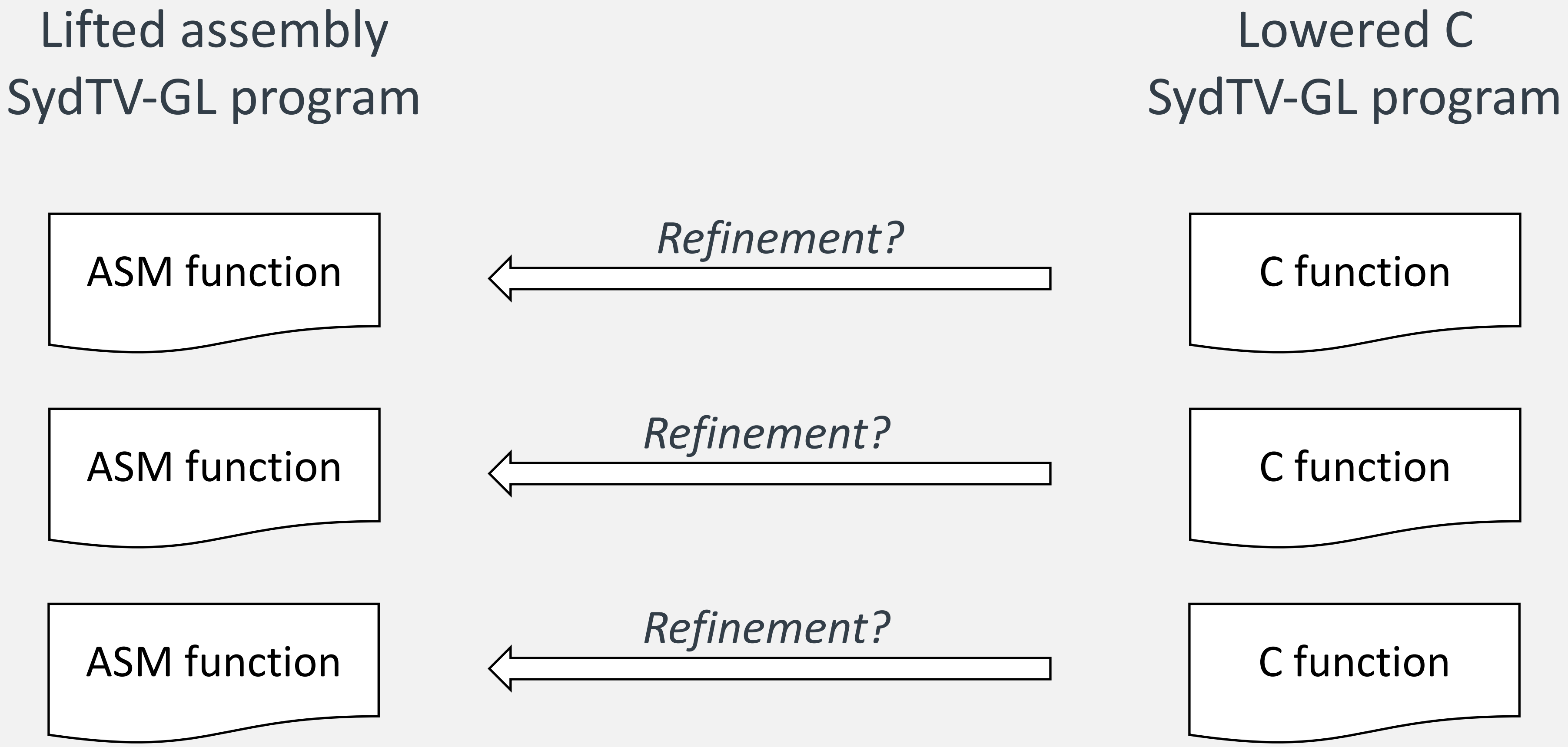


The refinement hypothesis



Colias Group

The refinement hypothesis



Colias Group

The refinement hypothesis

ASM function ⟸ *Refinement?* ⟸ C function

Per-function refinement hypothesis, loosely:

If the two functions are given **equivalent** input, then they return **equivalent** output.

↓

Equivalence as defined by the calling convention

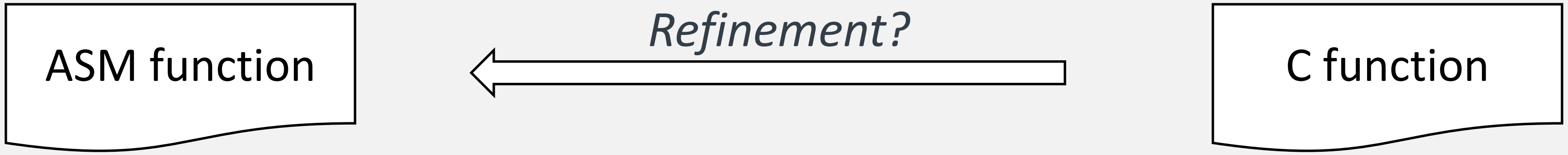

Procedure Call Standard for the Arm®
Architecture

2025Q1

Date of Issue: 07th April 2025

arm

The refinement hypothesis



if

- C inputs are mapped onto ASM registers and stack according to CC
- Memory inputs are equal
- C function does not reach `Err`
- CC ASM preconditions (e.g. stack alignment)

then

- C outputs are mapped onto ASM registers and stack according to CC
- Memory outputs are equal
- ASM function does not reach `Err`
- CC ASM invariants (e.g. callee-saved registers unchanged)

Recap

Verifying refinement

Verifying refinement: Leveraging an SMT solver

Satisfiability **M**odulo **T**heories

Colias Group

Verifying refinement: Leveraging an SMT solver

Satisfiability **M**odulo **T**heories

Verifying refinement: Leveraging an SMT solver

Satisfiability **M**odulo **T**heories

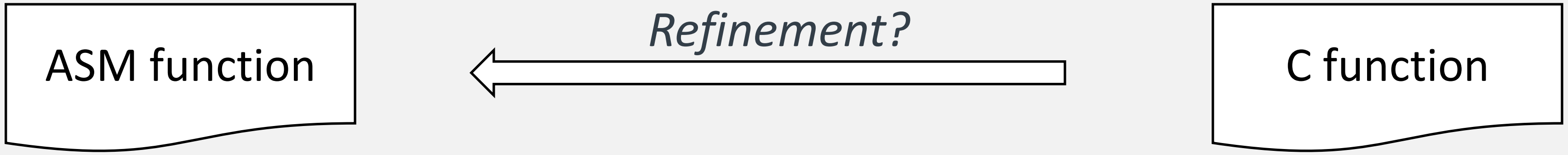
Colias Group

Verifying refinement: Leveraging an SMT solver

Interacting with the solver

```
> (declare-const p Bool)
> (assert (and p (not p)))
> (check-sat)
unsat
```

Verifying refinement: Leveraging an SMT solver



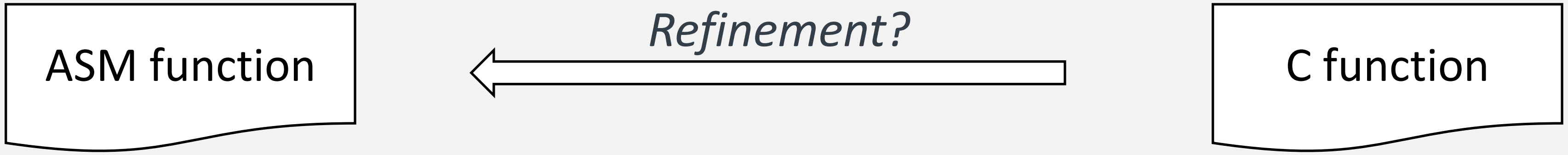
if

- C inputs are mapped onto ASM registers and stack according to CC
- Memory inputs are equal
- C function does not reach `Err`
- CC ASM preconditions (e.g. stack alignment)

then

- C outputs are mapped onto ASM registers and stack according to CC
- Memory outputs are equal
- ASM function does not reach `Err`
- CC ASM invariants (e.g. callee-saved registers unchanged)

Verifying refinement: Leveraging an SMT solver

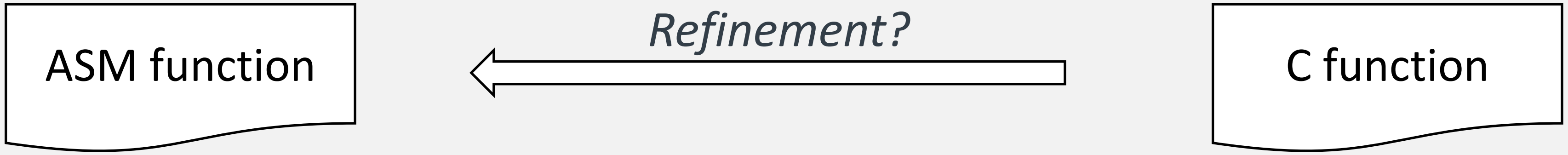


Our general approach

- Consider the execution of both functions simultaneously
- Declare all inputs as free symbols
- Assert the premises of the refinement hypothesis
- Assert some facts derived from the function bodies
- Assert that at least one of the conclusions of the refinement hypothesis is false
- Query satisfiability (unsatisfiable means refinement proven)

Colias Group

Verifying refinement: Simplest cases

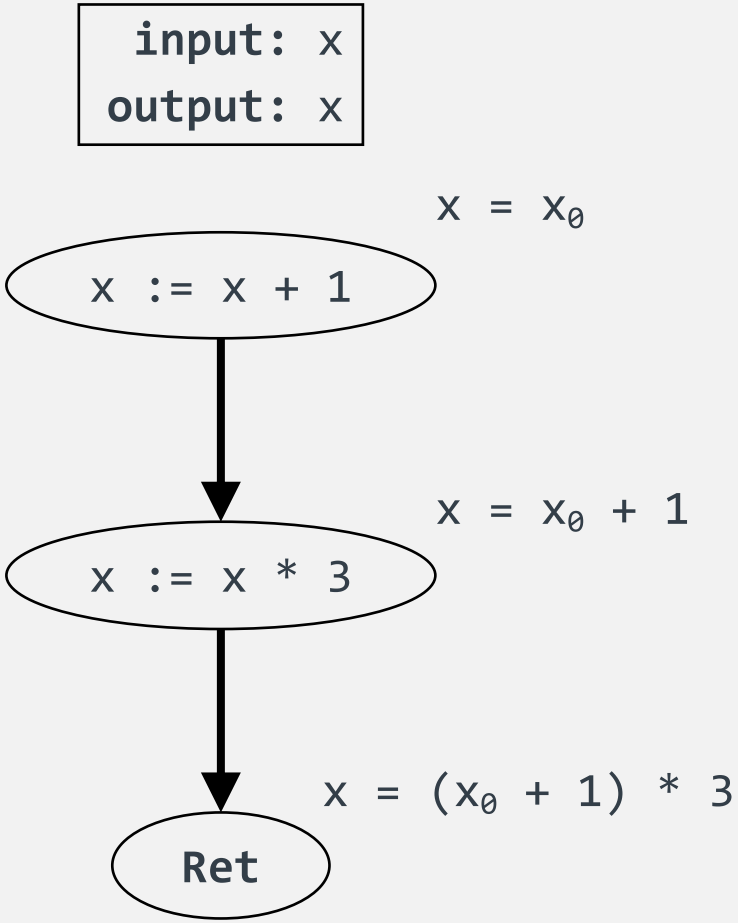


Our general approach

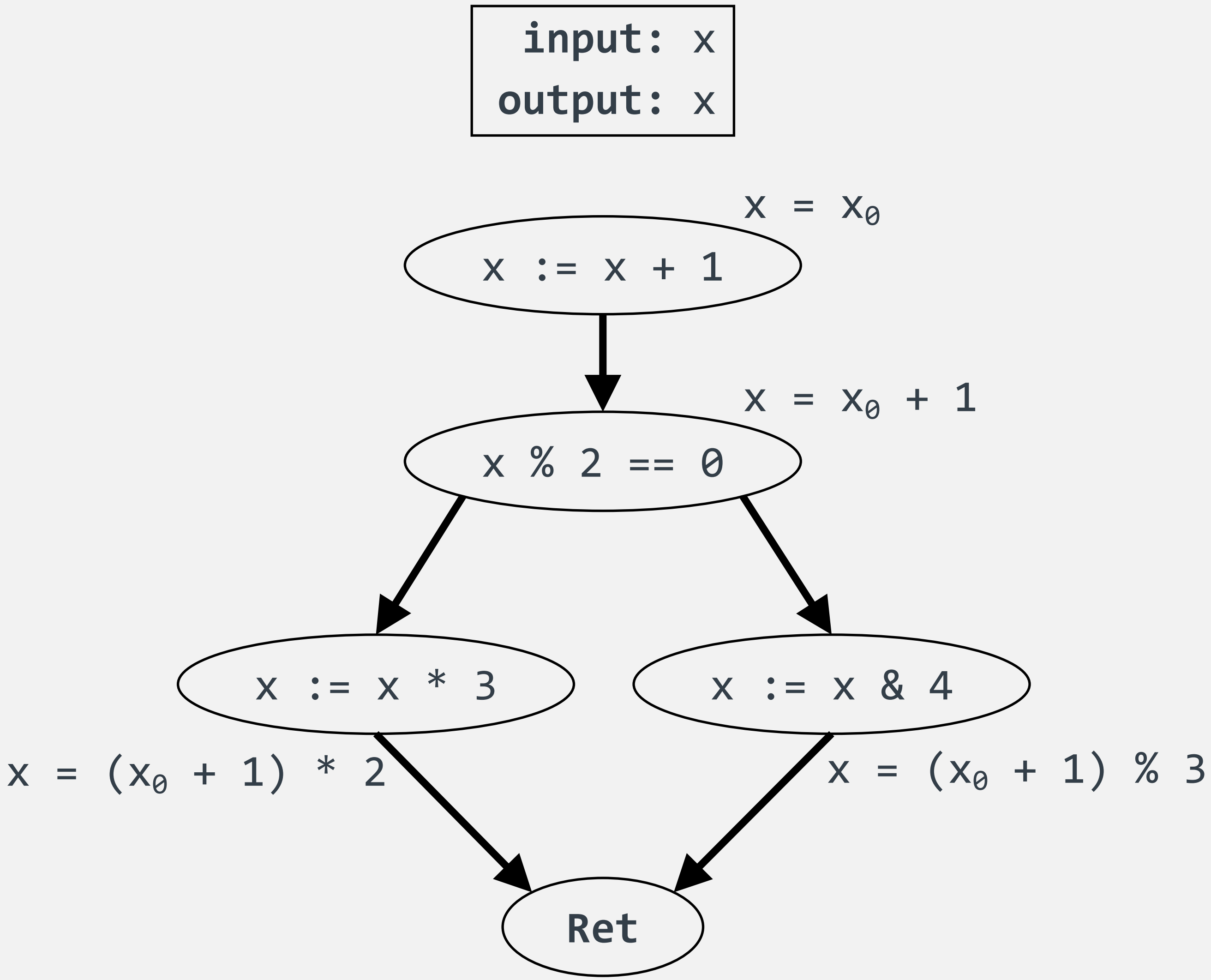
- Consider the execution of both functions simultaneously
- Declare all inputs as free symbols
- Assert the premises of the refinement hypothesis
- Assert some facts derived from the function bodies
- Assert that at least one of the conclusions of the refinement hypothesis is false
- Query satisfiability (unsatisfiable means refinement proven)

Colias Group

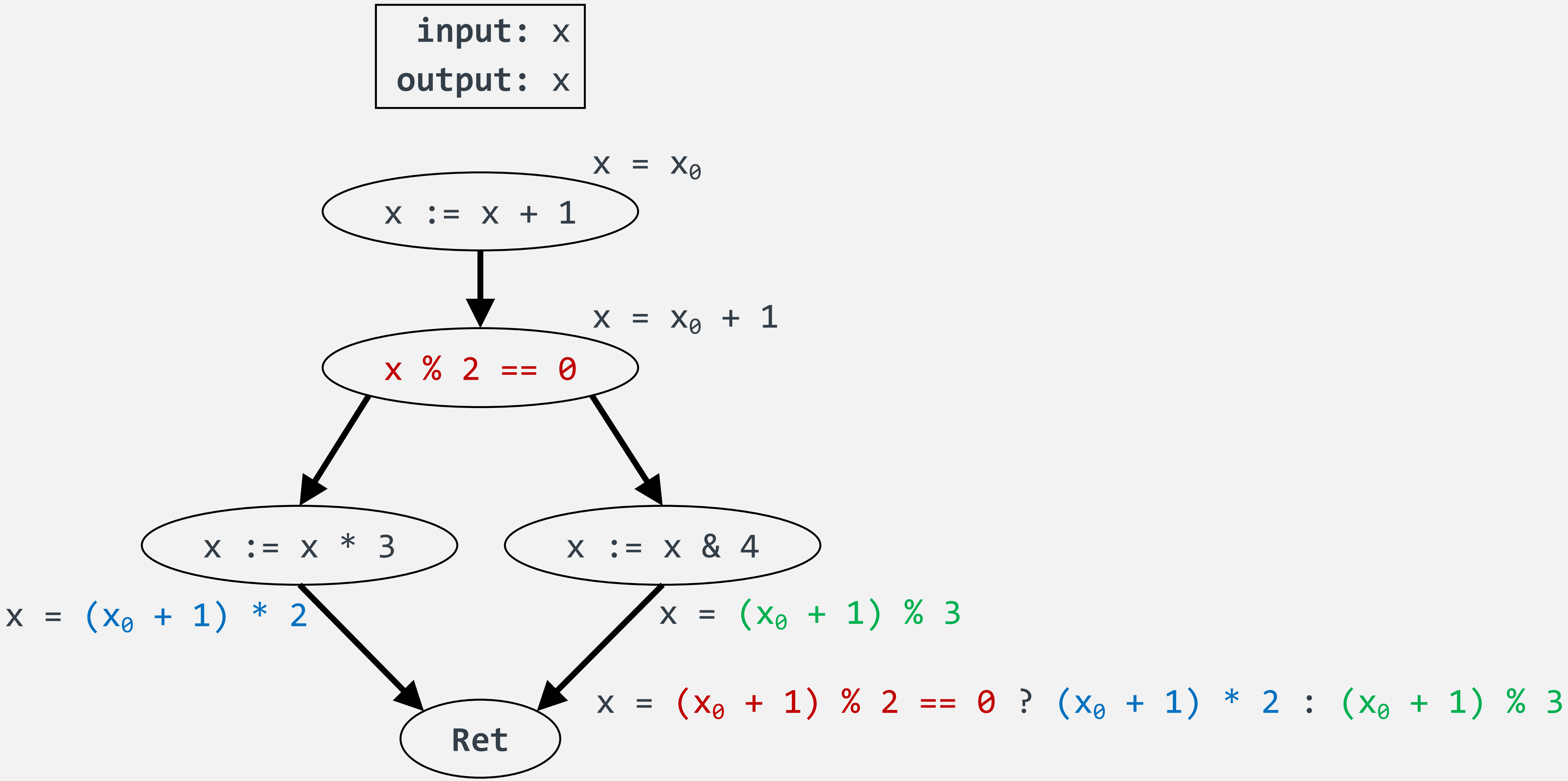
Verifying refinement: Simplest cases



Verifying refinement: Handling branches

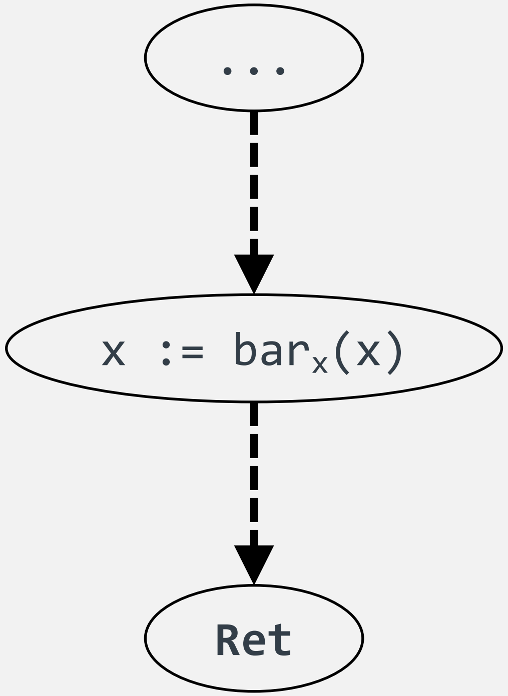


Verifying refinement: Handling branches



Verifying refinement: Handling function calls

foo_x
input: x
output: x



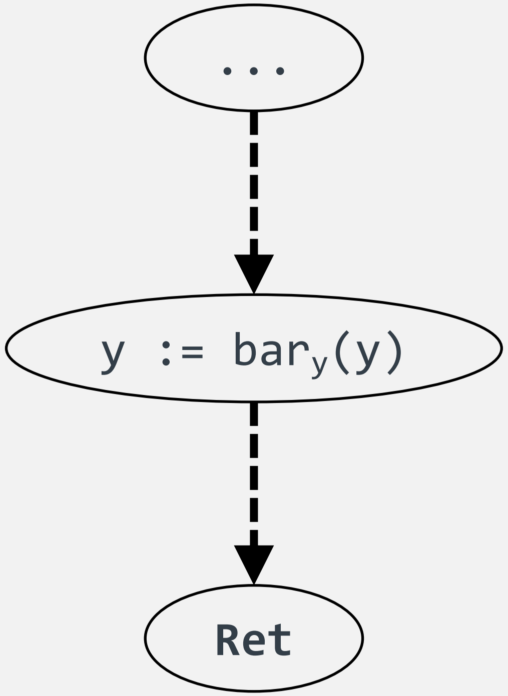
Assumptions

$bar_x \cong bar_y$

Goal

$foo_x \cong foo_y$

foo_y
input: y
output: y

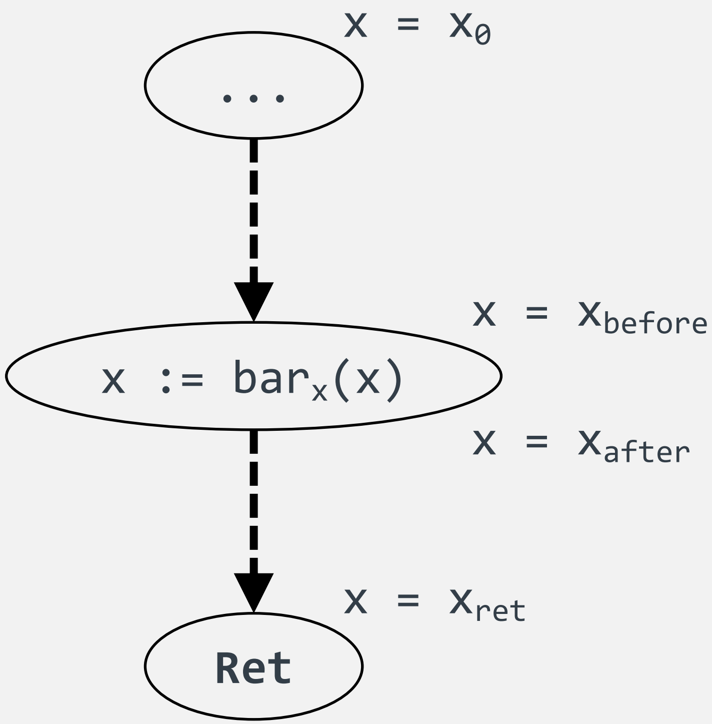


Verifying refinement: Handling function calls

foo_x

input: x

output: x



Assumptions

$x = y \longrightarrow \text{bar}_x(x) = \text{bar}_y(y)$

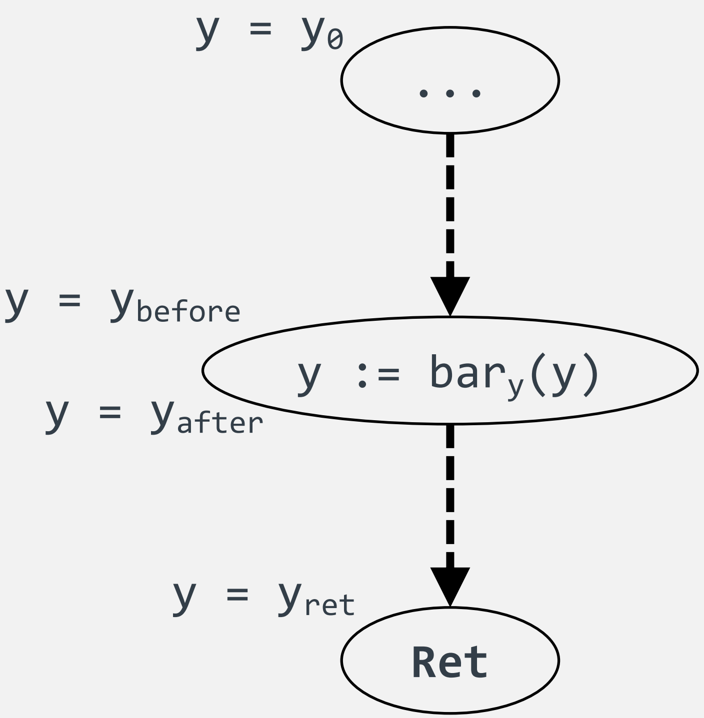
Goal

$x = y \longrightarrow \text{foo}_x(x) = \text{foo}_y(y)$

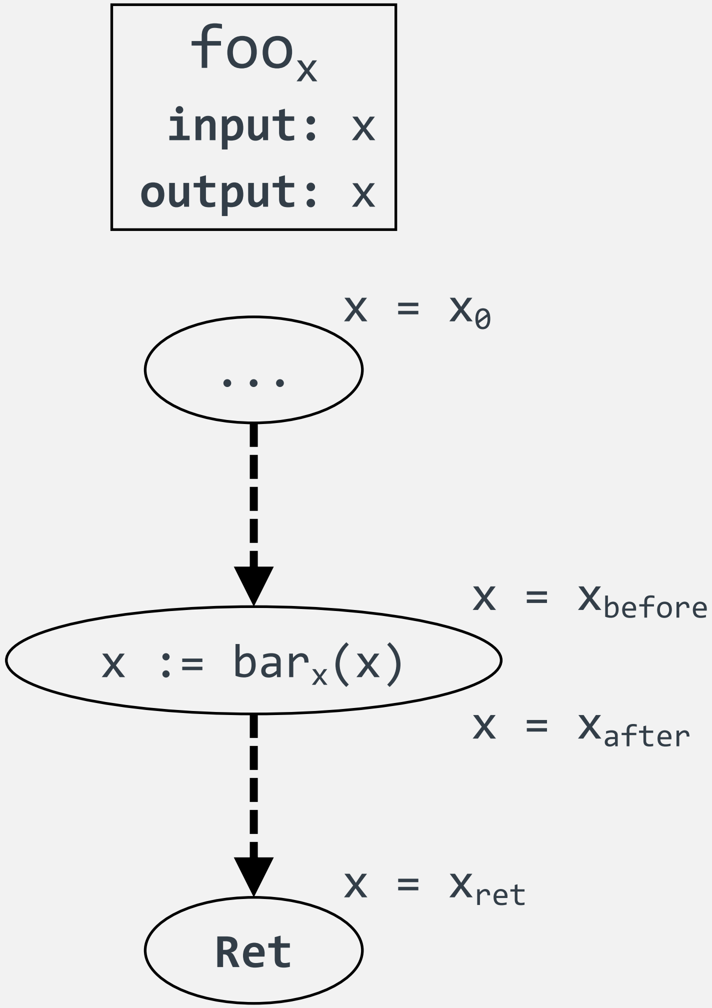
foo_y

input: y

output: y



Verifying refinement: Handling function calls



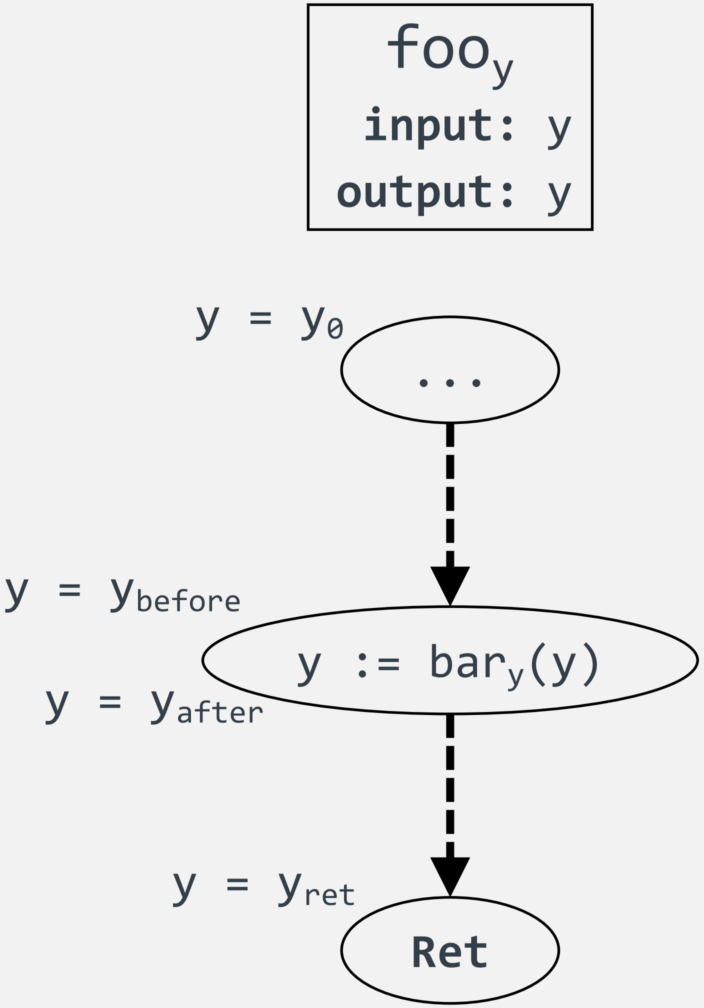
Assumptions

$x = y \longrightarrow bar_x(x) = bar_y(y)$

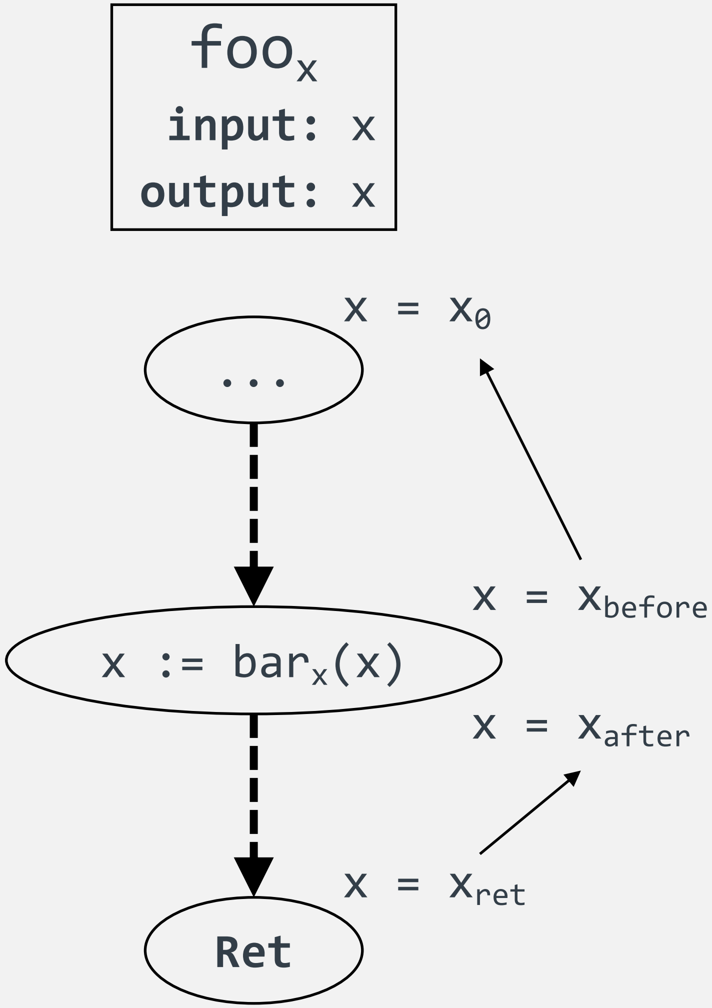
$x_0 = y_0$

Goal

$x_{ret} = y_{ret}$



Verifying refinement: Handling function calls



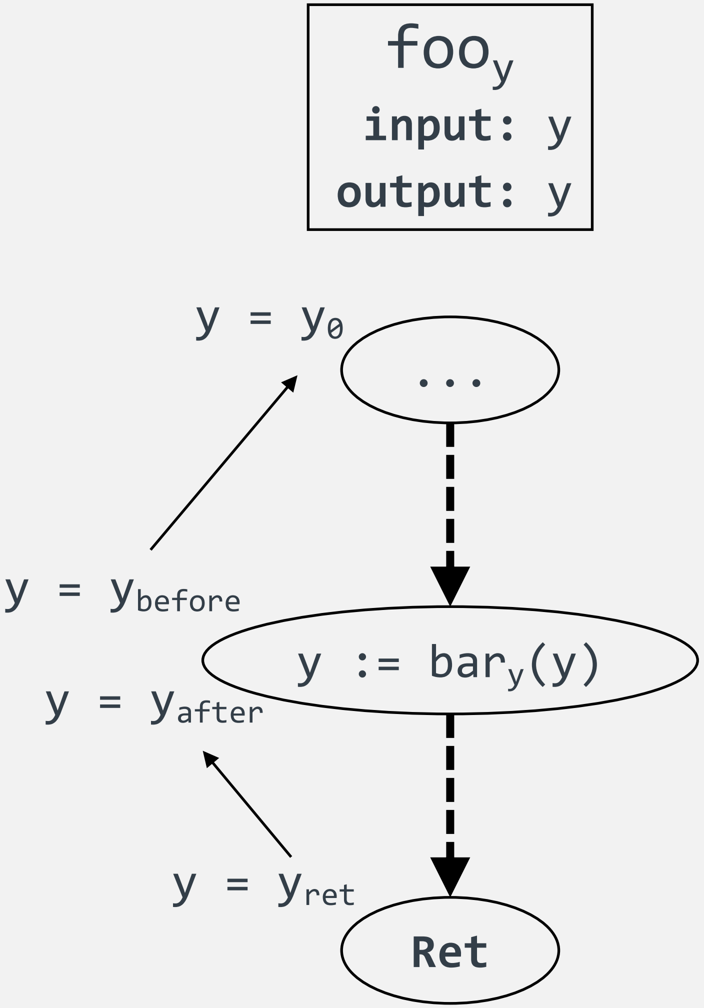
Assumptions

$x_{before} = y_{before} \longrightarrow x_{after} = y_{after}$

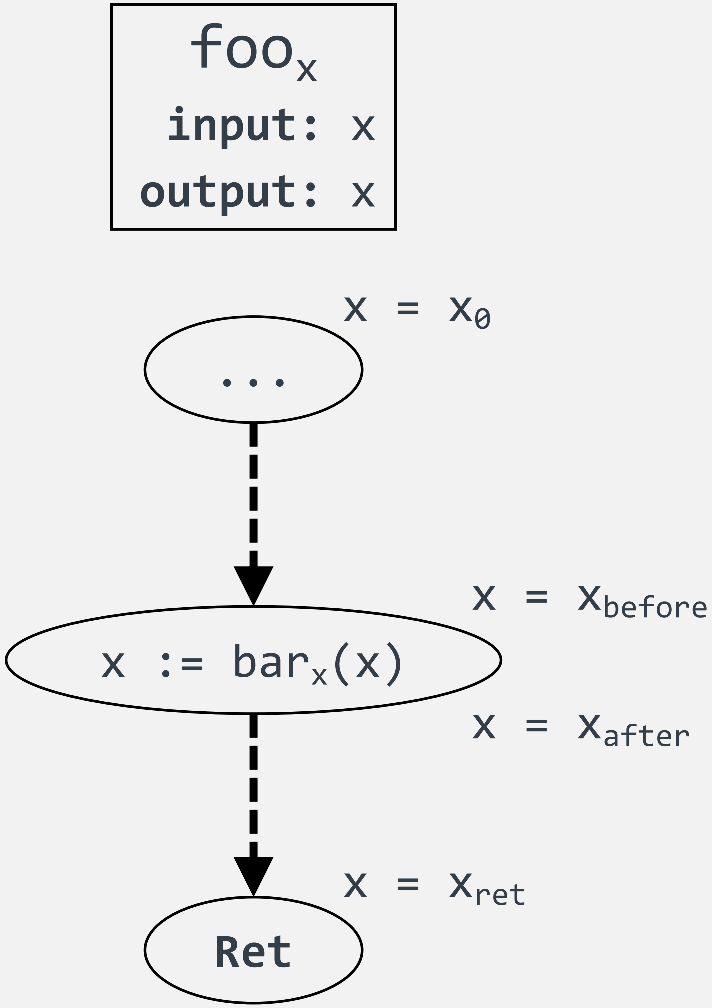
$x_0 = y_0$

Goal

$x_{ret} = y_{ret}$



Verifying refinement: Handling function calls



Assumptions

$$x_{before} = y_{before} \longrightarrow x_{after} = y_{after}$$

$$x_0 = y_0$$

$$x_{before} := \ldots\ x_0\ \ldots$$

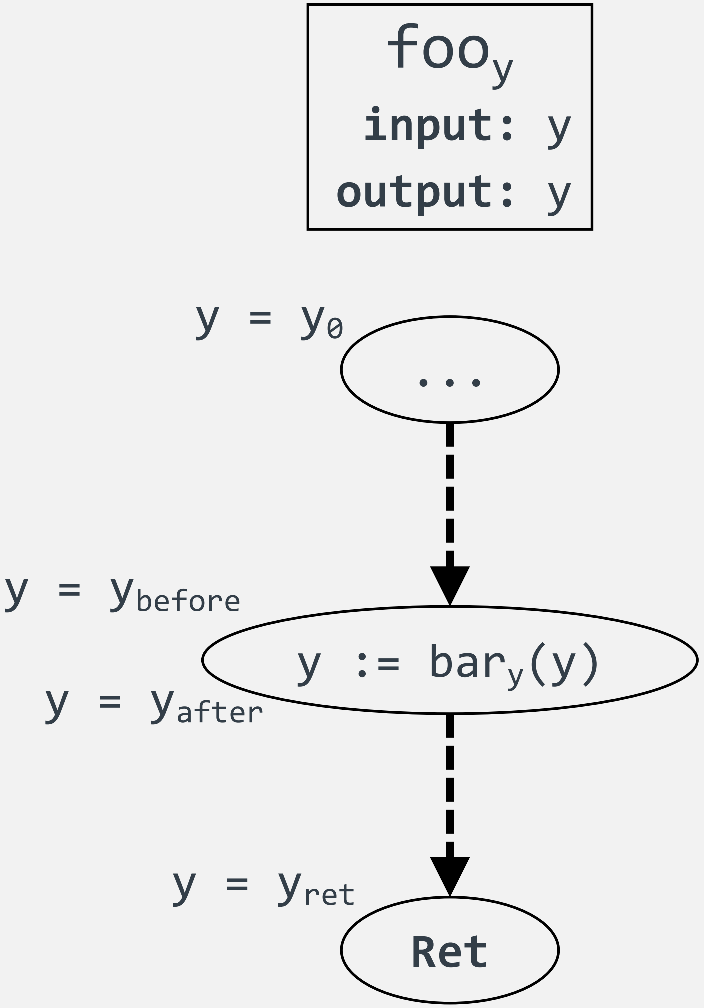
$$x_{ret} := \ldots\ x_{after}\ \ldots$$

$$y_{before} := \ldots\ y_0\ \ldots$$

$$y_{ret} := \ldots\ y_{after}\ \ldots$$

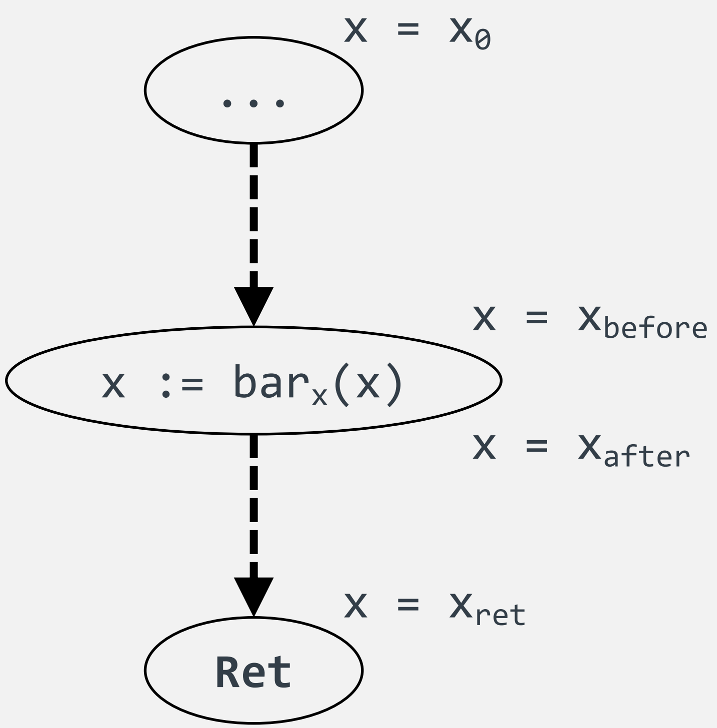
Goal

$$x_{ret} = y_{ret}$$



Verifying refinement: Handling function calls

foo$_x$
input: x
output: x



Assumptions

$x_{before} = y_{before} \longrightarrow x_{after} = y_{after}$

$x_0 = y_0$

$x_{before} := \ldots\ x_0\ \ldots$

$x_{ret} := \ldots\ x_{after}\ \ldots$

$y_{before} := \ldots\ y_0\ \ldots$

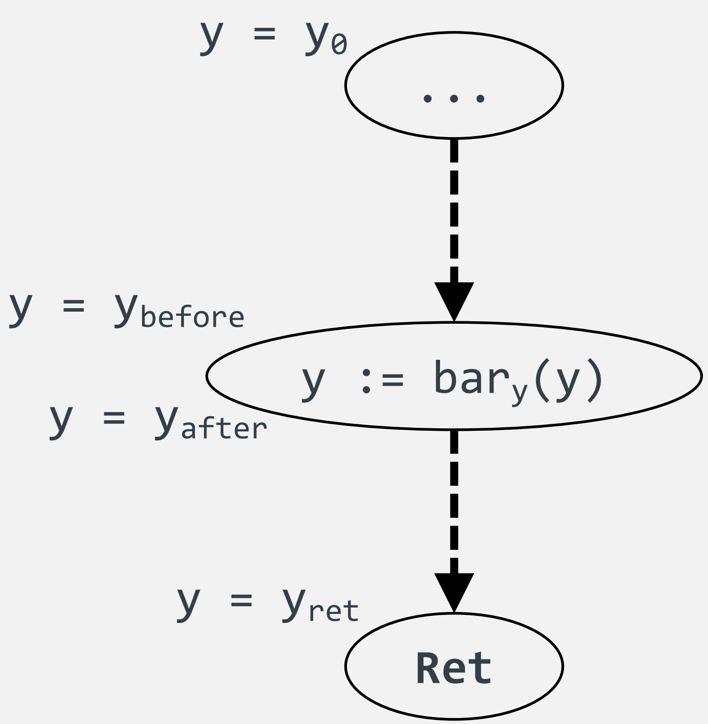
$y_{ret} := \ldots\ y_{after}\ \ldots$

$x_{ret} \neq y_{ret}$

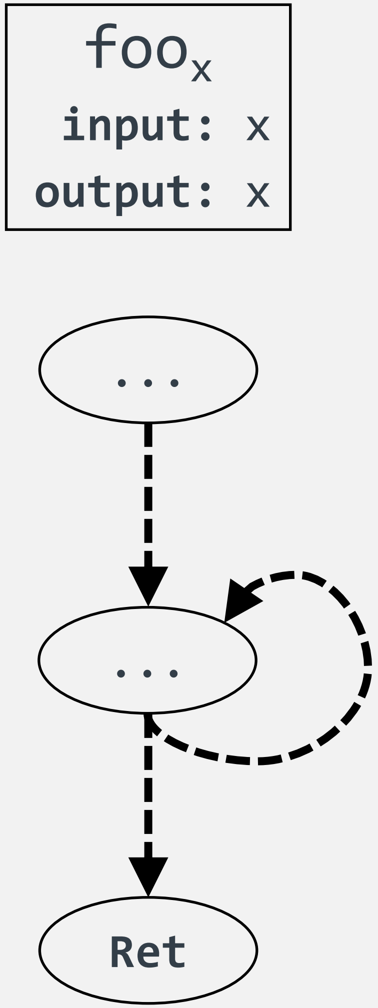
Query

Satisfiable?

foo$_y$
input: y
output: y



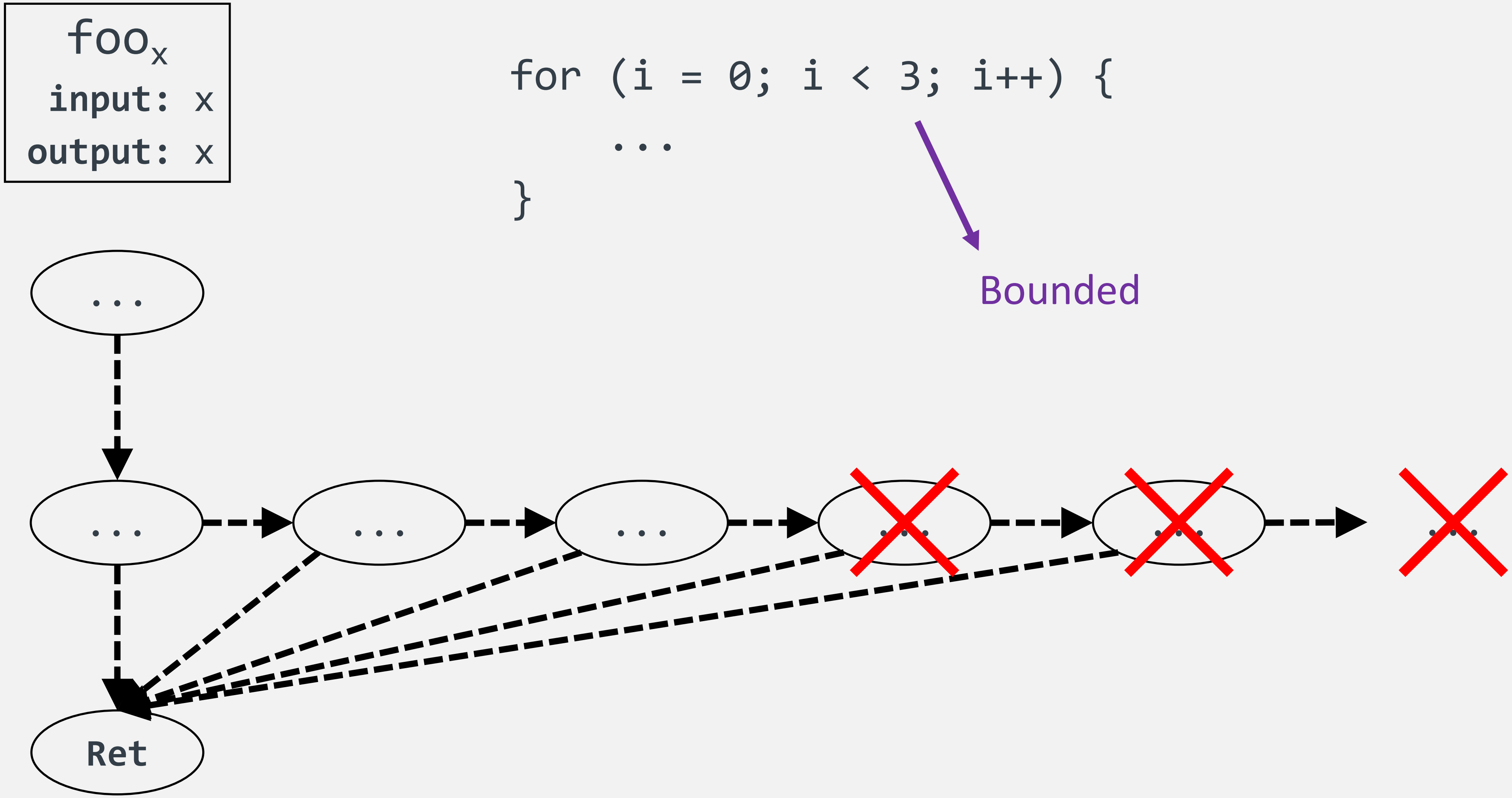
Verifying refinement: Handling loops




```
for (i = 0; i < 3; i++) {
    ...
}
```

Bounded

Verifying refinement: Handling loops



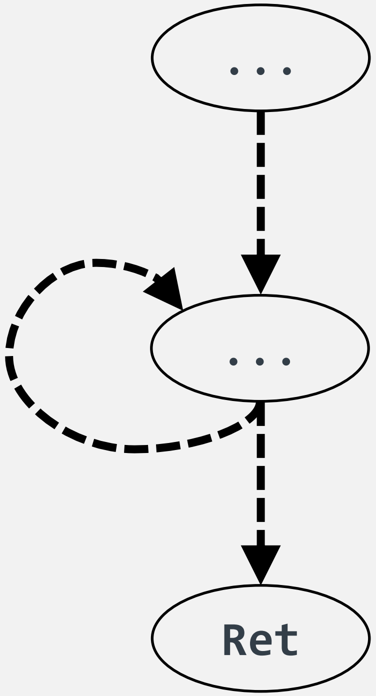

```
for (i = 0; i < 3; i++) {
    ...
}
```

Verifying refinement: Handling loops

foo_x

input: x

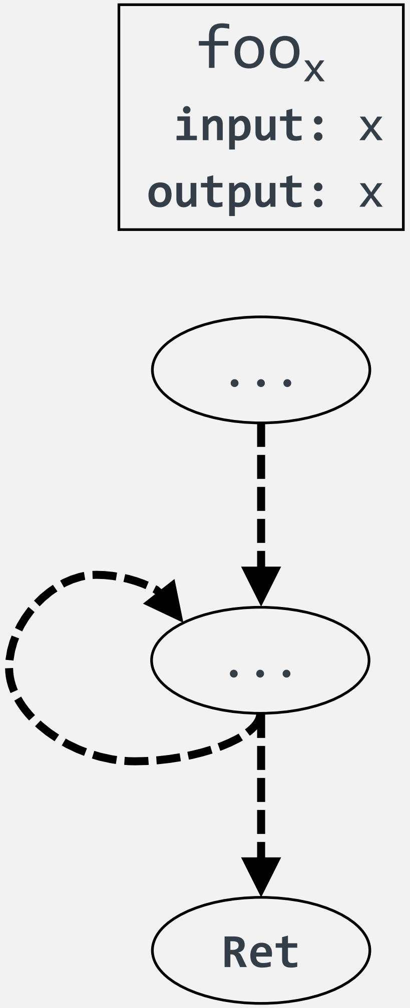
output: x




```
for (i = 0; i < 3; i++) {
    ...
}
```

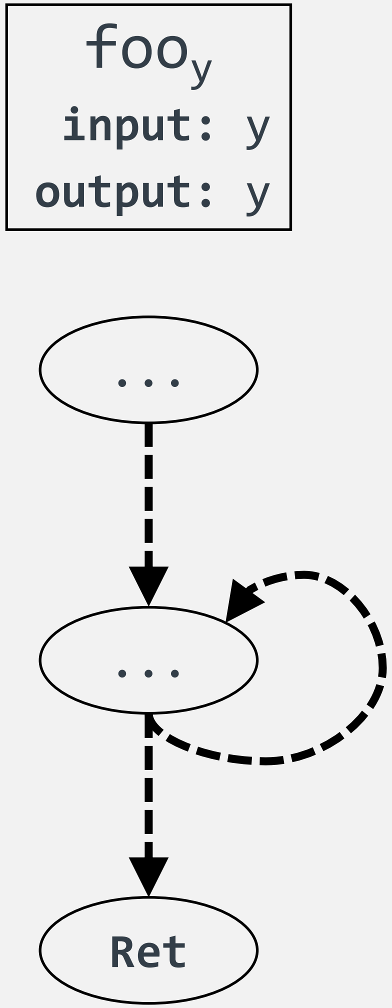
Bounded

Verifying refinement: Handling loops




```
for (i = 0; i < n; i++) {
    ...
}
```

Unbounded

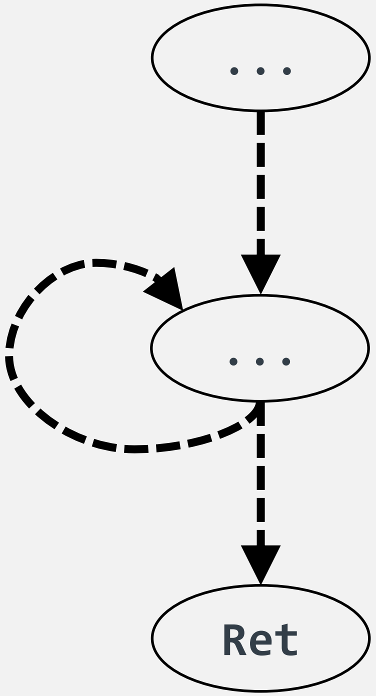


Loop relation!

- Visits to loop heads correspond
- Variable values at loop head visits are either:
  - Constant or a function of the visit number
  - Related to variables at the corresponding visit
  - Irrelevant

Verifying refinement: Handling loops

$\texttt{foo}_x$
input: x
output: x

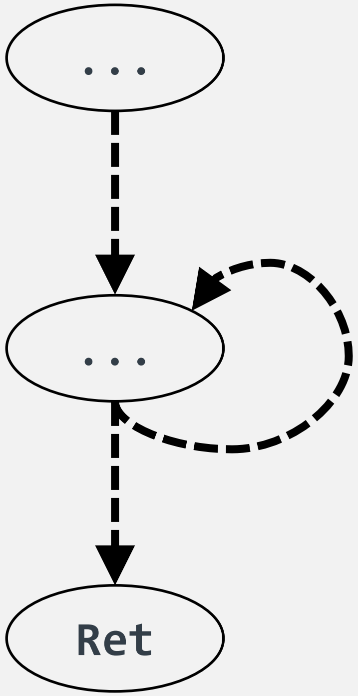


```
for (i = 0; i < n; i++) {
    ...
}
```

Unbounded

Loop relation!

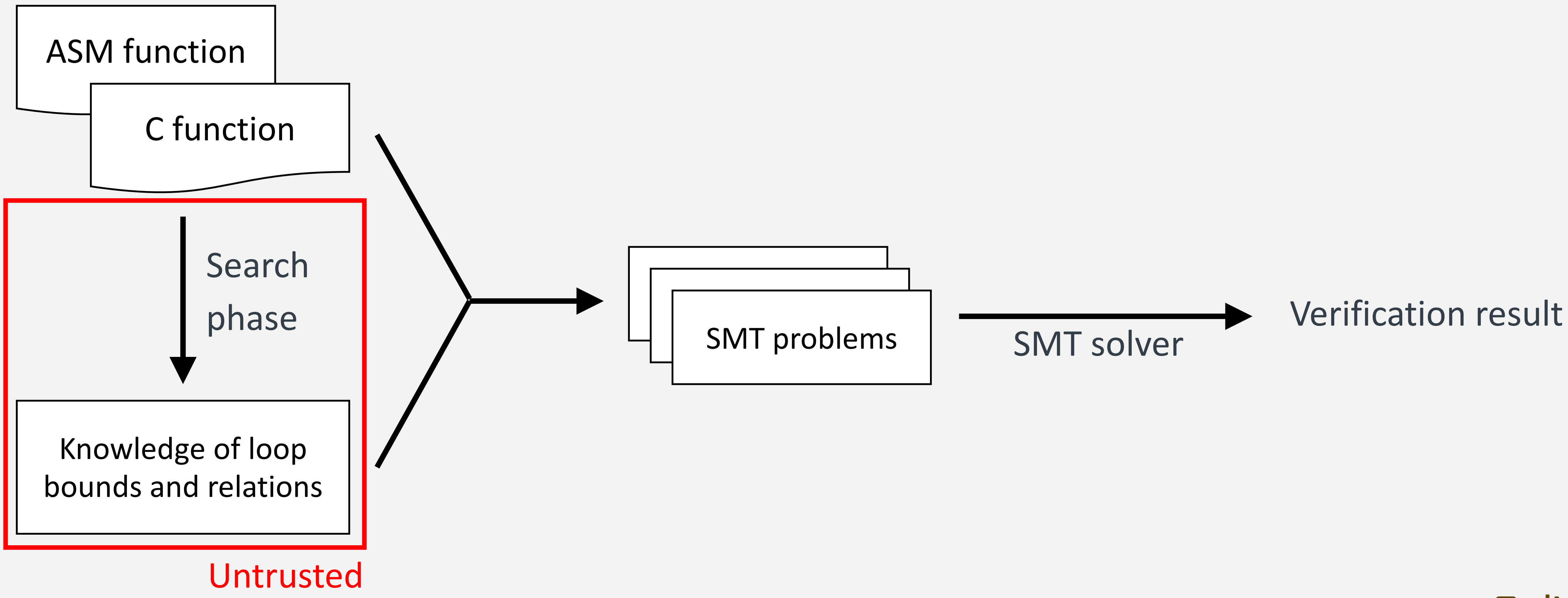
$\texttt{foo}_y$
input: y
output: y



Strategy:

1. Prove the loop relation using induction
2. Treat the loop as a black box

Verifying refinement: Search phase vs check phase



Colias Group

Implementation: graph-refine

Completed in 2013

Translation Validation for a Verified OS Kernel

Thomas Sewell
NICTA & UNSW, Sydney, Australia
thomas.sewell@nicta.com.au

Magnus Myreen
Cambridge University, UK
magnus.myreen@cl.cam.ac.uk

Gerwin Klein
NICTA & UNSW, Sydney, Australia
gerwin.klein@nicta.com.au

https://sel4.systems/Research/pdfs/translation-validation-verified-os-kernel.pdf

Thomas Sewell's 2017 PhD thesis

TRANSLATION VALIDATION FOR VERIFIED, EFFICIENT AND TIMELY OPERATING SYSTEMS

Thomas Sewell

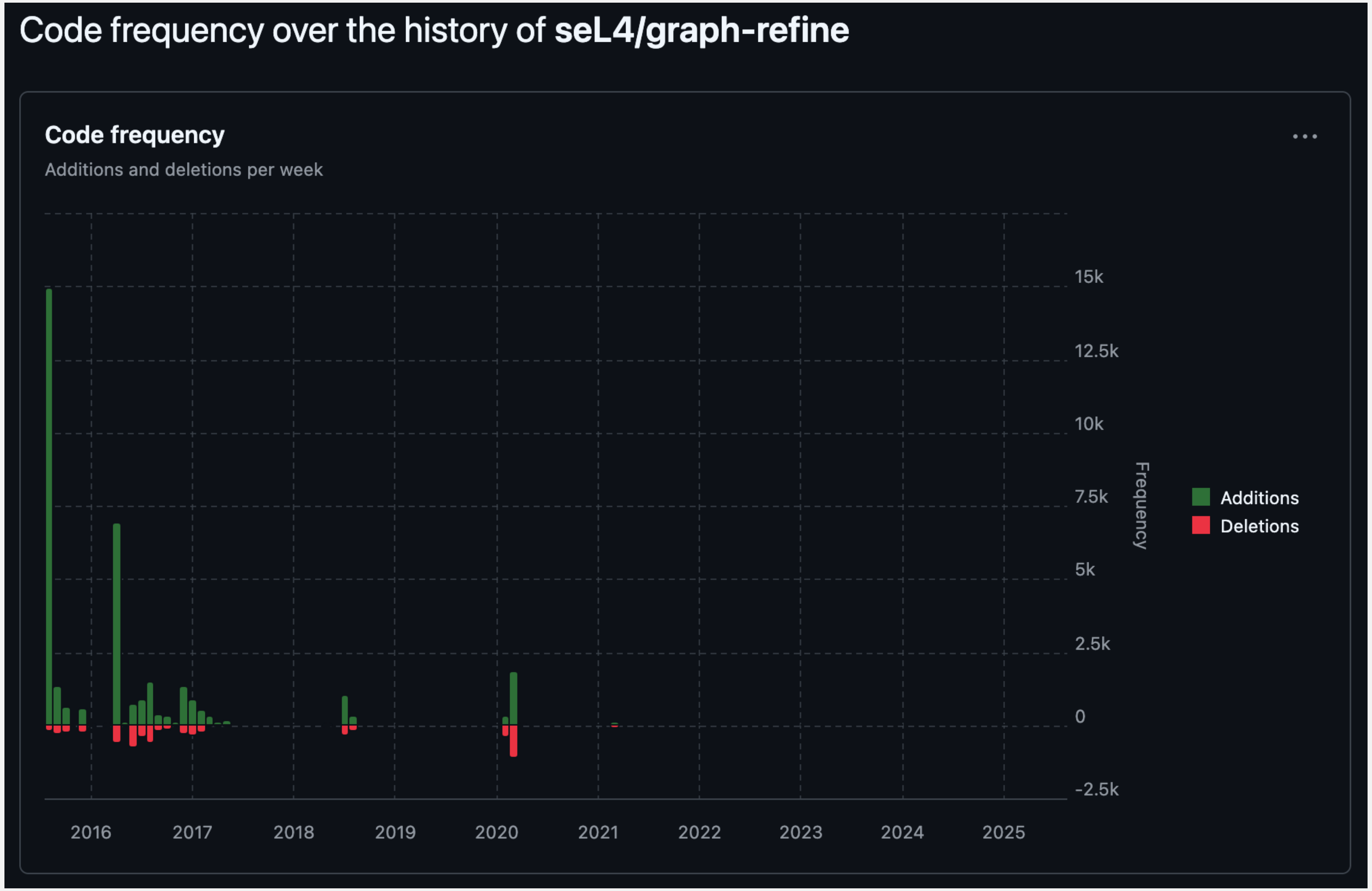
https://trustworthy.systems/publications/papers/Sewell%3Aphd.pdf

More recent work by Matt Brecknell, Yanyan Shen, and Zoltan Kocsis

Colias Group

Implementation: graph-refine

https://github.com/seL4/graph-refine



Implementation: graph-refine

Current state

- Account for changes to seL4
- Account for progress in GCC

Former glory

- Maintainability
- Scalability
- Auditability + testing
- AArch64
- -O2

Beyond

Colias Group

Implementation: New graph-refine

WIP: https://github.com/coliasgroup/seL4-binary-verification

New code design

Highlight: Cloud Haskell

Status: check phase complete, still working on search phase

Implementation: New graph-refine

Thanks to the seL4 Foundation for funding this work

Discussion

mailto:nick@nickspinale.com

Colias Group