NICTA

Ad hoc C: Reflections on Pragmatic Semantics

Michael Norrish
Software Systems Research Group

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Australian National University
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW GOVERNMENT Trade & Investment
ACT GOVERNMENT
State Government Victoria
THE UNIVERSITY OF MELBOURNE
THE UNIVERSITY OF SYDNEY
Queensland Government
Griffith UNIVERSITY
QUT Queensland University of Technology
The University Of Queensland AUSTRALIA

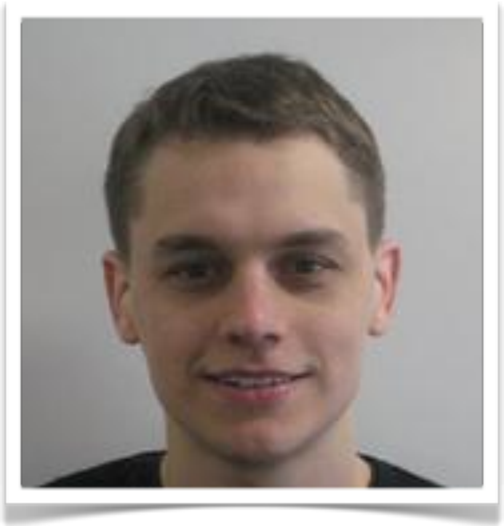
Three seL4 Stories

Basic C Technology (the "C parser")

- worryingly *ad hoc*

Binary Validation (Sewell & Myreen)

- *ad hoc,* but **dispels** worry
- PLDI 2013



Automatic Abstraction (Greenaway)

- *ad hoc* complexity/effort removal
- ITP 2012

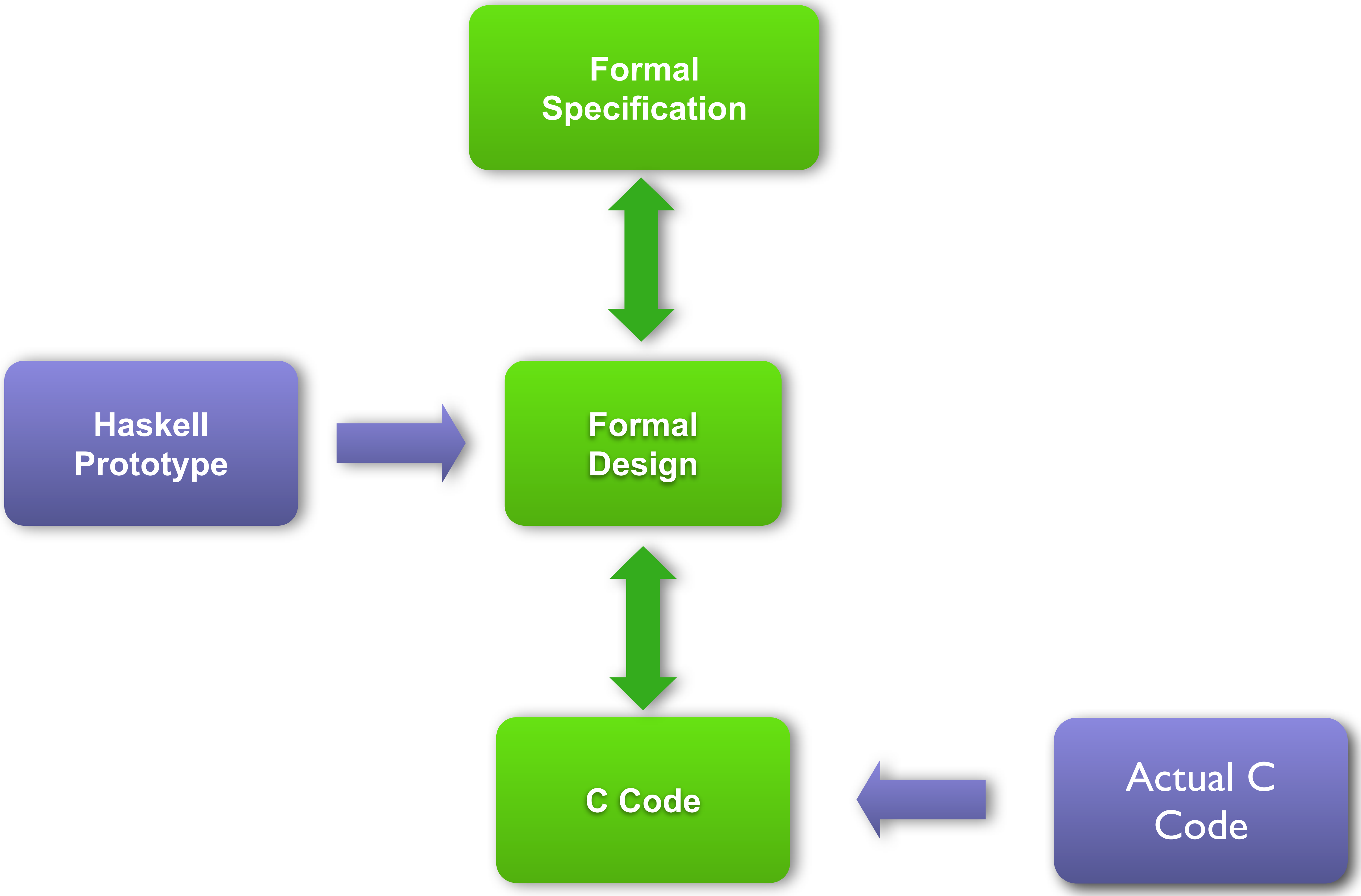
seL4, *c*2009

Fully featured operating system kernel

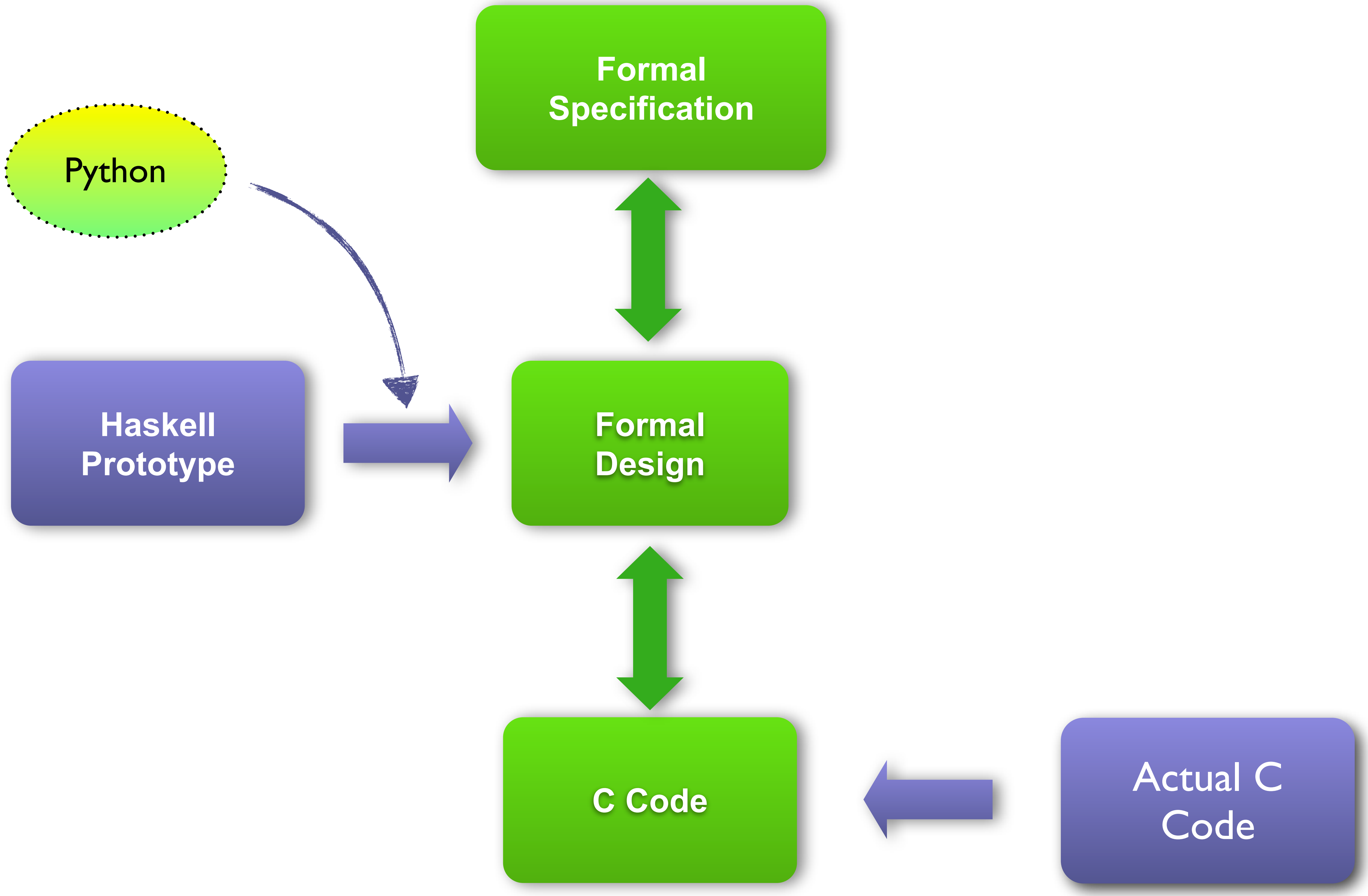
9000 lines of source code

Full functional correctness

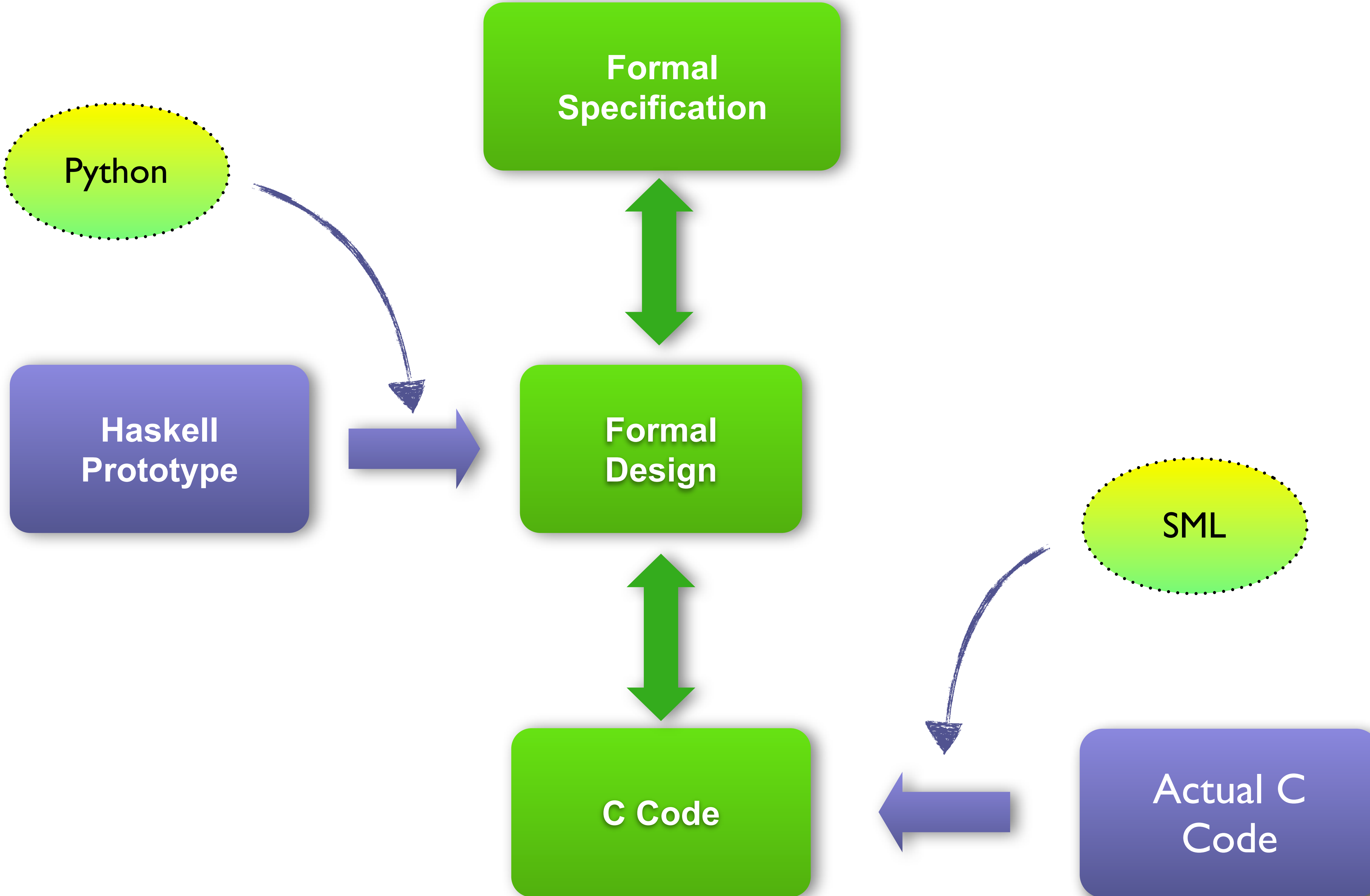
25 person-years of effort



Formal Specification
Haskell Prototype
Formal Design
C Code
Actual C Code



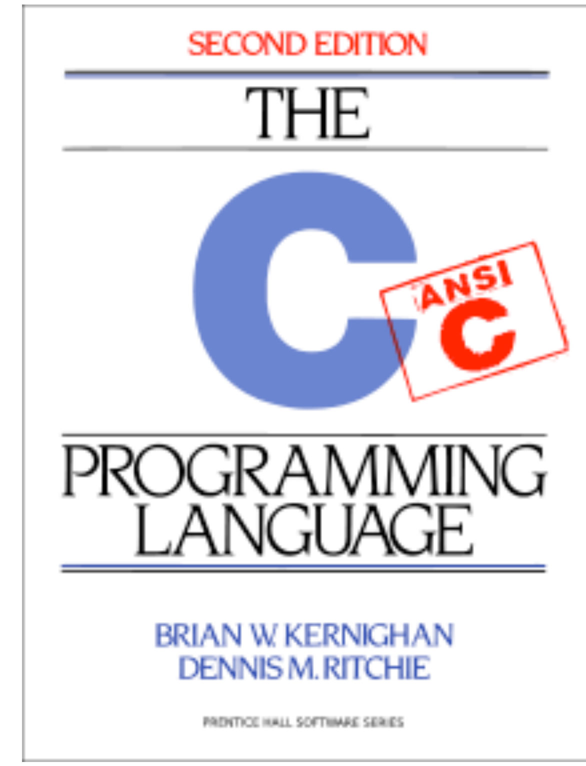
Python
Formal Specification
Haskell Prototype
Formal Design
C Code
Actual C Code


Formal Specification
Python
Haskell Prototype
Formal Design
SML
C Code
Actual C Code

C

Making C a high integrity language?

- some industrial experience, *e.g.*, MISRA C
- Better, CompCert is a **verified compiler**!
- but we can't use it



C in L4.verified

- for the OS hackers: pointer arithmetic, casts, interpreting memory as untyped bytes
- for the verifiers: restrictions on side effects
- tool-chain: very dependent on `gcc -O2` and linker

C subset

Everything from C standard

- **including**:
  - pointers, casts, pointer arithmetic
  - data types
  - structs, padding
  - pointers into structs
  - precise finite integer arithmetic

- **minus:**
  - goto, switch fall-through
  - reference to local variable
  - side-effects in expressions
  - function pointers (restricted)
  - unions

- **plus** compiler assumptions on:
  - data layout, encoding, endianess

```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
    case (word_t)SchedulerAction_ResumeCurrentThread:
        break;

    case (word_t)SchedulerAction_ChooseNewThread:
        chooseThread();
        ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
...
        thread; thread = next) {

        if(!isRunnable(thread)) {
            next = thread->tcbSchedNext;
            tcbSchedDequeue(thread);
        }
        else {
            switchToThread(thread);
            return;
        }
      }
    }

    switchToIdleThread();
}
```

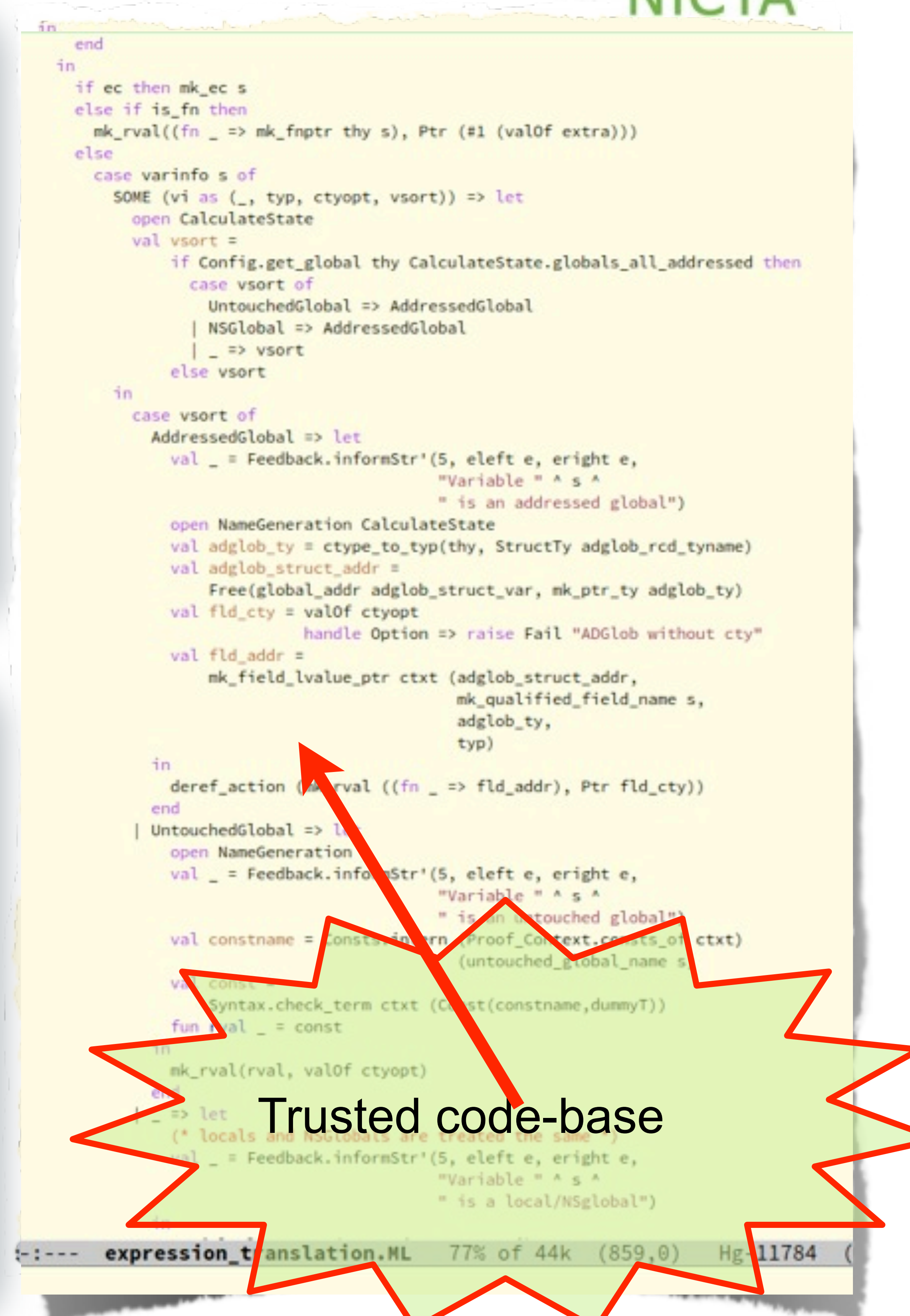
C, as defined by our tools

11000 lines of SML

- (+19kloc in parser/lexer code)
- turns C into Isabelle/HOL definitions
- Verifier uses Isabelle to prove refinement theorem

Errors

- In translator: unexpected goals
  - (Verifier says “This isn’t what I think the C means”)
- In the code: unsolvable goals



Other Bits of C Technology

SIMPL's Verification Condition Generator

- general tool that often "just works"

seL4 Refinement Tactic

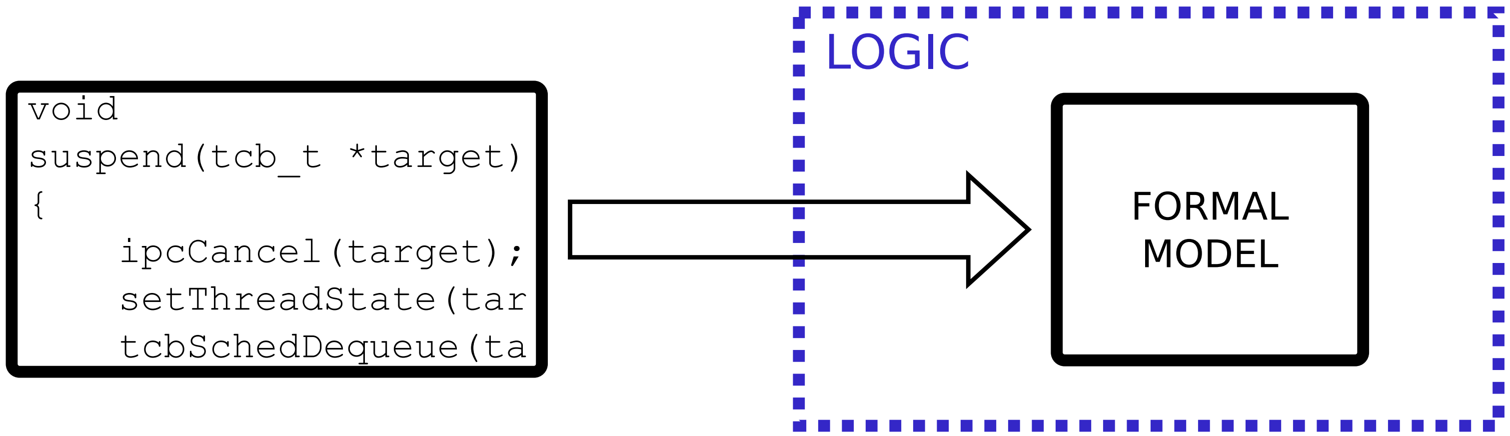
- used to verify link between C code to and (Haskell-derived) design-level spec.

Union preprocessing

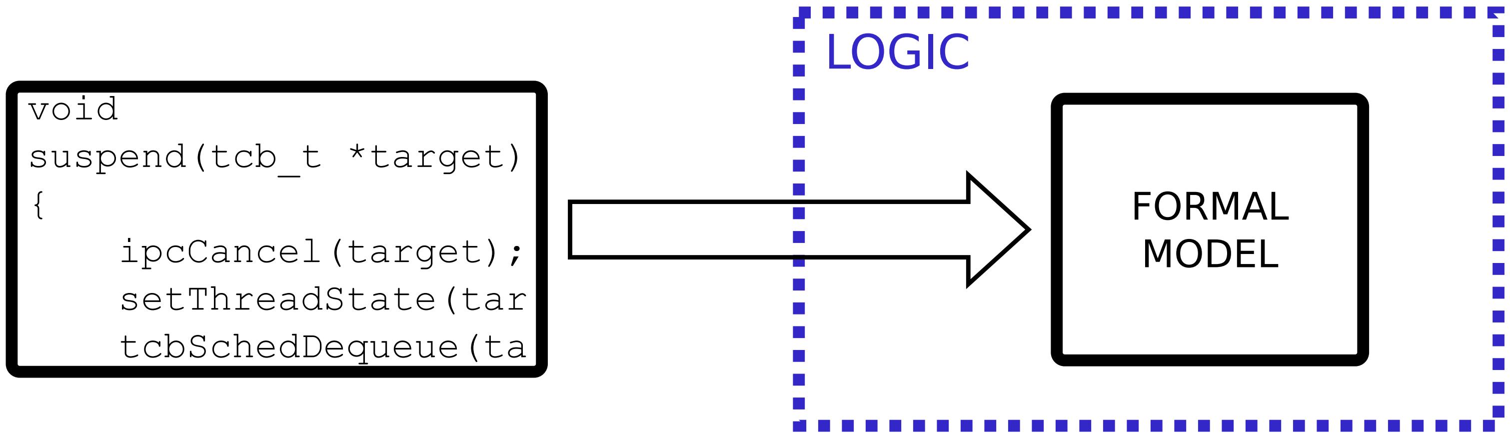
- seL4 hackers didn't trust/like gcc's implementation of unions
- custom tool translates out all unions into structs and casts

[TPHOLs 2009]

Binary Validation: the Problem



Binary Validation: the Problem

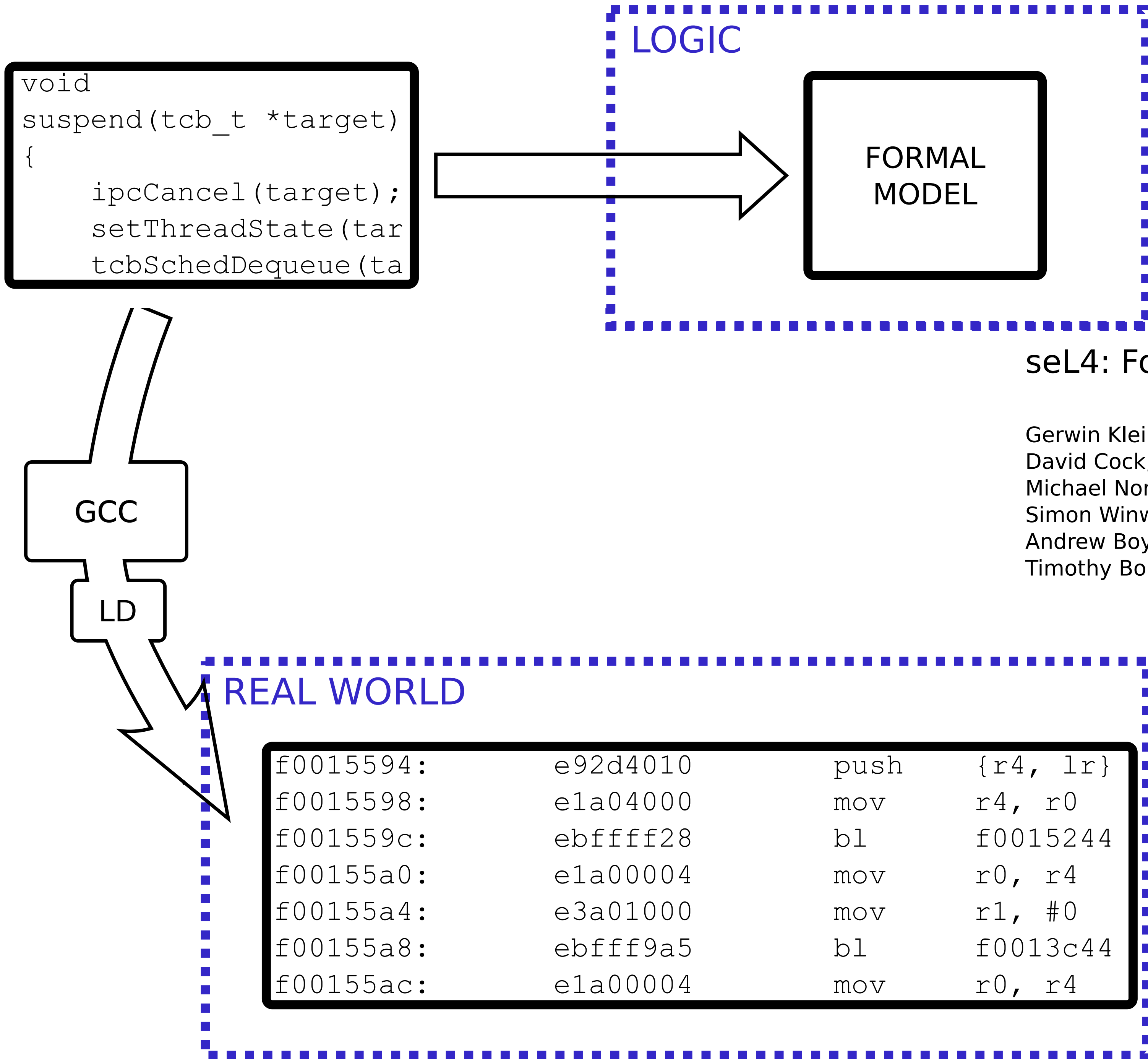


NICTA

seL4: Formal Verification and All That

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick,
David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt,
Michael Norrish, Rafal Kolanski, Thomas Sewell, Harvey Tuch,
Simon Winwood, Peter Gammie, Toby Murray, David Greenaway,
Andrew Boyton, Daniel Matichuk, Matthew Brassil,
Timothy Bourke, Sean Seefried, Corey Lewis and Xin Gao.

Binary Validation: the Problem



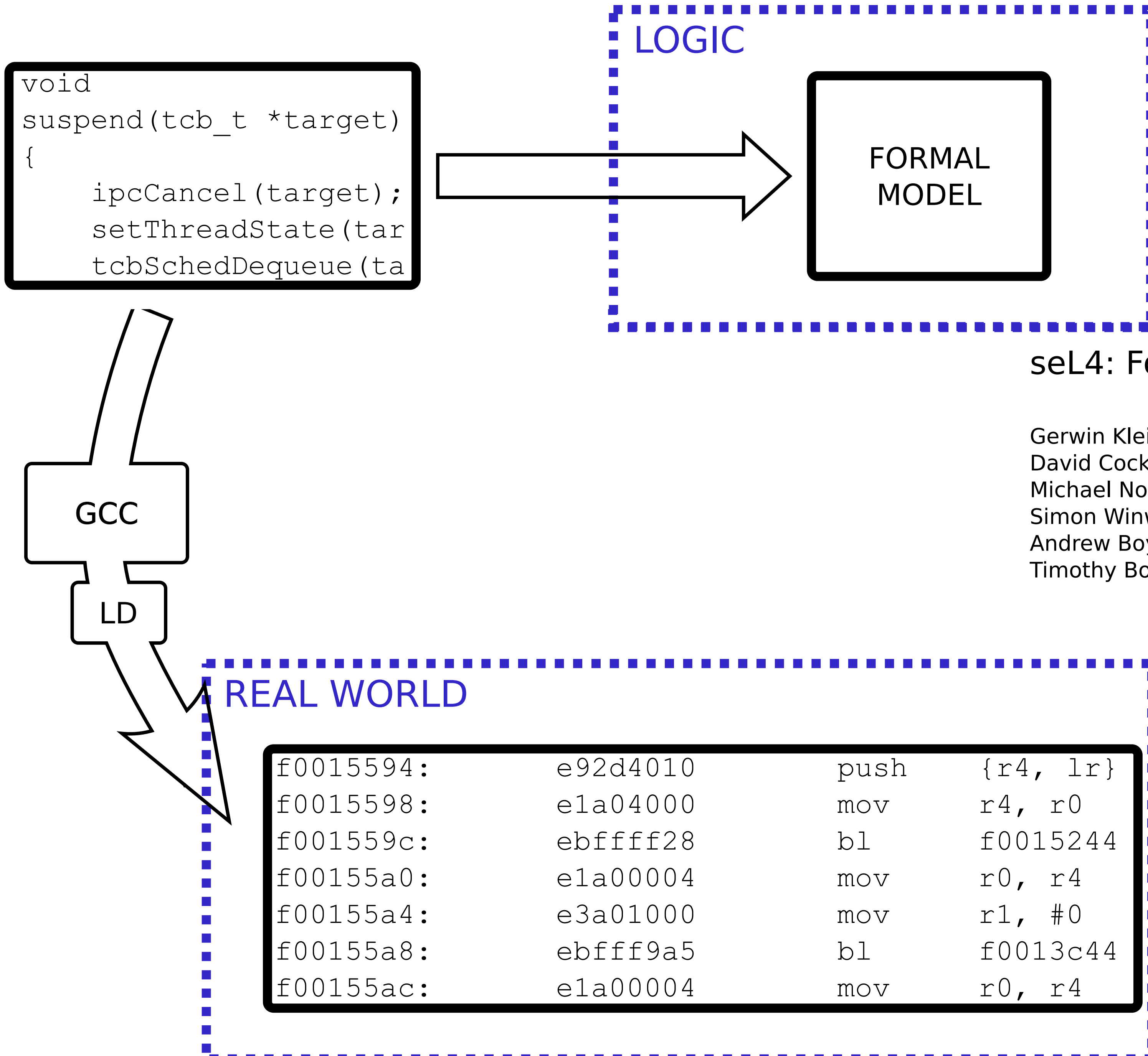
NICTA

seL4: Formal Verification and All That

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Michael Norrish, Rafal Kolanski, Thomas Sewell, Harvey Tuch, Simon Winwood, Peter Gammie, Toby Murray, David Greenaway, Andrew Boyton, Daniel Matichuk, Matthew Brassil, Timothy Bourke, Sean Seefried, Corey Lewis and Xin Gao.

Binary Validation: the Problem

NICTA



NICTA

seL4: Formal Verification and All That

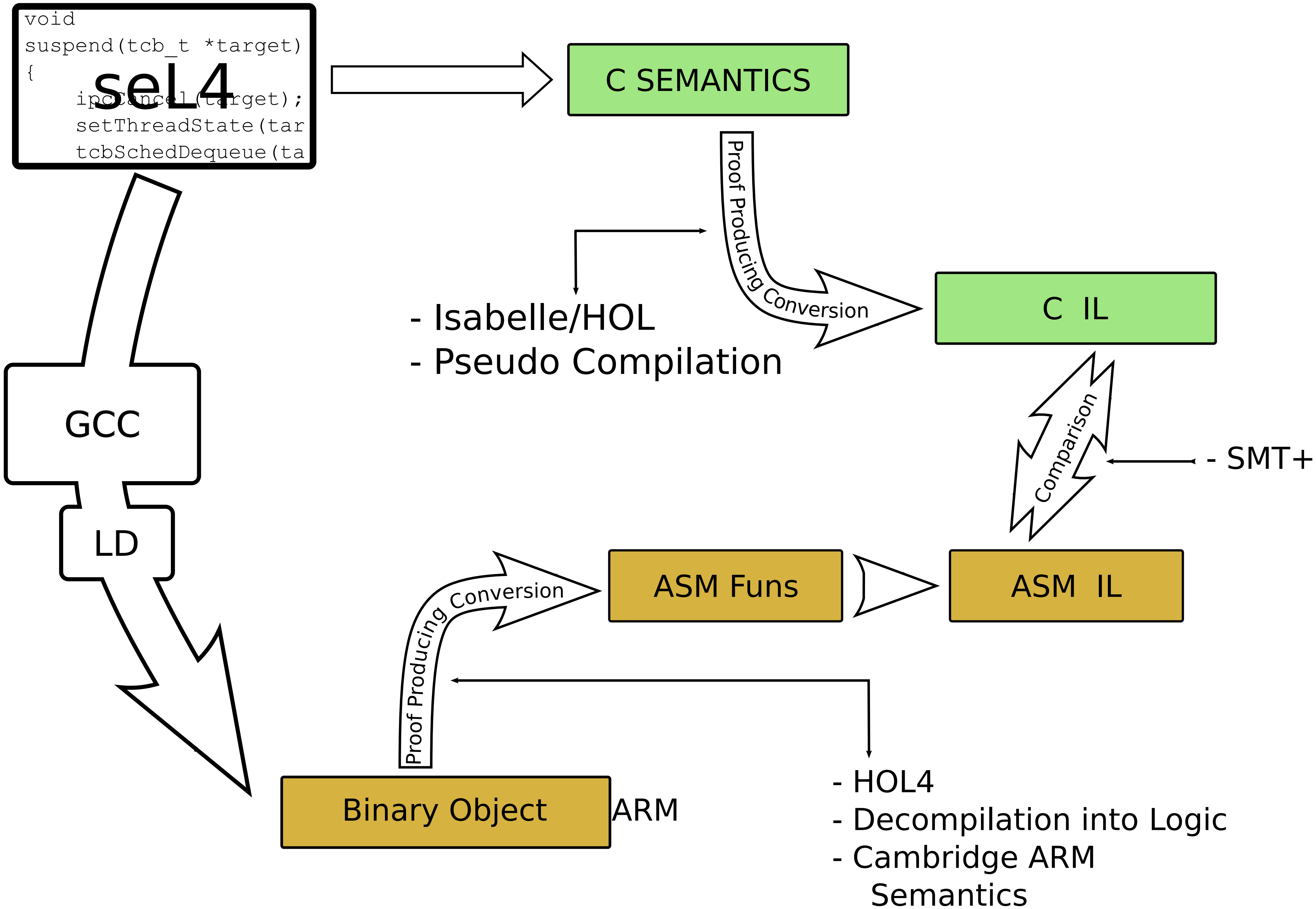
Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Michael Norrish, Rafal Kolanski, Thomas Sewell, Harvey Tuch, Simon Winwood, Peter Gammie, Toby Murray, David Greenaway, Andrew Boyton, Daniel Matichuk, Matthew Brassil, Timothy Bourke, Sean Seefried, Corey Lewis and Xin Gao.

UNIVERSITY OF CAMBRIDGE

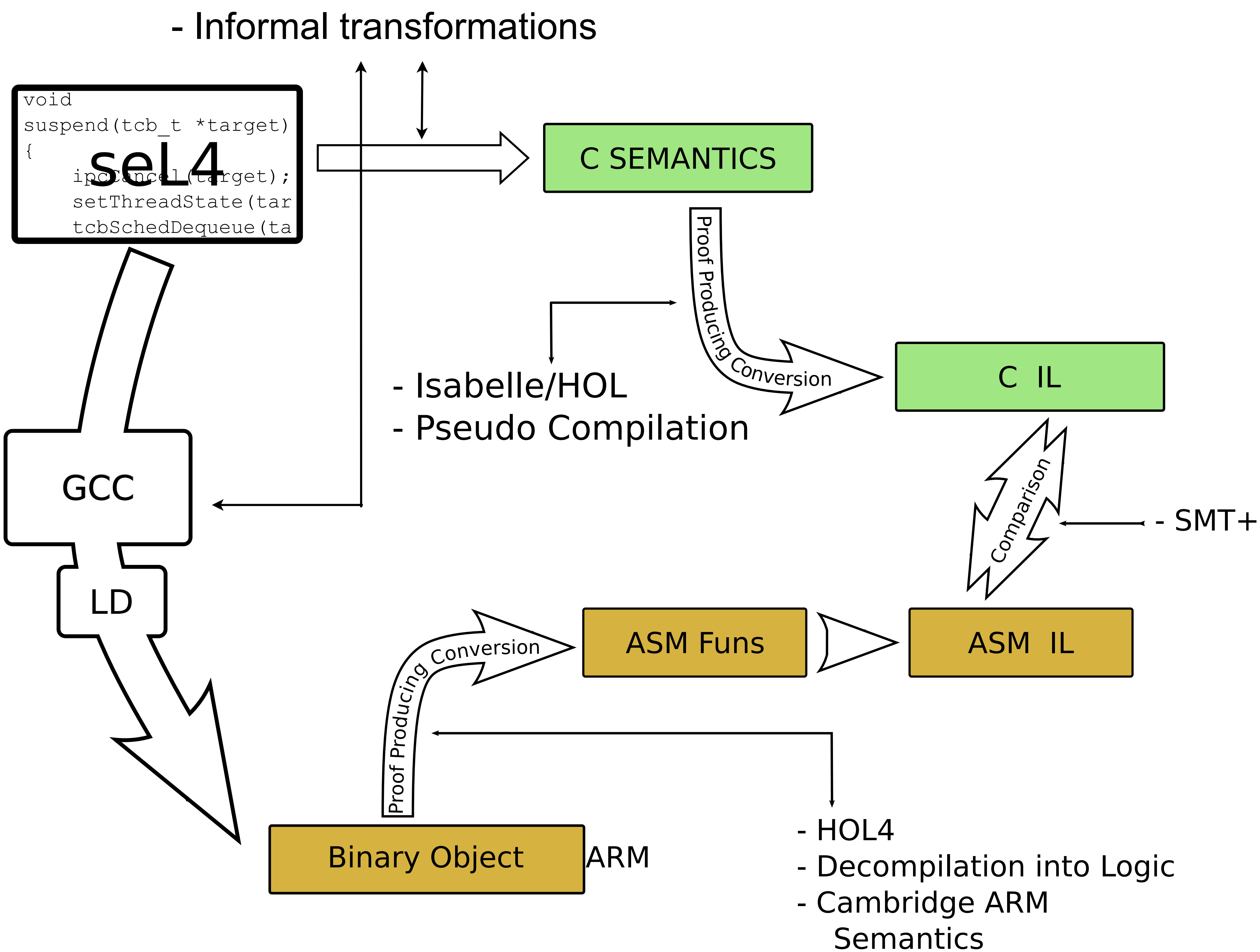
ARM ISA and Decompiler

Anthony Fox and Magnus Myreen

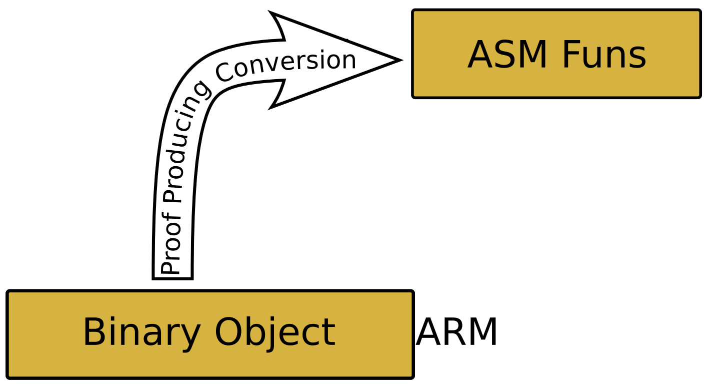
Binary Validation: Solution in a Graph



Binary Validation: Solution in a Graph



Machine Code Decompilation




```
uint avg (uint i, uint j) {
  return (i + j) / 2;
}
```

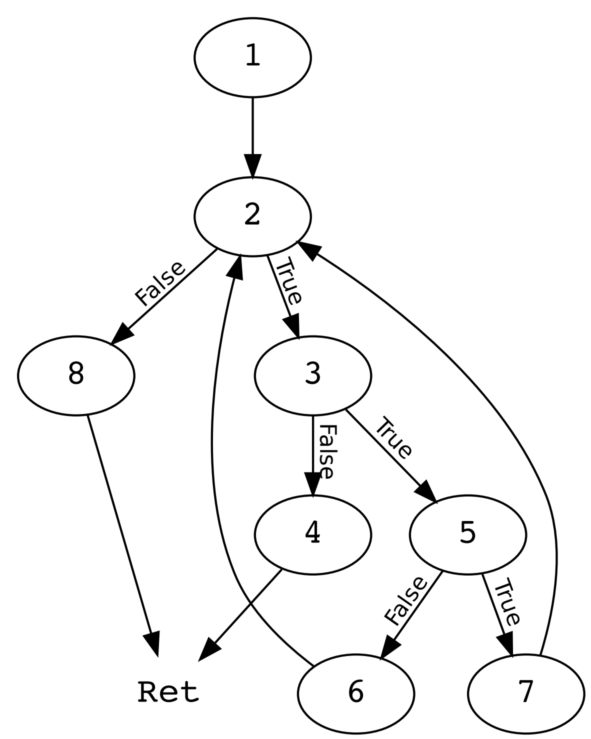
```
<avg>:
avg+0    e0810000    add r0, r1, r0    // add r1 to r0
avg+4    e1a000a0    lsr r0, r0, #1    // shift r0 right
avg+8    e12fff1e    bx  lr            // return
```

$$
\begin{aligned}
\text{avg}\,(r_0, r_1) \quad = \quad & \text{let } r_0 = r_1 + r_0 \text{ in} \\
& \text{let } r_0 = r_0 >>> 1 \text{ in} \\
& \quad r_0
\end{aligned}
$$

Control-Flow Graphs

Machine code and C code eventually turn into annotated control flow graphs

```
struct node *
find (struct tree *t, int k) {
  struct node *p = t->trunk;
  while (p) {
    if (p->key == k)
      return p;
    else if (p->key < k)
      p = p->right;
    else
      p = p->left;
  }
  return NULL;
}
```




```
1: p := Mem[t + 4];

2: p == 0 ?

8: ret := 0
3: Mem[p] == k ?

4: ret := p;
5: Mem[p] < k ?

6: p := Mem[p + 4];
7: p := Mem[p + 8];
```

Critically, can consider one function at a time when verifying (modulo inlining).

Comparing Graphs: Challenges

NICTA

inlining

- if gcc inlines a function, the “C graph” has to do so too

loops

- must prove that loop points are reached the same number of times
- must also cope with complete or partial unrollings

treatment of the stack

- machine code accesses to memory that are loads/stores of spilled parameters need to be recognised as such

Loops, Loops, Loops, Loops, Loops, ...

Can sometimes infer that a loop should only ever execute a fixed *n* times.

- *e.g.*, `for(i=0; i < 10; i++) { .... }`
- SMT solver can confirm such an inference ...
  - formula is finite, of size *O(n)*
- *...* and establish appropriate post-condition(s)

Alternatively, use *k*-induction to prove that every visit to a loop point in the binary is matched by one in the C.

- invariant relating C and binary variables is also preserved
- offsets may be required to handle partial unrollings

Graph Equivalence Technology

Problem domain is good fit for QF_ABV SMT category

- *i.e.*, arrays + boolean vectors

Used both Z3 and Sonolar

- Sonolar performs better on arrays (used to represent memory)
- Z3 can restart (retracting facts), making efficiency performance better

seL4, *c*2009

Fully featured operating system kernel

9000 lines of source code

Full functional correctness

25 person-years of effort

seL4, *c*2009

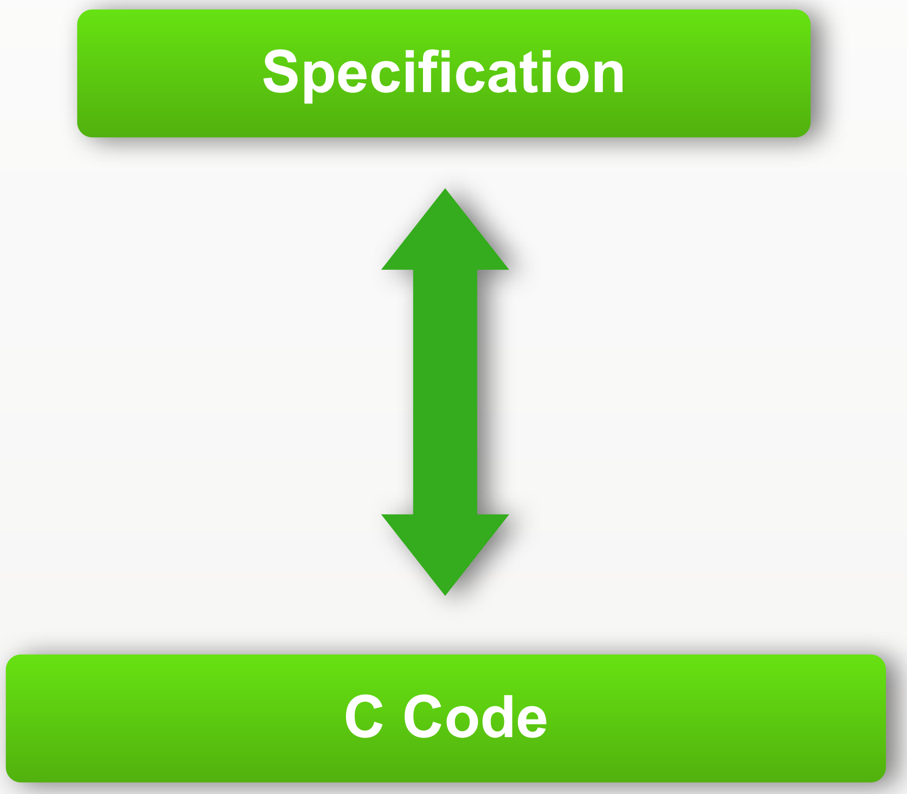
Fully featured operating system kernel

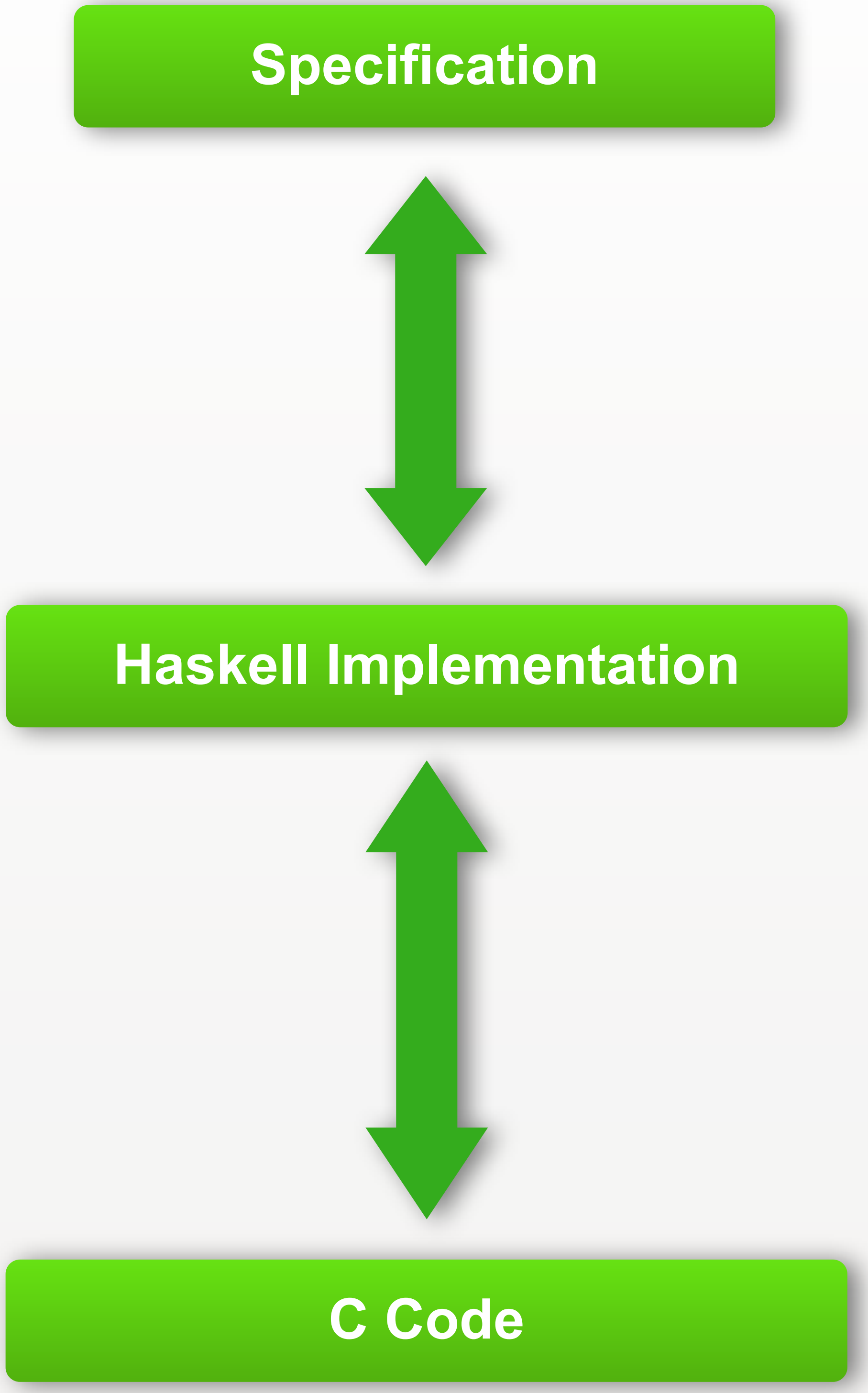
9000 lines of source code

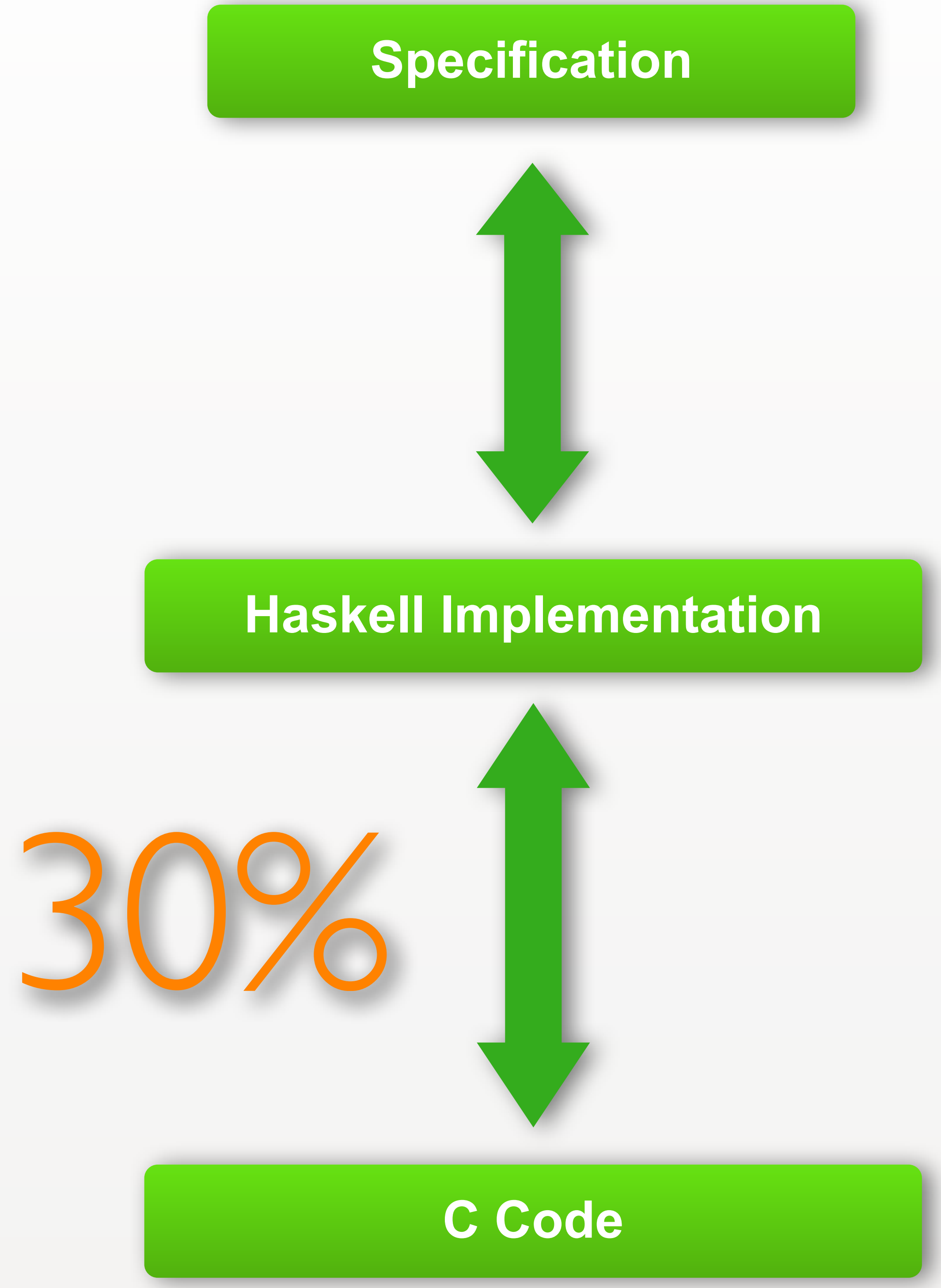
Full functional correctness

25 person-years of effort

This is a disturbingly large number; where can we improve productivity?


Specification
C Code


Specification
Haskell Implementation
C Code


Specification
Haskell Implementation
30%
C Code

Some background to this mess

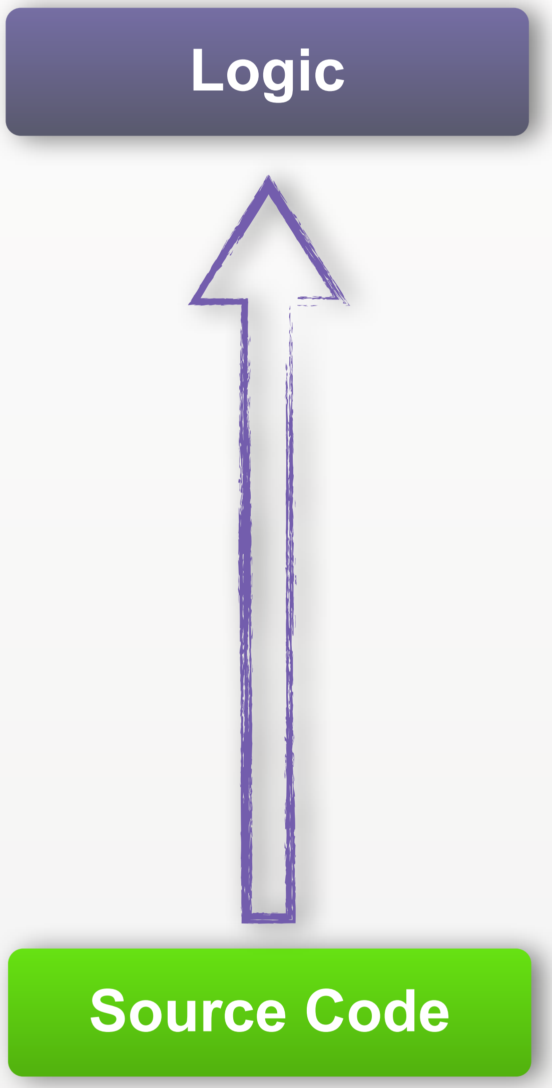
```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

Some background to this mess

Source Code

```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

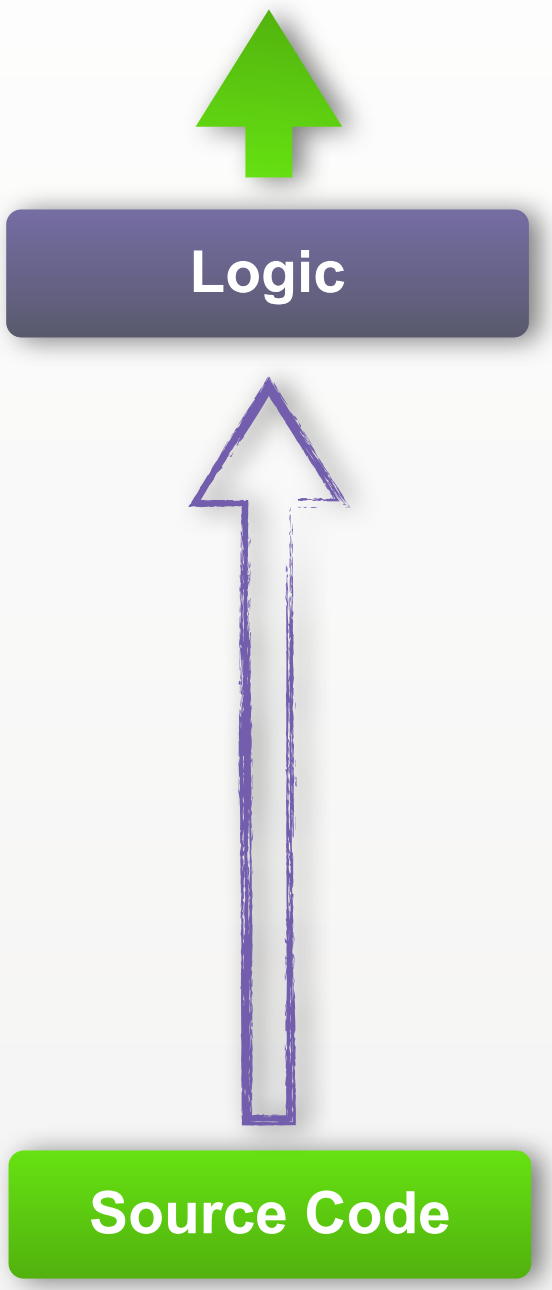
Some background to this mess




```
max a b ≡
  if a > b then b else a
```

```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

Some background to this mess




```
max a b ≡
  if a > b then b else a
```

```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

Some background to this mess

```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

```
max ≡
  TRY
    IF ⦃ ´a <s ´b ⦄ THEN
      ´ret_int :== ´b;;
      ´exn_var :== Return;;
      THROW
    ELSE
      SKIP
    FI;;
    ´ret_int :== ´a;;
    ´exn_var :== Return;
    THROW;
    GUARD DontReach {}
      SKIP;;
  CATCH
    SKIP
  END
```

Some background to this mess

NICTA

```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

```
max ≡
  TRY
    IF ⦃ ´a <s ´b ⦄ THEN
      ´ret_int :== ´b;;
      ´exn_var :== Return;;
      THROW
    ELSE
      SKIP
    FI;;
    ´ret_int :== ´a;;
    ´exn_var :== Return;
    THROW;
    GUARD DontReach {}
      SKIP;;
  CATCH
    SKIP
  END
```

Some background to this mess

```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

```
max ≡
  TRY
    IF ⦃ ´a <s ´b ⦄ THEN
      ´ret_int :== ´b;;
      ´exn_var :== Return;;
      THROW
    ELSE
      SKIP
    FI;;
    ´ret_int :== ´a;;
    ´exn_var :== Return;
    THROW;
    GUARD DontReach {}
      SKIP;;
  CATCH
    SKIP
  END
```

Some background to this mess

```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

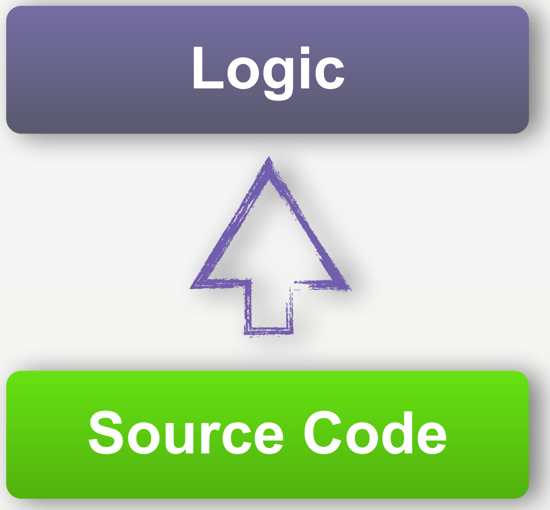
```
max ≡
  TRY
    IF ⦃ ´a <s ´b ⦄ THEN
      ´ret_int :== ´b;;
      ´exn_var :== Return;;
      THROW
    ELSE
      SKIP
    FI;;
    ´ret_int :== ´a;;
    ´exn_var :== Return;
    THROW;
    GUARD DontReach {}
      SKIP;;
  CATCH
    SKIP
  END
```

- Uninitialised variables
- Undefined behaviour
- Pointer arithmetic
- Type casting

Our approach

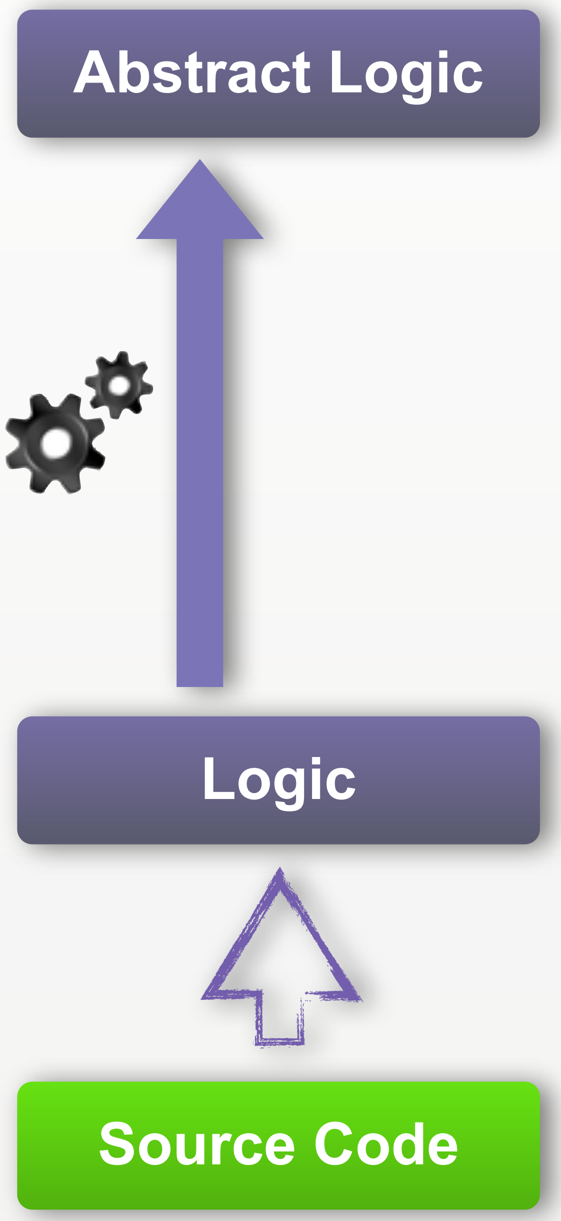
Source Code

Our approach



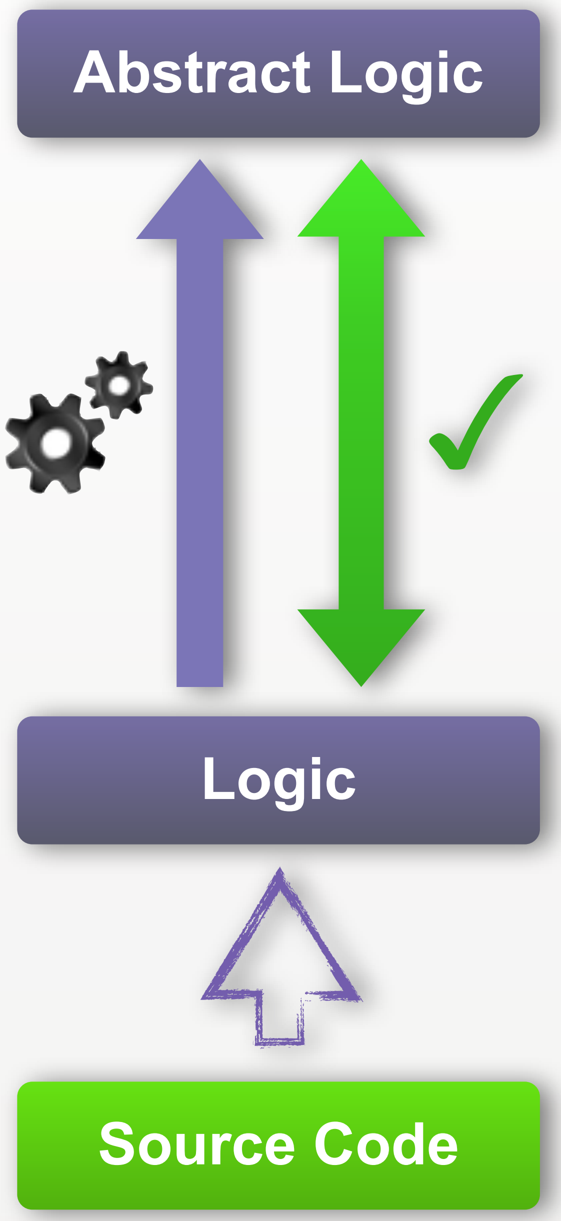
- Start with a conservative logical representation

Our approach



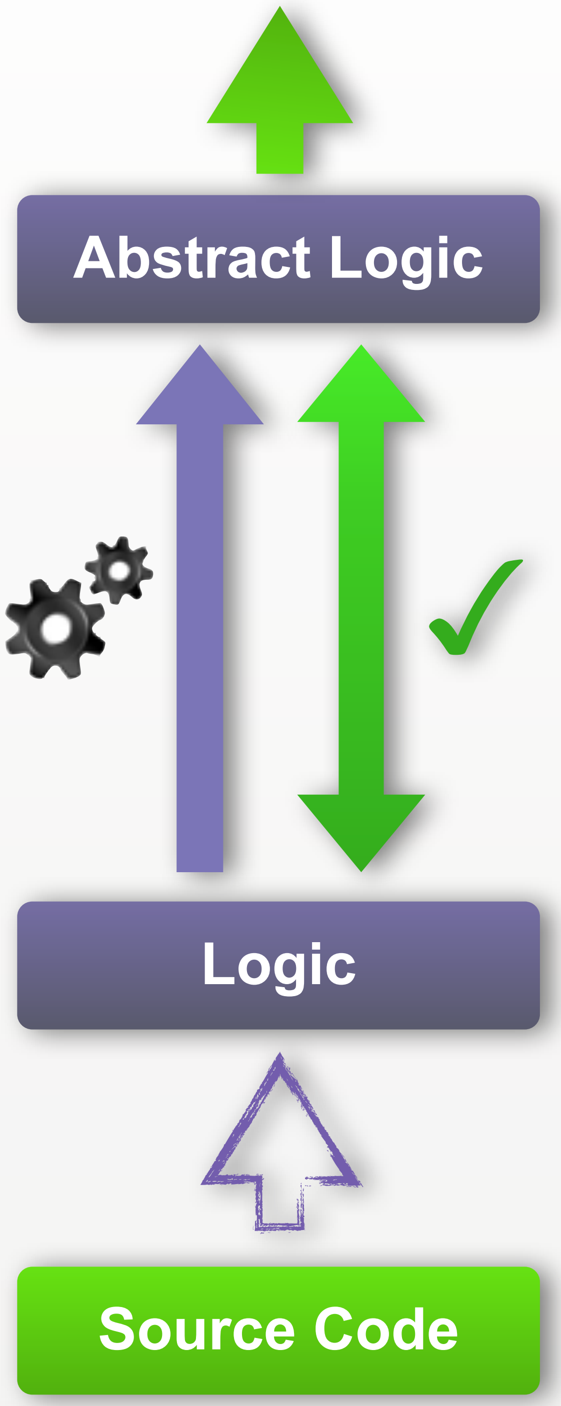
- Start with a conservative logical representation
- Automatically abstract low-level spec into higher-level spec
  - Goal: Suitable for human consumption

Our approach



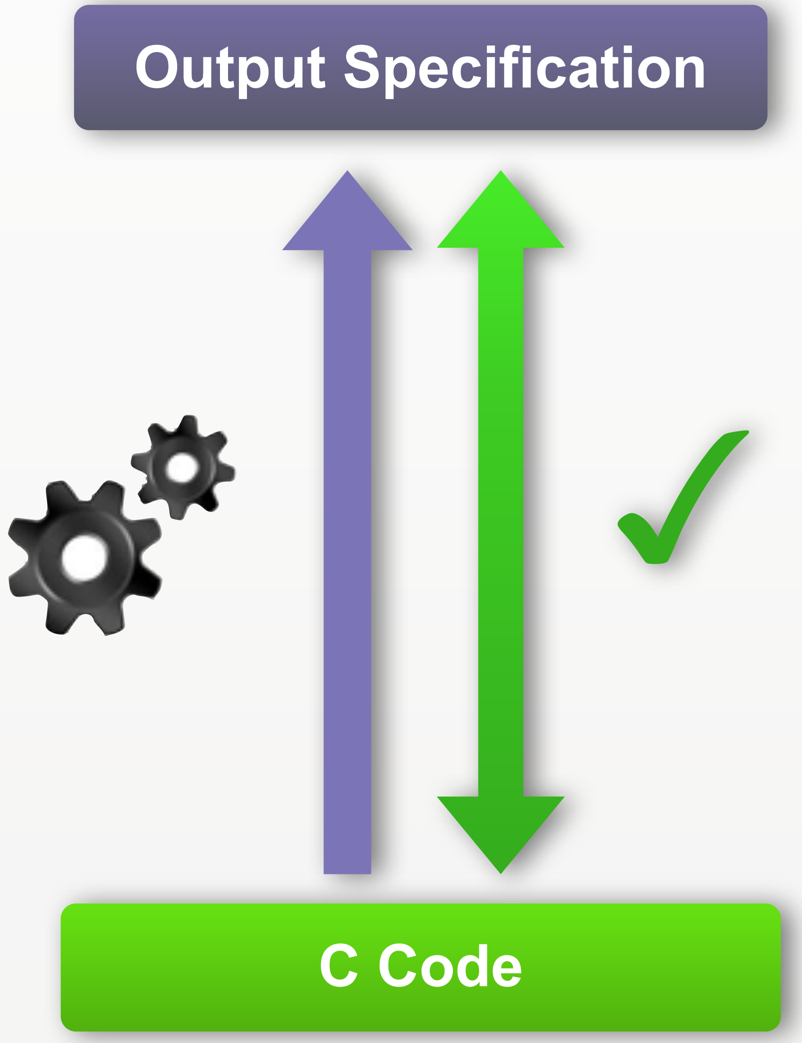
- Start with a conservative logical representation
- Automatically abstract low-level spec into higher-level spec
  - Goal: Suitable for human consumption
- Automatically generate a *refinement proof* showing the original spec is a refinement of the generated spec

Our approach

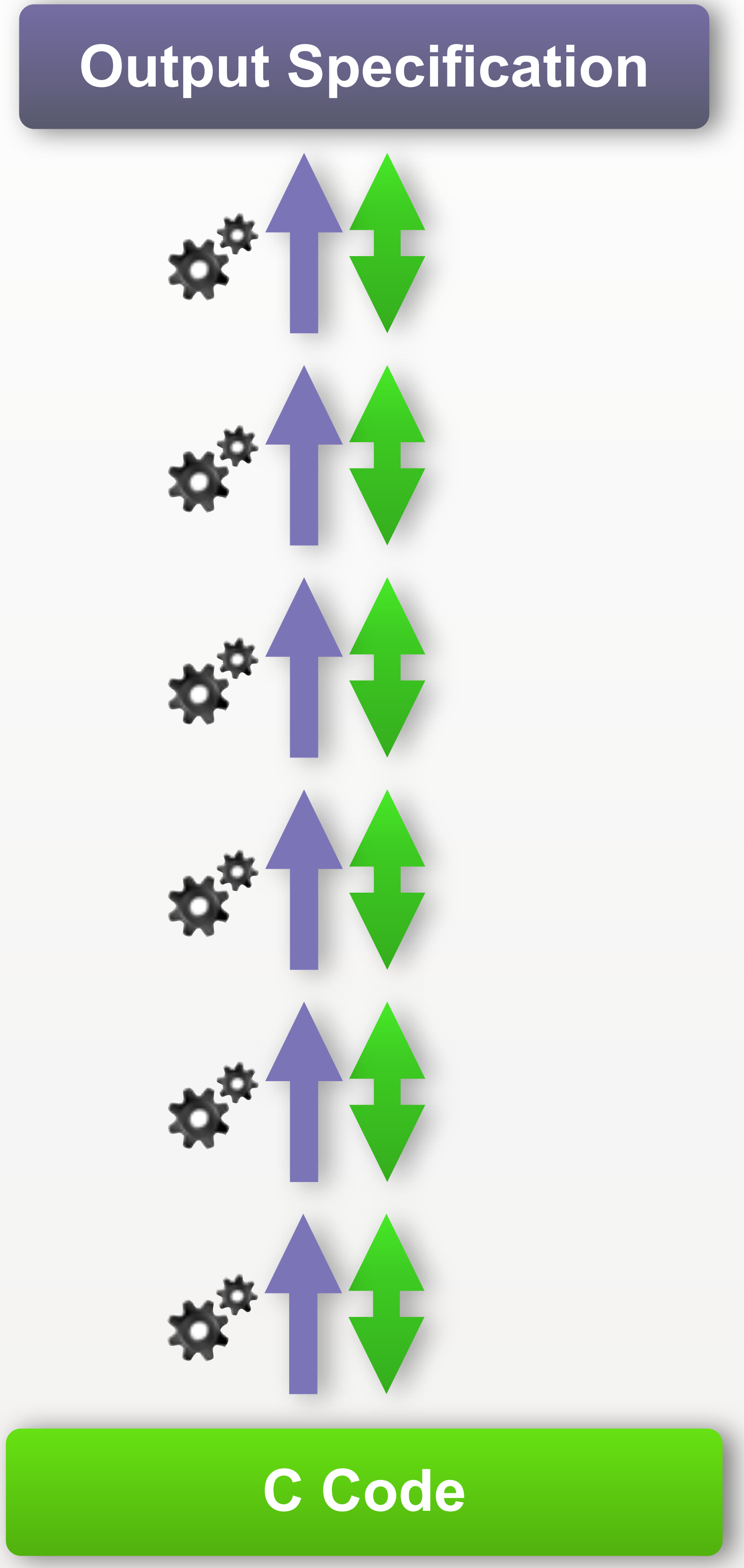


- Start with a conservative logical representation
- Automatically abstract low-level spec into higher-level spec
  - Goal: Suitable for human consumption
- Automatically generate a *refinement proof* showing the original spec is a refinement of the generated spec

Our approach



Our approach



Our approach

⑥ Type Strengthening

⑤ Flow-Sensitive Optimisations

④ Local Variable Extraction

③ Exception Elimination

② Peephole Optimisations

① Shallow Embedding

```
max ≡
  TRY
    IF {| ´a <s ´b |} THEN
      ´ret_int :== ´b;;
      ´exn_var :== Return;;
      THROW
    ELSE
      SKIP
    FI;;
    ´ret_int :== ´a;;
    ´exn_var :== Return;
    THROW;
    GUARD DontReach {}
      SKIP;;
  CATCH
    SKIP
  END
```

```
max a b ≡
  if (a <s b) then b else a
```

⑥ Type Strengthening

```
max a b ≡
  condition (a <s b)
    return b
    return a
```

```c
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

```
max ≡
  TRY
    IF ⦃ ´a <s ´b ⦄ THEN
      ´ret_int :== ´b;;
      ´          :== Return;;
      THROW
    ELSE
      SKIP
    FI;;
    ´ret_int :== ´a;;
    ´exn_var :== Return;
    THROW;
    GUARD DontReach {}
      SKIP;;
  CATCH
    SKIP
  END
```

NICTA

```
int max(int a, int b) {
    if (a < b) {
        return b;
    }
    return a;
}
```

```
max a b ≡
  if a < b then b else a
```

```
max ≡
  TRY
    IF {| ´a <s ´b |} THEN
      ´ret_int :== ´b;;
      ´         :== Return;;
      THROW
    ...
      ´ret_int :== ´a;;
      ´exn_var :== Return;
      THROW;
      GUARD DontReach {}
        SKIP;;
  CATCH
    SKIP
  END
```

```
int gcd(int a, int b) {
    int c;
    while (a != 0) {
        c = a;
        a = b % a;
        b = c;
    }
    return b;
}
```

```c
int gcd(int a, int b) {
    int c;
    while (a != 0) {
        c = a;
        a = b % a;
        b = c;
    }
    return b;
}
```

```
gcd ≡
  TRY
    NonDetInit ´c ´c_update
    WHILE {| ´a ≠ 0 |} THEN
      ´c :== ´a;;
      GUARD Div0 {| ´a ≠ 0 |}
        ´a :== ´b mod ´a;;
      ´b :== ´c
    OD;;
    ´ret_int :== ´b;
    ´exn_var :== Return;
    THROW;
    GUARD DontReach {}
      SKIP
  CATCH
    SKIP
  END
```

```
gcd ≡
  TRY
    NonDetInit ´c ´c_update
    WHILE {| ´a ≠ 0 |} THEN
      ´c :== ´a;;
```

```c
int gcd(int a, int b) {
    int c;
    while (a != 0) {
        c = a;
        a = b % a;
        b = c;
    }
    return b;
}
```

```
gcd a b ≡ do
  (a, b) ←
    while (λ(a, b) _. a ≠ 0)
      (λ(a, b). return (b mod a, a))
                                  (a, b);
  return b
od
```

```
    THROW;
    GUARD DontReach {}
      SKIP
  CATCH
    SKIP
  END
```

```
void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
```

```
void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
```

```
swap ≡
TRY
  GUARD C_Guard {|c_guard ´a|}
   (´t :== h_val (hrs_mem ´t_hrs) ´a);;
  GUARD C_Guard {|c_guard ´a|}
   (GUARD C_Guard {|c_guard ´b|}
     (´globals :==
        t_hrs_'_update
         (hrs_mem_update
           (heap_update ´a
             (h_val (hrs_mem ´t_hrs)
               ´b))))));;
  GUARD C_Guard {|c_guard ´b|}
   (´globals :==
      t_hrs_'_update
       (hrs_mem_update
         (heap_update ´b ´t)))
CATCH
  SKIP
END
```

```c
void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
```

```
swap ≡
TRY
  GUARD C_Guard {|c_guard ´a|}
   (´t :== h_val (hrs_mem ´t_hrs) ´a);;
  GUARD C_Guard {|c_guard ´a|}
   (GUARD C_Guard {|c_guard ´b|}
```

```
swap a b ≡
  do guard (λs. c_guard a);
     t ← gets (λs. h_val (hrs_mem (t_hrs_' s)) a);
     guard (λs. c_guard b);
     modify
      (λs. t_hrs_'_update
             (hrs_mem_update
               (heap_update a
                 (h_val (hrs_mem (t_hrs_' s)) b))) s);
     modify
      (t_hrs_'_update
        (hrs_mem_update (heap_update b t)))
  od
```

C Heap Semantics

```
void swap(int *a, int *b)
{
     int t = *a;
     *a = *b;
     *b = t;
}
```

C Heap Semantics

```
void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
```

C Heap Semantics

```
void swap(int *a, int *b)
{
     int t = *a;
     *a = *b;
     *b = t;
}
```

C Heap Semantics

```
void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
```

```
swap a b ≡
  do guard (λs. c_guard a);
     t ← gets (λs. h_val (hrs_mem (t_hrs_' s)) a);
     guard (λs. c_guard b);
     modify
      (λs. t_hrs_'_update
             (hrs_mem_update
               (heap_update a
                 (h_val (hrs_mem (t_hrs_' s)) b))) s);
     modify
      (t_hrs_'_update
        (hrs_mem_update (heap_update b t)))
  od
```

Ensure “a” is aligned, non-NULL.

```
swap a b ≡
  do guard (λs. c_guard a);
     t ← gets (λs. h_val (hrs_mem (t_hrs_' s)) a);
     guard (λs. c_guard b);
     modify
      (λs. t_hrs_'_update
             (hrs_mem_update
               (heap_update a
                 (h_val (hrs_mem (t_hrs_' s)) b))) s);
     modify
      (t_hrs_'_update
        (hrs_mem_update (heap_update b t)))
  od
```

```
swap a b ≡
  do guard (λs. c_guard a);
     t ← gets (λs. h_val (hrs_mem (t_hrs_' s)) a);
     guard (λs. c_guard b);
     modify
      (λs. t_hrs_'_update
             (hrs_mem_update
               (heap_update a
                 (h_val (hrs_mem (t_hrs_' s)) b))) s);
     modify
      (t_hrs_'_update
        (hrs_mem_update (heap_update b t)))
  od
```

Ensure "a" is aligned, non-NULL.

Decode the bytes at "a".

```
swap a b ≡
  do guard (λs. c_guard a);
     t ← gets (λs. h_val (hrs_mem (t_hrs_' s)) a);
     guard (λs. c_guard b);
     modify
      (λs. t_hrs_'_update
             (hrs_mem_update
               (heap_update a
                 (h_val (hrs_mem (t_hrs_' s)) b))) s);
     modify
      (t_hrs_'_update
        (hrs_mem_update (heap_update b t)))
  od
```

Ensure “a” is aligned, non-NULL.

Decode the bytes at “a”.

Decode the bytes at “b”,
encode back into bytes,
store at “a”.

In Short

Treatment of Heap is Promising Work in Progress

Word Arithmetic Made More Beautiful

- having to worry about overflow is drudgery...
- make the tools do it as automatically as possible

Conclusion

Ad hoc **Proof** makes ad hoc hackery

- sound, and
- beautiful

Tools:

– ssrg.nicta.com.au/software/TS/{c-parser,graph-refine}
– ssrg.nicta.com.au/projects/TS/autocorres/