Trickle: Automated Infeasible Path Detection Using All Minimal Unsatisfiable Subsets

NICTA

Bernard Blackham[†], Mark Liffiton[‡], Gernot Heiser[†]

[†] School of Computer Science & Engineering, UNSW
Software Systems Research Group, NICTA
Sydney, Australia

[‡] Illinois Wesleyan University

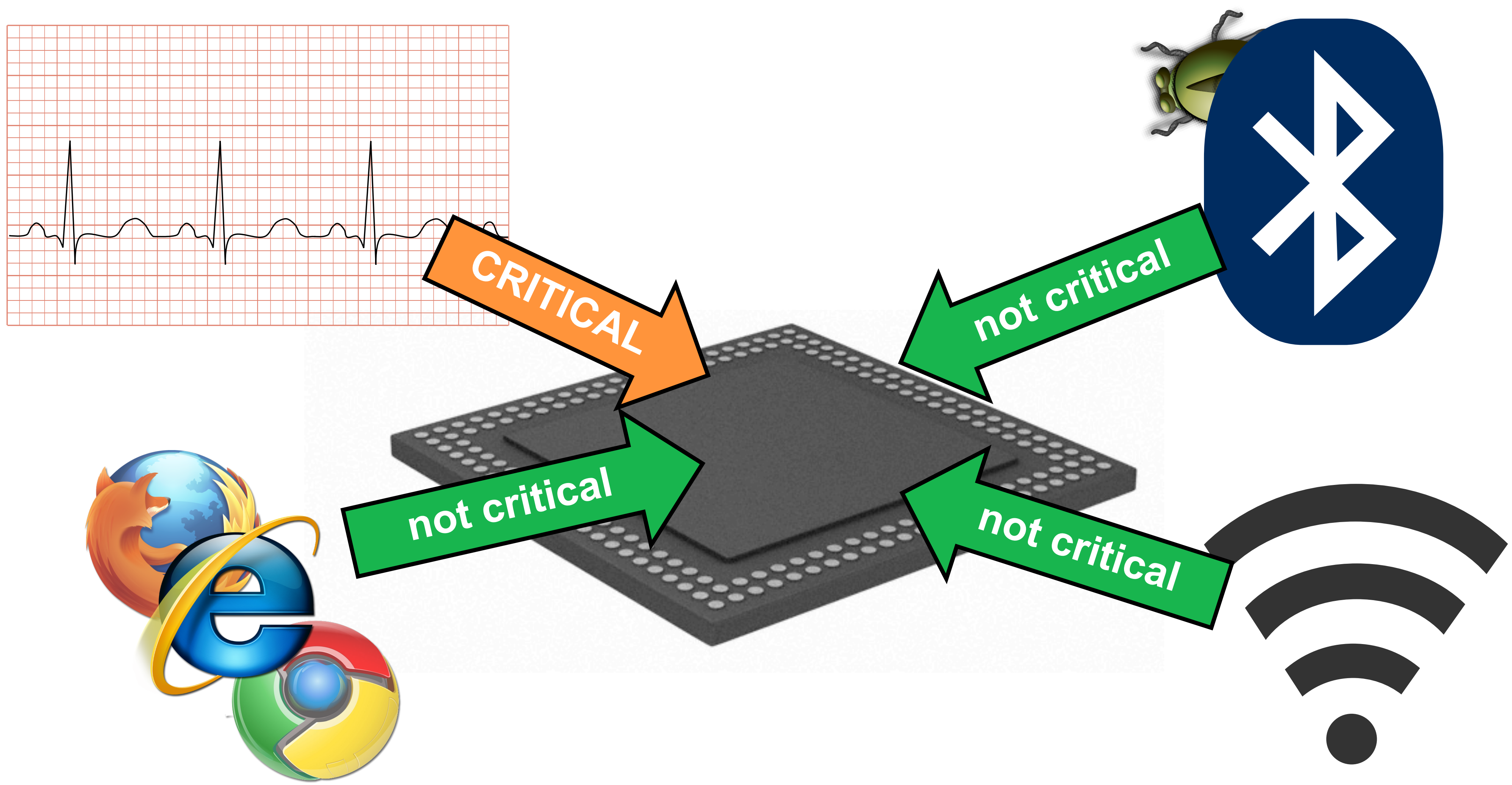
Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Motivation

NICTA

- The desire for a *trustworthy* kernel to build reliable **mixed-criticality** **real-time** systems



Motivation

NICTA

- The desire for a *trustworthy* kernel to build reliable **mixed-criticality** **real-time** systems

- Using **seL4** to guarantee:
  - **functional correctness** through formal proof
    (Klein et al., SOSP 2009)
  - **timing constraints** through sound WCET analysis
    (Blackham et al., RTSS 2011)

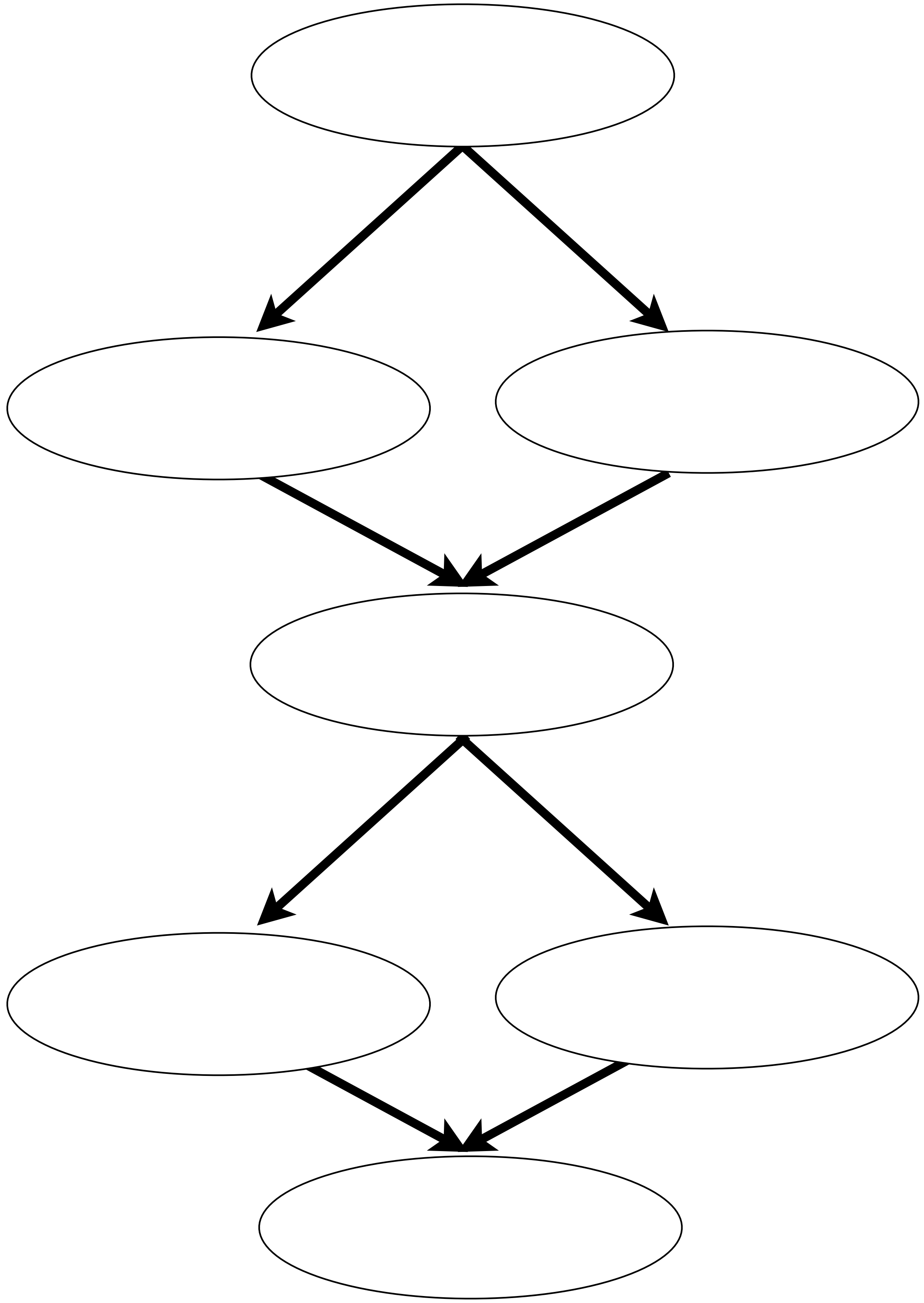
Motivation

NICTA

- The accuracy of WCET estimates directly affects the hardware provisioning of such systems (e.g. CPU speed) ➔ cost
- Inaccurate WCET estimates by static analysis can be caused by **infeasible paths**
- By excluding culprit infeasible paths
  ➔ less provisioning required
  ➔ reduced hardware cost

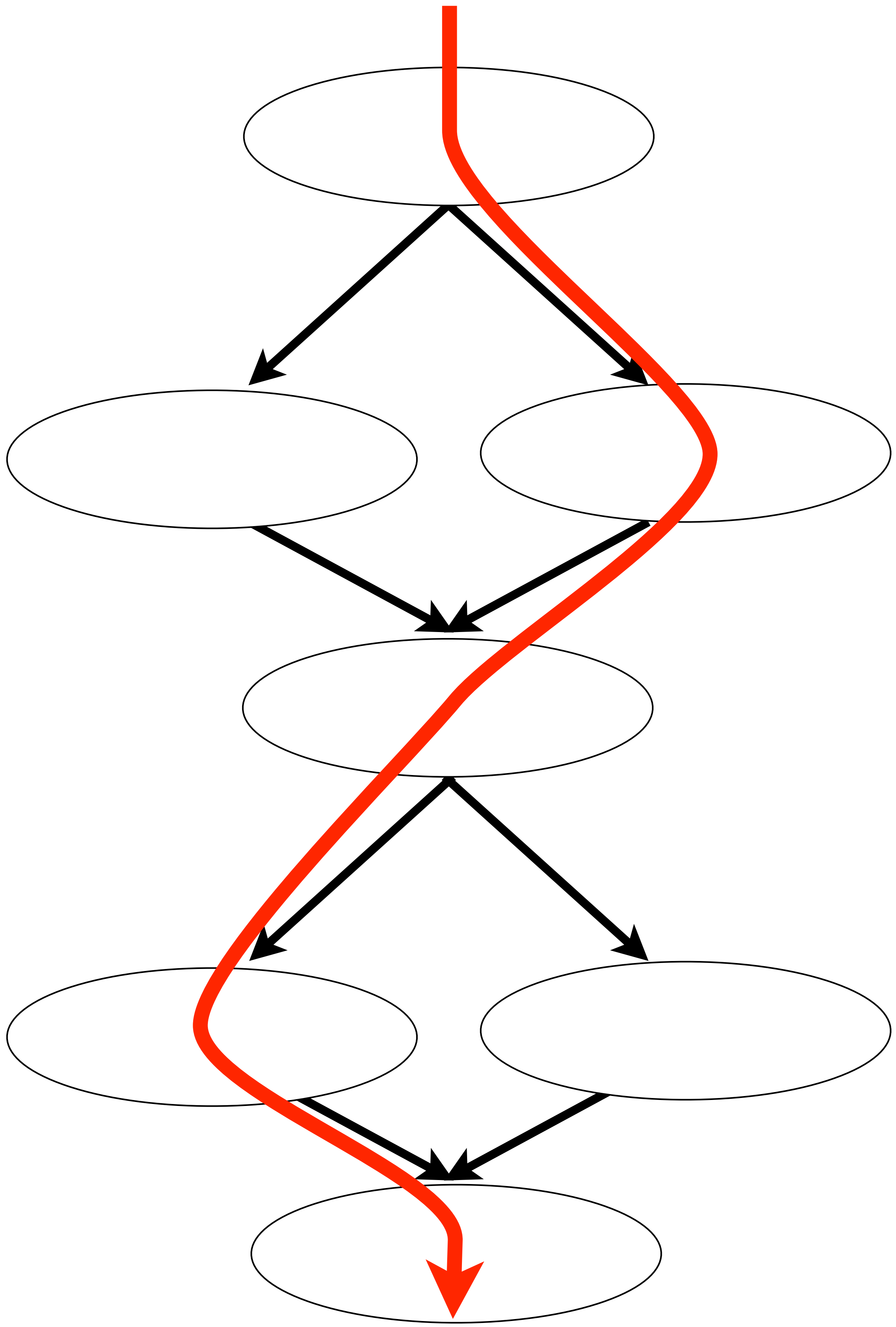
Infeasible paths

```
int f(int a)
{
    if ((a & 4)==0)
        ...
    else
        ...
    ...
    if ((a & 4)==0)
        ...
    else
        ...
}
```



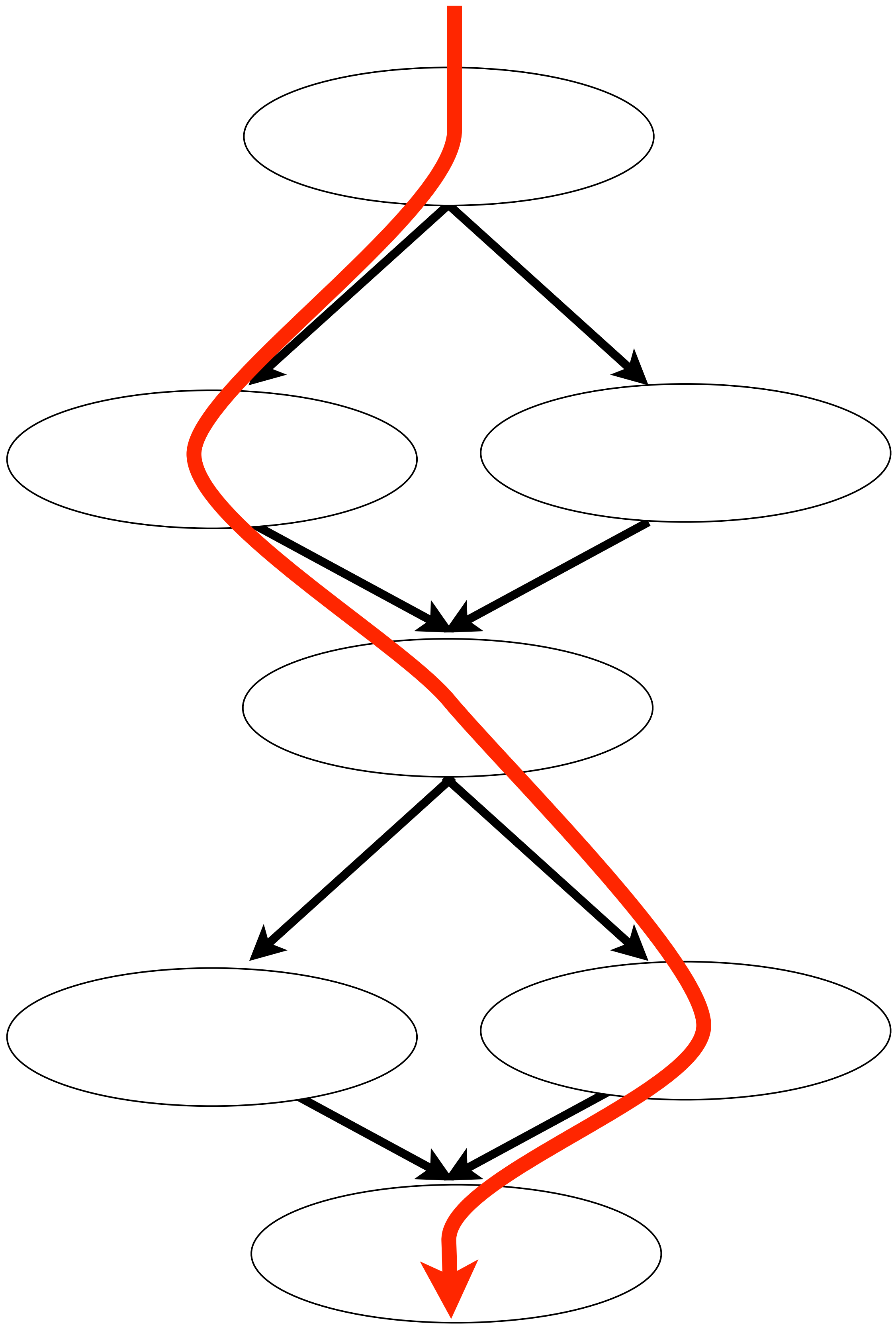
Infeasible paths

```
int f(int a)
{
    if ((a & 4)==0)
        ...
    else
        ...
    ...
    if ((a & 4)==0)
        ...
    else
        ...
}
```

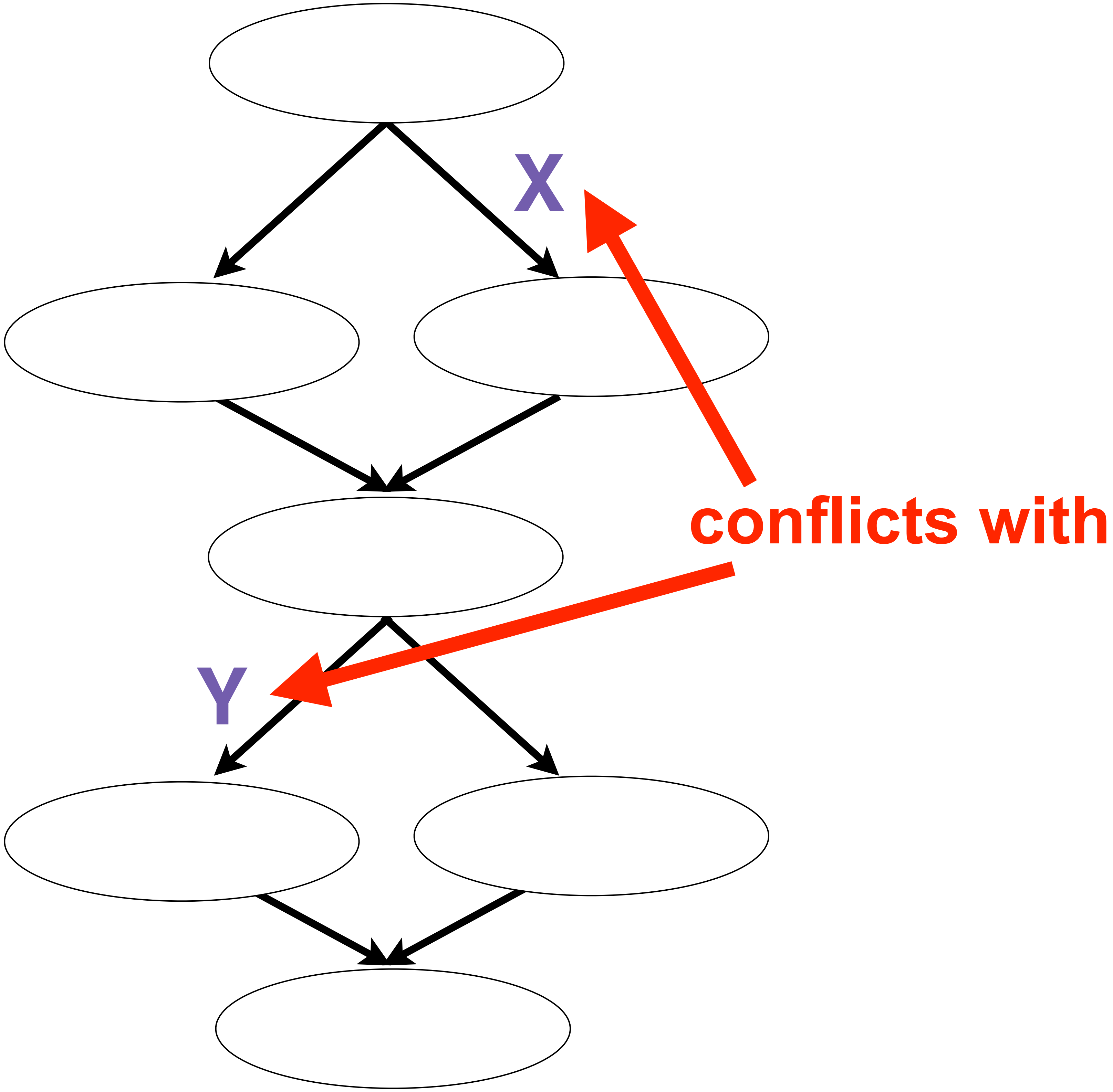


Infeasible paths

```
int f(int a)
{
    if ((a & 4)==0)
        ...
    else
        ...
    ...
    if ((a & 4)==0)
        ...
    else
        ...
}
```



Infeasible paths



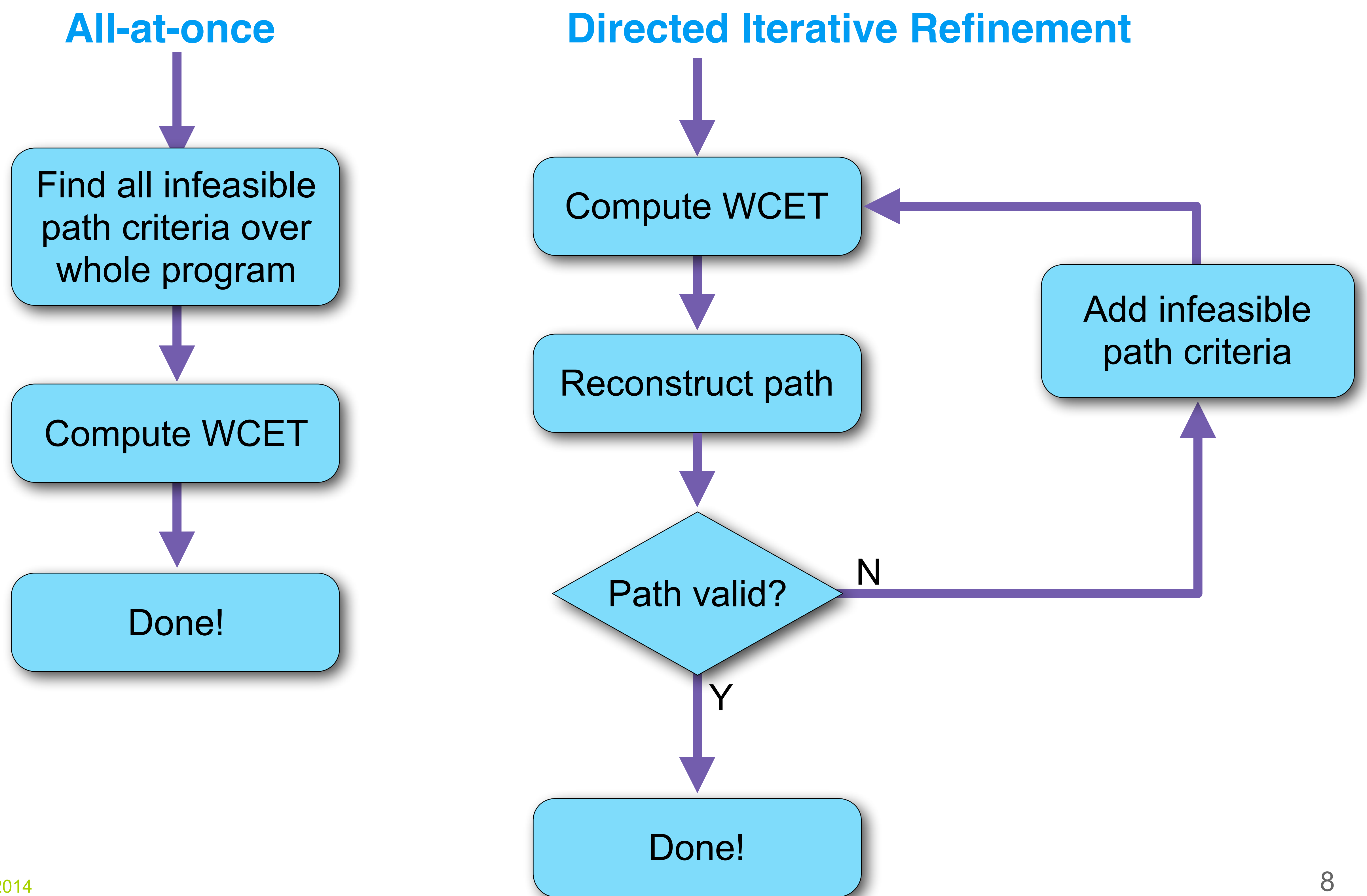
Background

NICTA

- **Chronos** used to compute WCET via **IPET**
- **Sequoll** can validate manually provided infeasible path information using a model checker

  (Blackham & Heiser, RTAS 2013)
- Reduces risk of human error when specifying infeasible path information
- **Can we find infeasible paths automatically?**

Finding infeasible path constraints



seL4 is large

NICTA

- Small by microkernel standards
- Large by WCET standards

C source

~8,700 lines

316 functions

76 loops

Binary (ARM)

~10,000 instructions

228 functions

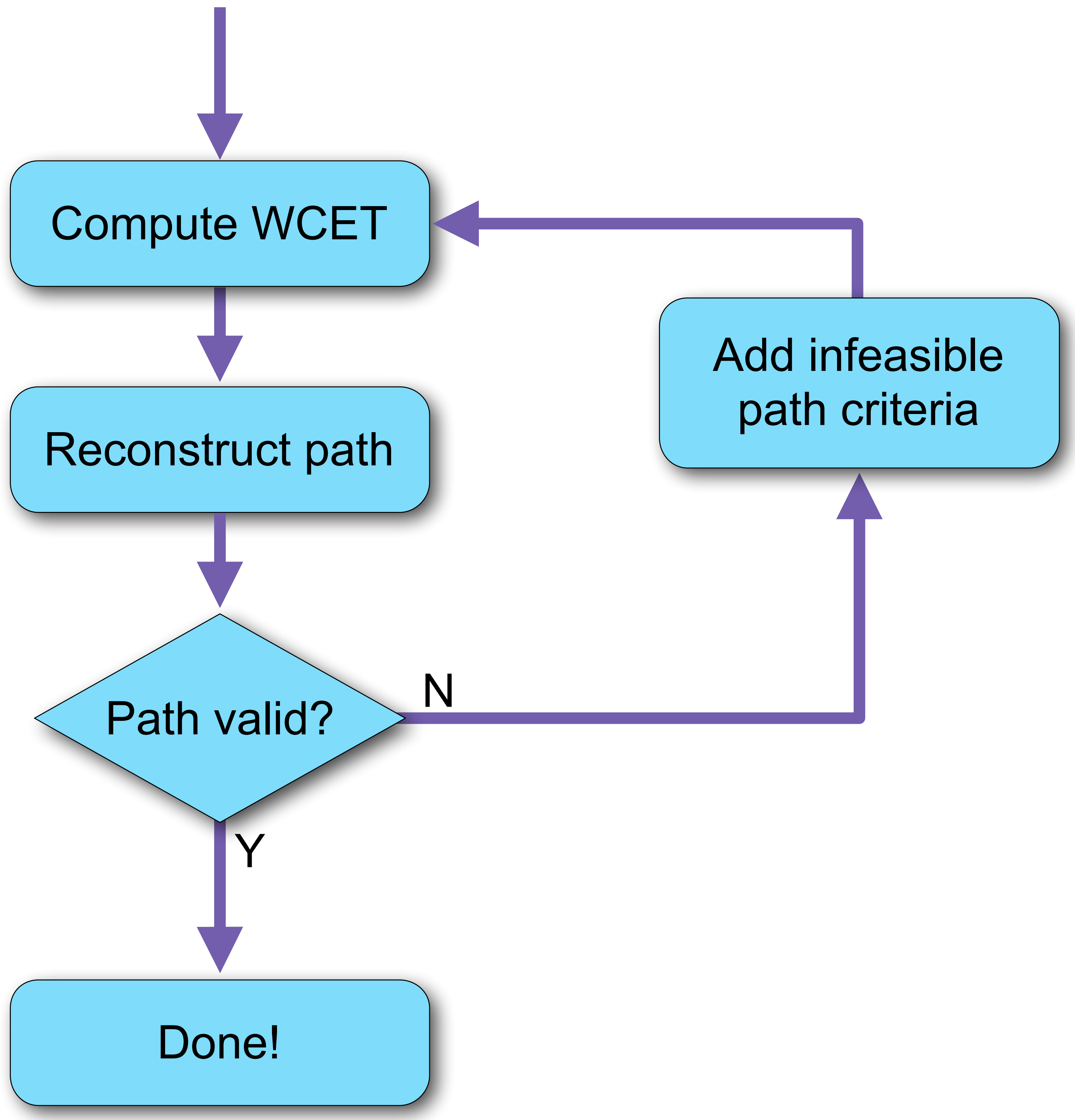
56 loops

2,384 basic blocks

~400,000 basic blocks when inlined

Trickle: Directed Iterative Refinement

NICTA



Types of infeasible path criteria

NICTA

- Infeasible path criteria may arise because of:

local constraints

global program invariants

```
y = count;
if (y > 15)
   y = 15;
while (y > 0)
{
   ...
   y = y - 1;
}
```

```
// count guaranteed to be <= 15
y = count;

while
{

   y = y -
}
```

Types of infeasible path criteria

NICTA

- Infeasible path criteria may arise because of:

local constraints

```
y = count;
if
   y =
while
{

   y = y -
}
```

global program invariants

```
// count guaranteed to be <= 15
y = count;

while (y > 0)
{
   ...
   y = y - 1;
}
```

Types of infeasible path criteria

- Infeasible path criteria may arise because of:

local constraints

```
y = count;
if (y > 15)
   y = 15;
while (y > 0)
{
   ...
   y = y - 1;
}
```

global program invariants

```
// count guaranteed to be <= 15
y = count;

while (y > 0)
{
   ...
   y = y - 1;
}
```

Detecting Infeasible Paths

NICTA

- **Sequoll** computes a **branch condition** for every conditional branch or conditional instruction, in terms of *SSA variables*

- Once a worst-case path is identified, **Trickle** collects all branch conditions required to execute it, as *SMT expressions*

- All branch conditions are given to an **SMT solver** to find a satisfying assignment

Detecting Infeasible Paths

NICTA

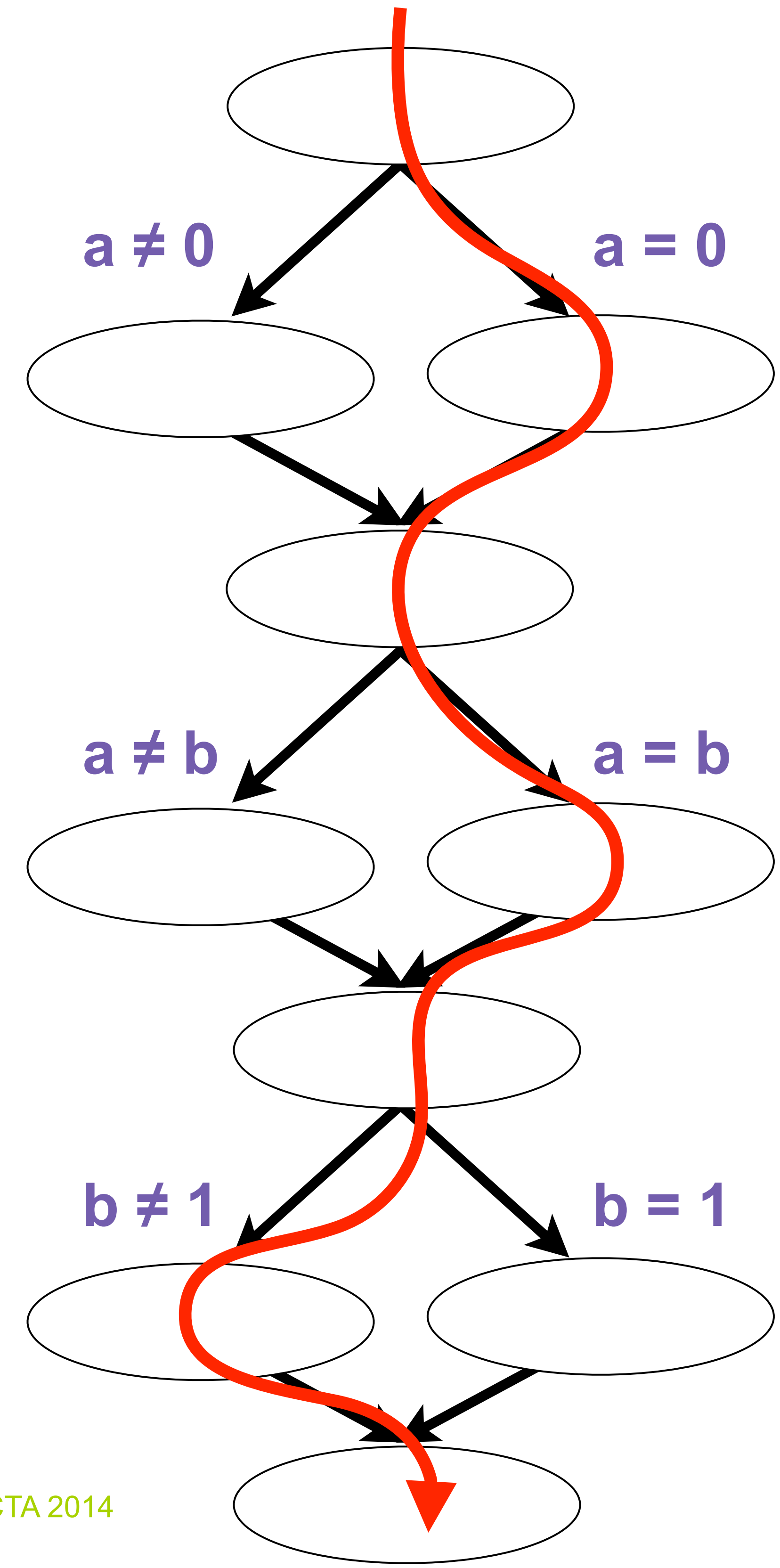
- If SMT solver finds a satisfying assignment, path is declared **feasible***

- If SMT solver shows that the constraints are unsatisfiable, the path is **infeasible**
  - ➡ An **unsatisfiable subset** is returned

* up to the limit of reasoning ability of the SMT solver and Trickle

Detecting Infeasible Paths

NICTA

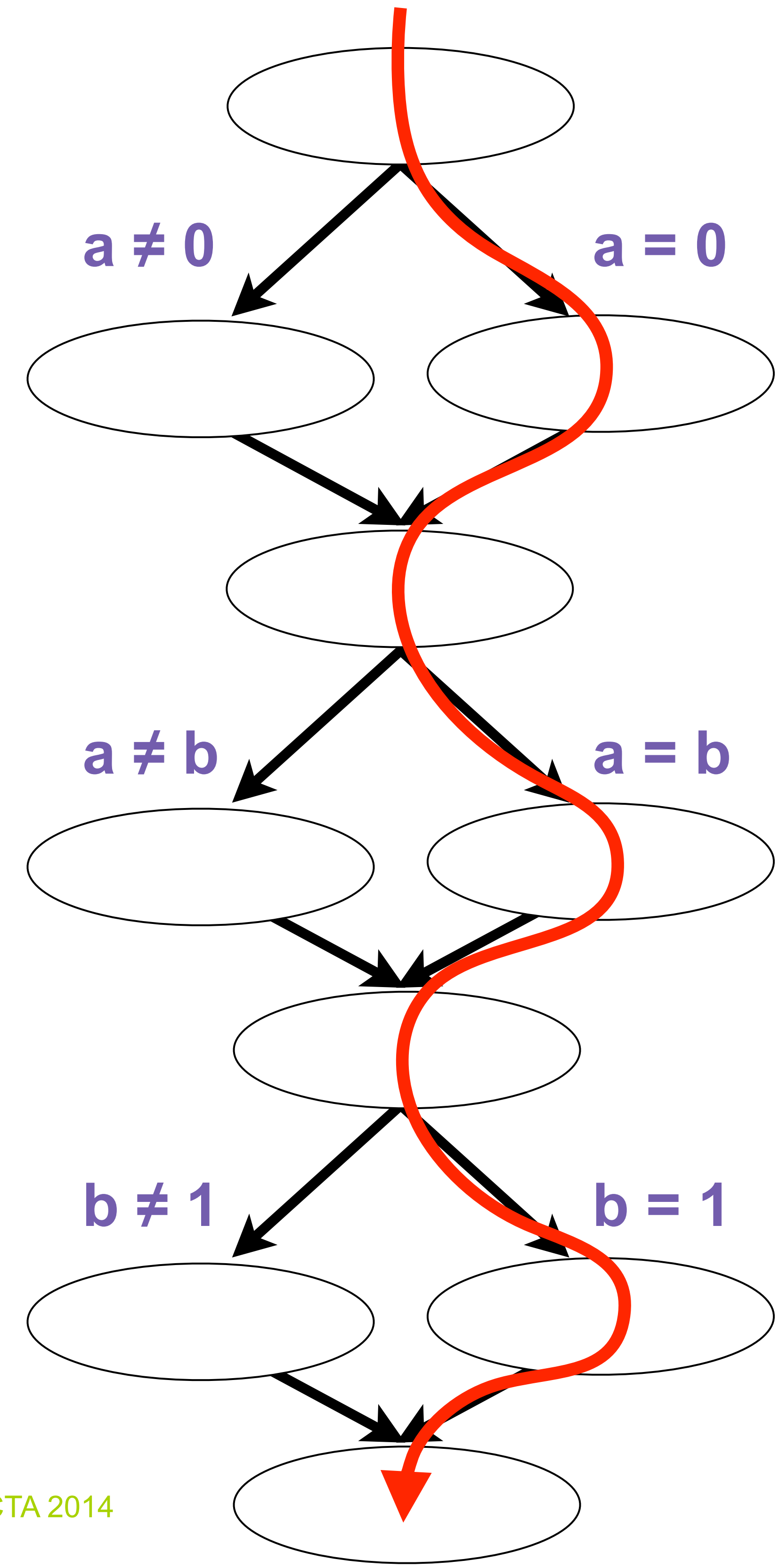


A: a=0
B: a=b
C: b≠1

Satisfiable! 👍

Assignment: a=0, b=0

Detecting Infeasible Paths

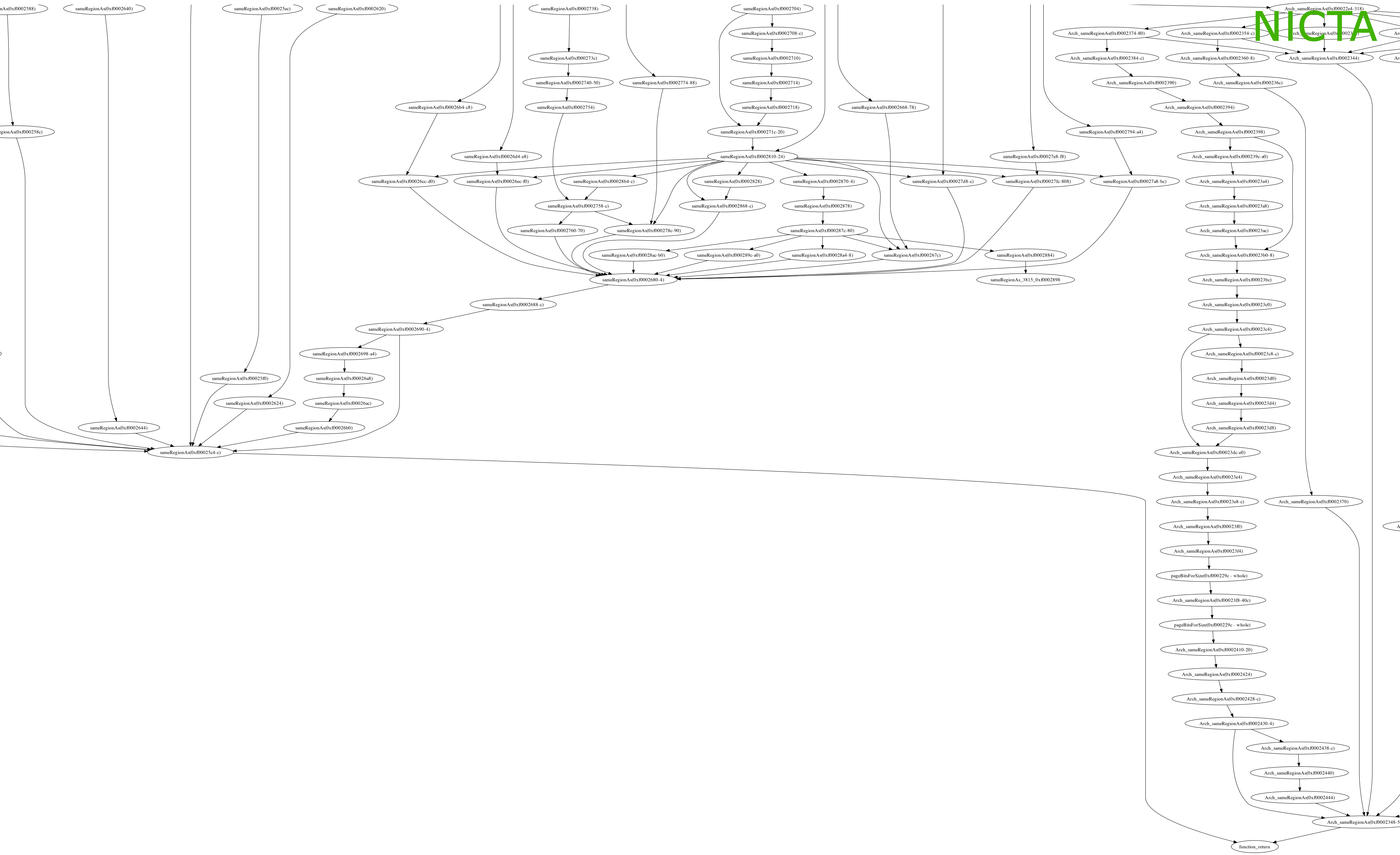


A: a=0
B: a=b
C: b=1

Unsatisfiable 👎

Minimal **U**nsatisfiable **S**ubset:
{A,B,C}

$E_A + E_B + E_C < 3$



NICTA

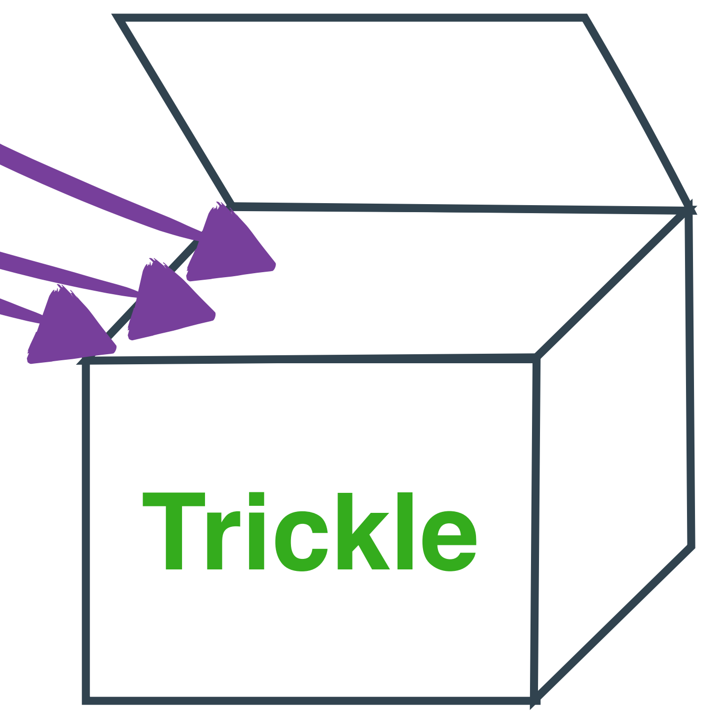
- A path may contain **1000s of instructions**
- And **100s of branch constraints**
- And may contain **several MUSes**
- SMT solvers typically only find **one**
- Can we reduce the number of refinement iterations?

Trickle: Enter CAMUS

Sequoll – framework for analysis of compiled ARM binaries

CAMUS – **C**ompute **A**ll **M**inimal **U**nsatisfiable **S**ubsets

Yices – SMT solver



CAMUS

NICTA

- Developed by Liffiton & Sakallah (JAR 2008)

- Finds **all** minimal unsatisfiable subsets of a given set of constraints

➡i.e. finds all infeasible path constraints along a given path

CAMUS

NICTA

- The **worst-case run time** of the CAMUS algorithm is **exponential** in the number of MUSes (+ SMT solver time)

How can we prevent this?

CAMUS: Sliding Window

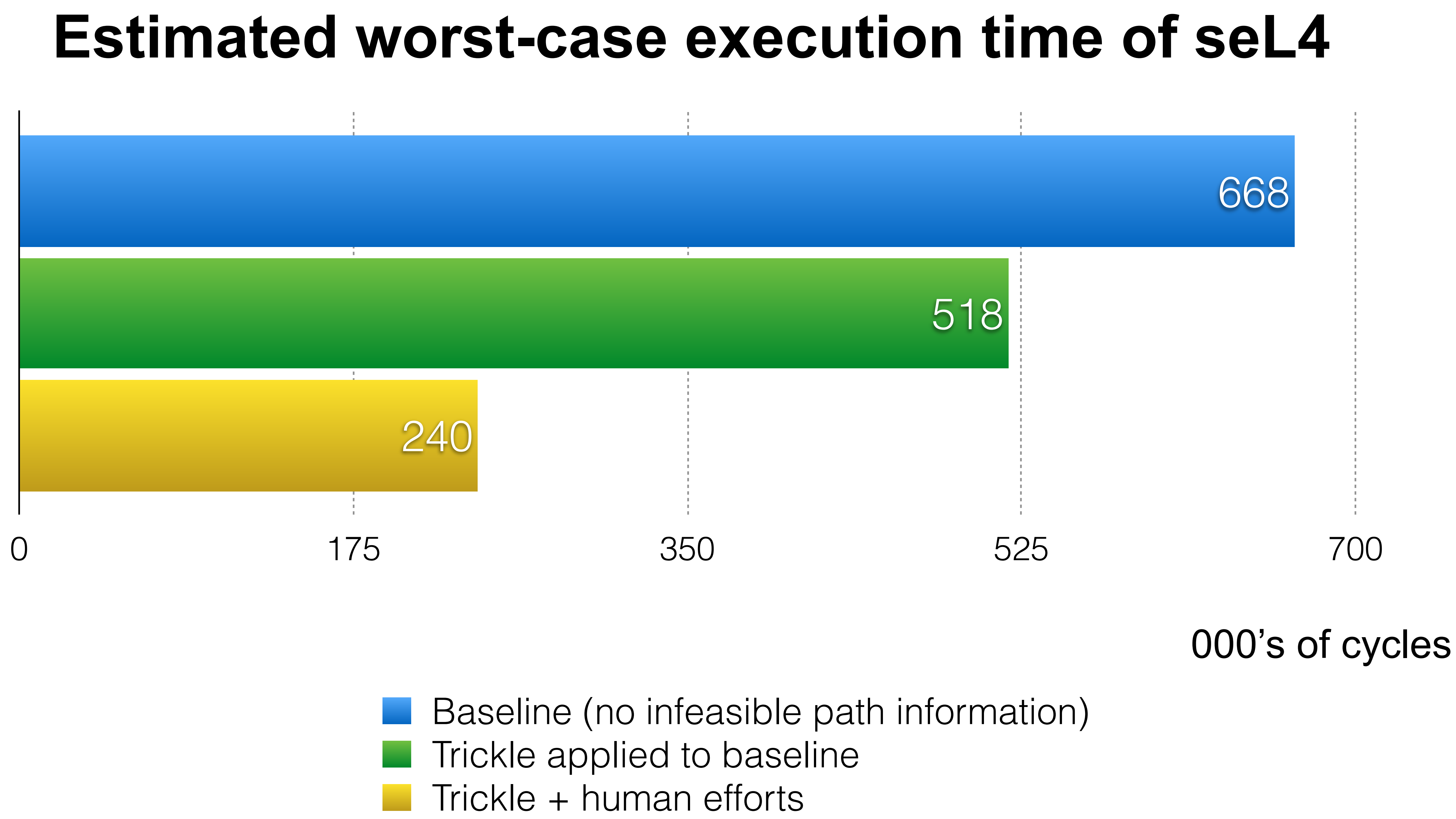
NICTA

A B C D E F G H I J K L M N O

- Try with a smaller subset of constraints first
- Choose constraints close together on path as they are more likely to conflict
- Increase size of window and repeat if no constraints found

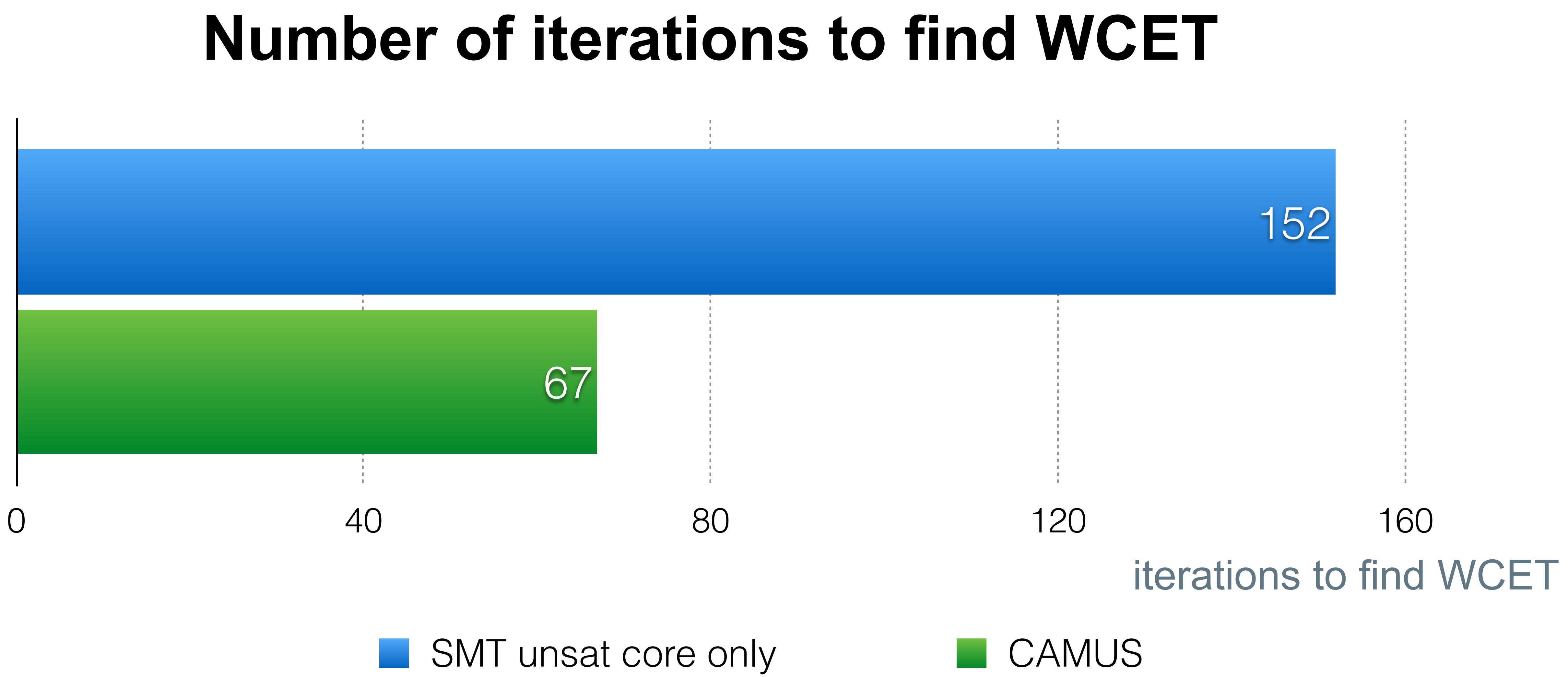
Results

NICTA



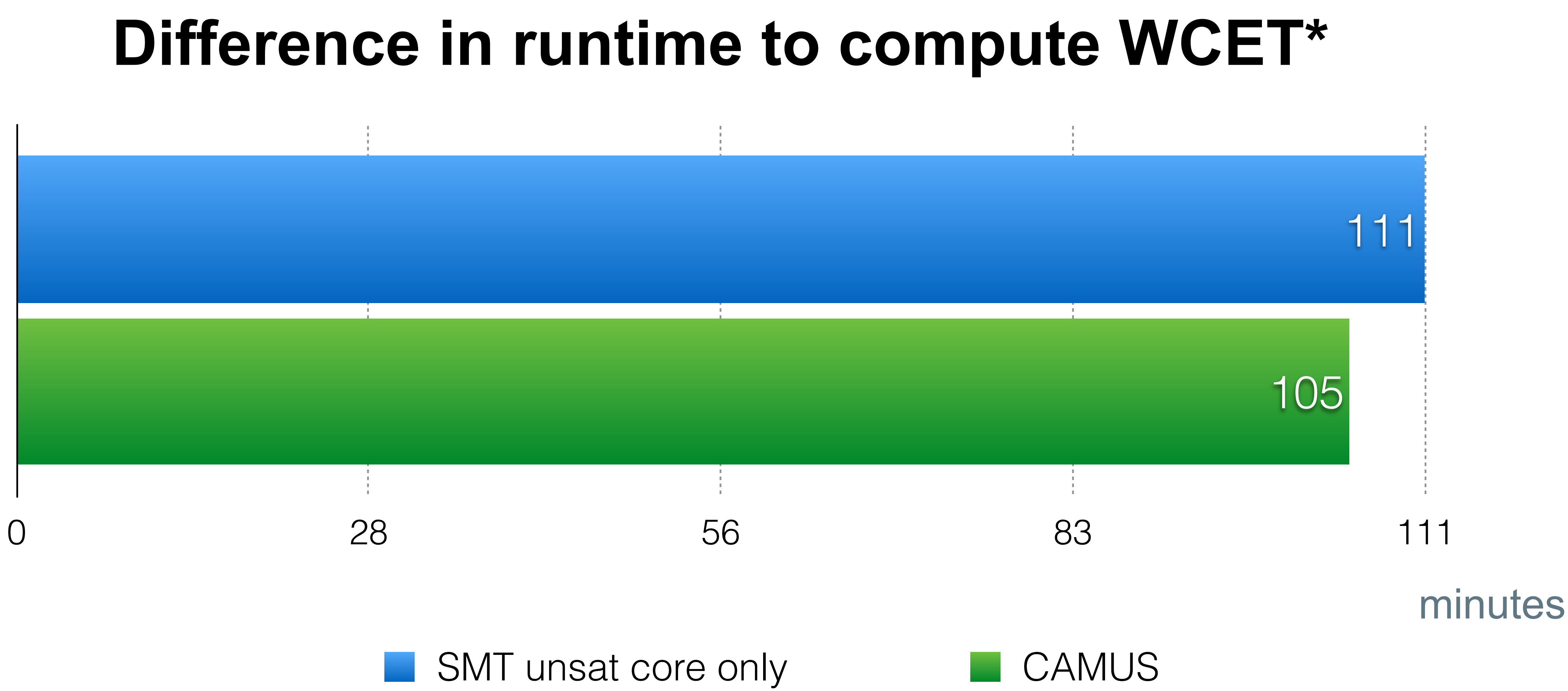
Results

NICTA



Results

NICTA



* Implementation-specific compilation overheads subtracted

Limitations

- What about **loops**?
  - Analysis is limited to loop-free regions of code
  - Limitation of SMT solvers
  - Limitations of IPET method

- **Run time** of analysis is long
  - Full analysis takes > 2 hours

Research directions

- Integrate Trickle with **proof invariants**
- Find infeasible paths **across loop iterations**
- Improved CAMUS algorithm to **avoid exponential behaviour** (no need for sliding window)
- Improve **memory aliasing** analysis

Summary

NICTA

Trickle is able to:

- automatically compute infeasible path information on compiled ARM binaries
- improve WCET estimates of an IPET analysis
- reason about more interesting constraints than integer intervals (e.g. bit arithmetic)

➡ reduce scope for errors in WCET analysis!

Download it!

http://www.ssrg.nicta.com.au/software/TS/wcet-tools

Bernard.Blackham@nicta.com.au