Sliced Path Prefixes: An Effective Method to Enable Refinement Selection

Dirk Beyer, Stefan Löwe, Philipp Wendler

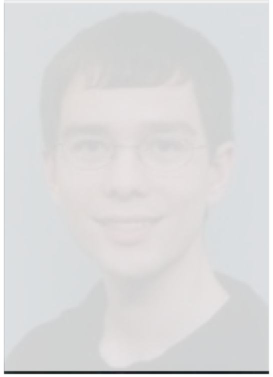
SoSy-Lab
Software Systems

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

We want Refinement Selection !!!

Because straight-forward interpolation completely and utterly sucks !!!11!

Dirk Beyer, Stefan Löwe Philipp Wendler

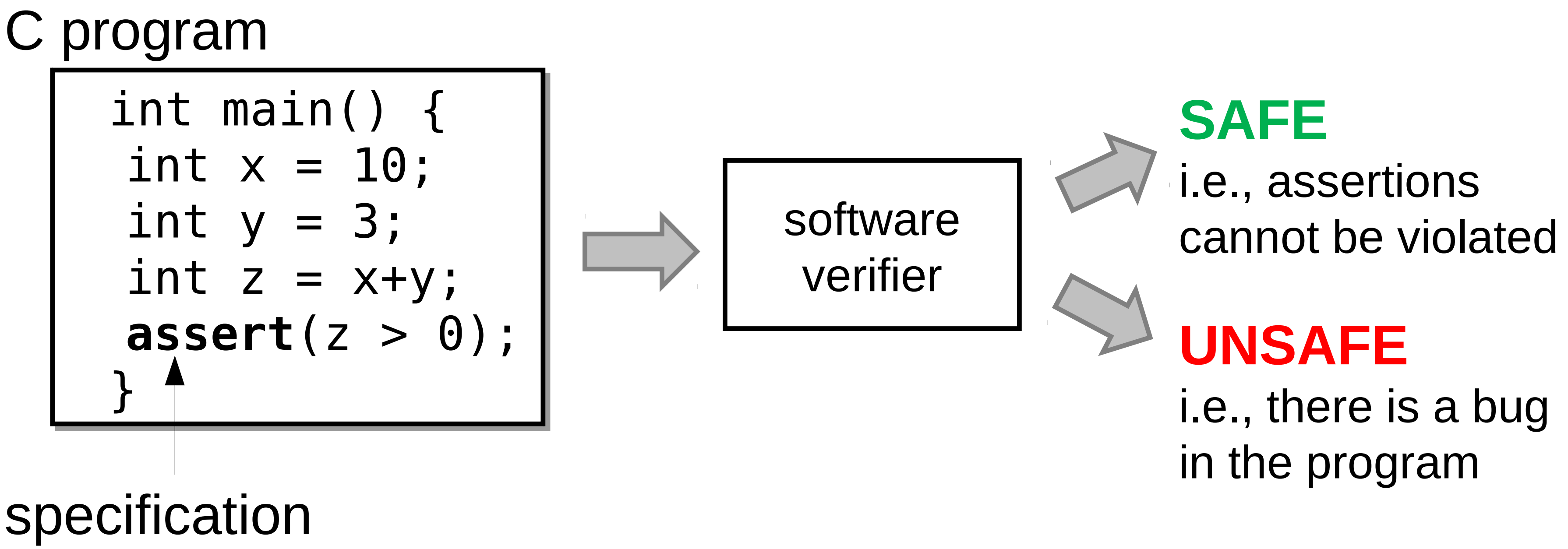


SoSy-Lab
Software Systems

UNIVERSITÄT
PASSAU
Fakultät für Informatik und Mathematik

Software Verification

Goal: Build an **automatic** software verifier



C program

```
int main() {
 int x = 10;
 int y = 3;
 int z = x+y;
 assert(z > 0);
}
```

specification

software verifier

SAFE

i.e., assertions cannot be violated

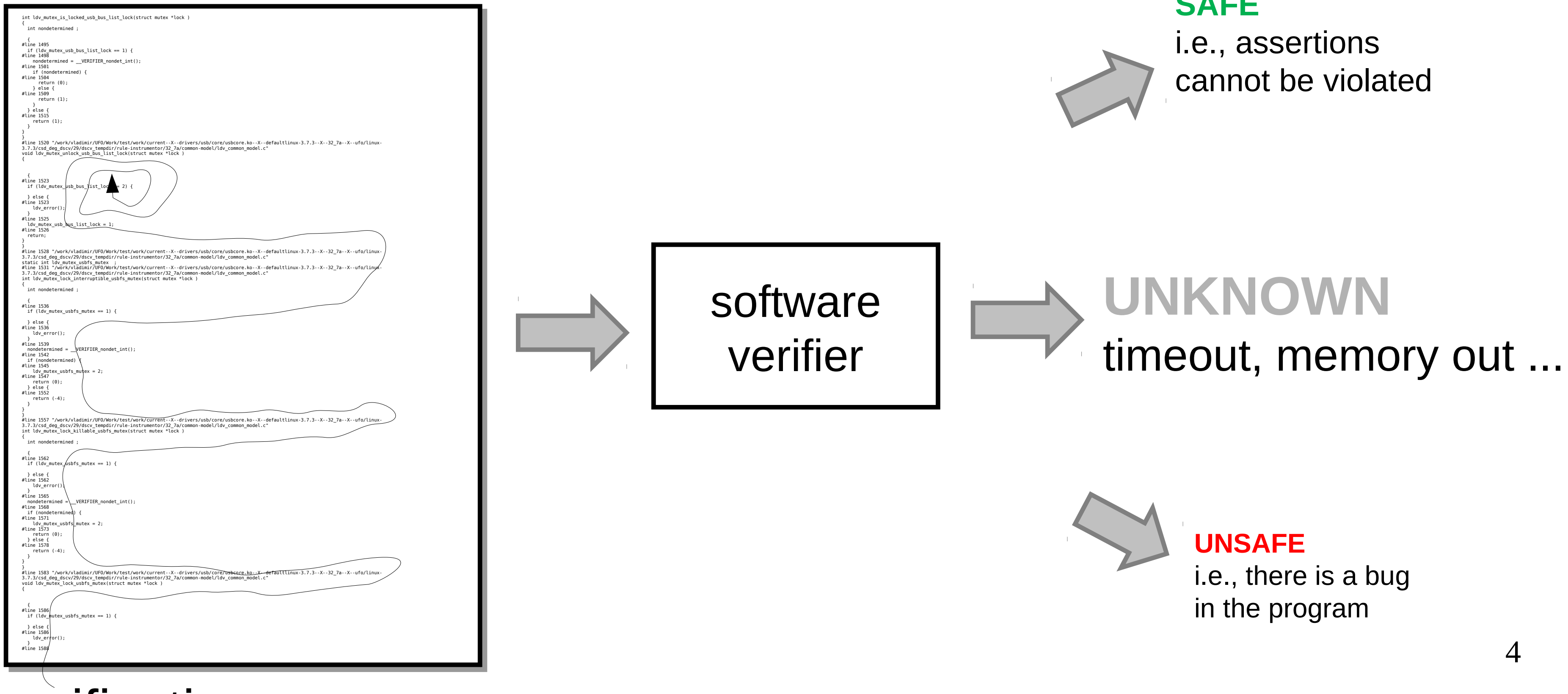
UNSAFE

i.e., there is a bug in the program

Software Verification

Goal: Build an **automatic** software verifier

C program – Linux Device Driver



```
int ldv_mutex_is_locked_usb_bus_list_lock(struct mutex *lock )
{
  int nondetermined ;

  {
#line 1495
  if (ldv_mutex_usb_bus_list_lock == 1) {
#line 1498
    nondetermined = __VERIFIER_nondet_int();
#line 1501
    if (nondetermined) {
#line 1504
      return (0);
    } else {
#line 1509
      return (1);
    }
  } else {
#line 1515
    return (1);
  }
}
}
#line 1520 "/work/vladimir/UFO/Work/test/work/current--X--drivers/usb/core/usbcore.ko--X--defaultlinux-3.7.3--X--32_7a--X--ufo/linux-
3.7.3/csd_deg_dscv/29/dscv_tempdir/rule-instrumentor/32_7a/common-model/ldv_common_model.c"
void ldv_mutex_unlock_usb_bus_list_lock(struct mutex *lock )
{


  {
#line 1523
  if (ldv_mutex_usb_bus_list_lock == 2) {

  } else {
#line 1523
    ldv_error();
  }
#line 1525
  ldv_mutex_usb_bus_list_lock = 1;
#line 1526
  return;
}
}
#line 1528 "/work/vladimir/UFO/Work/test/work/current--X--drivers/usb/core/usbcore.ko--X--defaultlinux-3.7.3--X--32_7a--X--ufo/linux-
3.7.3/csd_deg_dscv/29/dscv_tempdir/rule-instrumentor/32_7a/common-model/ldv_common_model.c"
static int ldv_mutex_usbfs_mutex  ;
#line 1531 "/work/vladimir/UFO/Work/test/work/current--X--drivers/usb/core/usbcore.ko--X--defaultlinux-3.7.3--X--32_7a--X--ufo/linux-
3.7.3/csd_deg_dscv/29/dscv_tempdir/rule-instrumentor/32_7a/common-model/ldv_common_model.c"
int ldv_mutex_lock_interruptible_usbfs_mutex(struct mutex *lock )
{
  int nondetermined ;

  {
#line 1536
  if (ldv_mutex_usbfs_mutex == 1) {

  } else {
#line 1536
    ldv_error();
  }
#line 1539
  nondetermined = __VERIFIER_nondet_int();
#line 1542
  if (nondetermined) {
#line 1545
    ldv_mutex_usbfs_mutex = 2;
#line 1547
    return (0);
  } else {
#line 1552
    return (-4);
  }
}
}
#line 1557 "/work/vladimir/UFO/Work/test/work/current--X--drivers/usb/core/usbcore.ko--X--defaultlinux-3.7.3--X--32_7a--X--ufo/linux-
3.7.3/csd_deg_dscv/29/dscv_tempdir/rule-instrumentor/32_7a/common-model/ldv_common_model.c"
int ldv_mutex_lock_killable_usbfs_mutex(struct mutex *lock )
{
  int nondetermined ;

  {
#line 1562
  if (ldv_mutex_usbfs_mutex == 1) {

  } else {
#line 1562
    ldv_error();
  }
#line 1565
  nondetermined = __VERIFIER_nondet_int();
#line 1568
  if (nondetermined) {
#line 1571
    ldv_mutex_usbfs_mutex = 2;
#line 1573
    return (0);
  } else {
#line 1578
    return (-4);
  }
}
}
#line 1583 "/work/vladimir/UFO/Work/test/work/current--X--drivers/usb/core/usbcore.ko--X--defaultlinux-3.7.3--X--32_7a--X--ufo/linux-
3.7.3/csd_deg_dscv/29/dscv_tempdir/rule-instrumentor/32_7a/common-model/ldv_common_model.c"
void ldv_mutex_lock_usbfs_mutex(struct mutex *lock )
{


  {
#line 1586
  if (ldv_mutex_usbfs_mutex == 1) {

  } else {
#line 1586
    ldv_error();
  }
#line 1588
```

SAFE
i.e., assertions
cannot be violated

UNKNOWN
timeout, memory out ...

UNSAFE
i.e., there is a bug
in the program

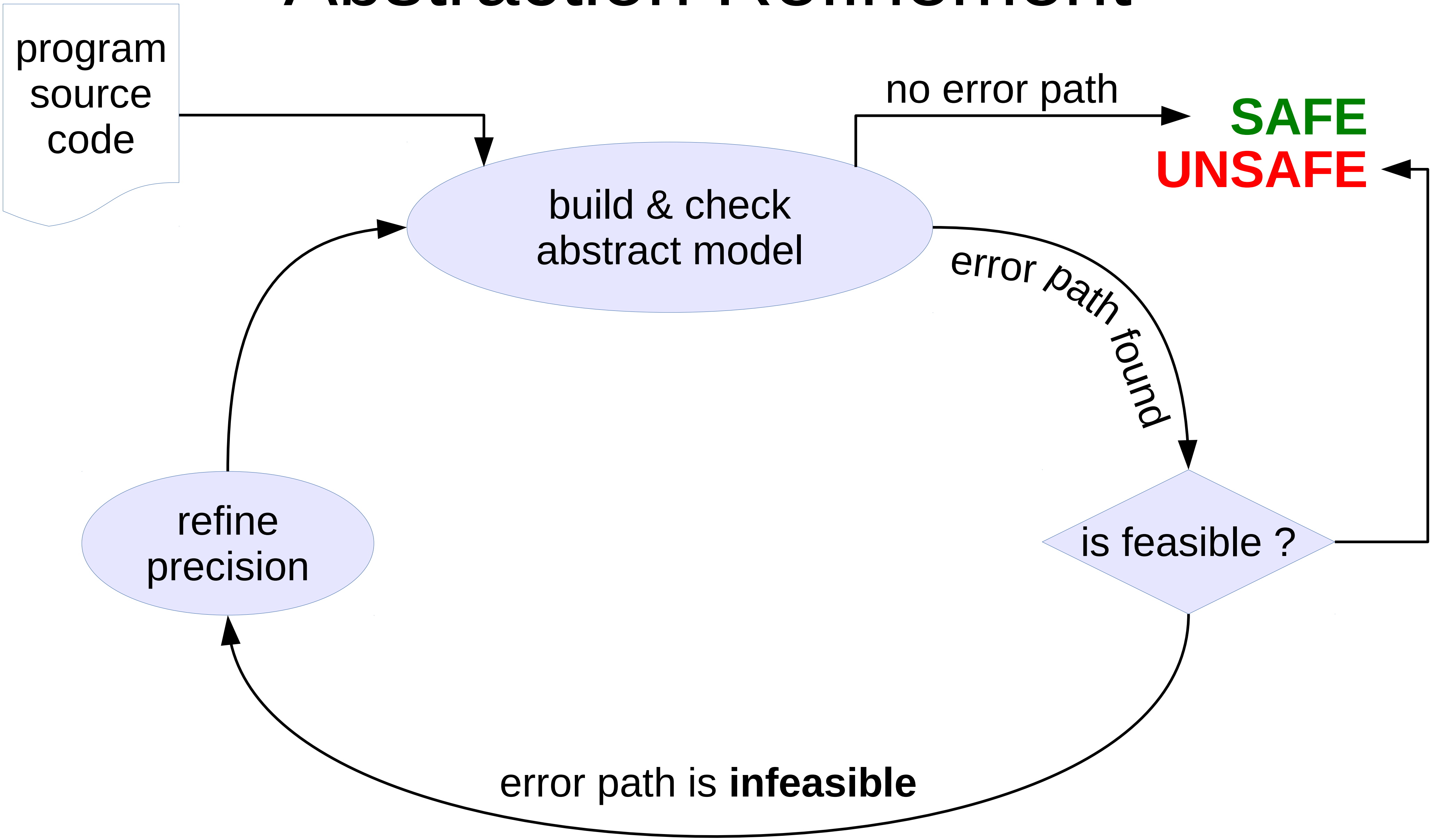
specification

Abstraction

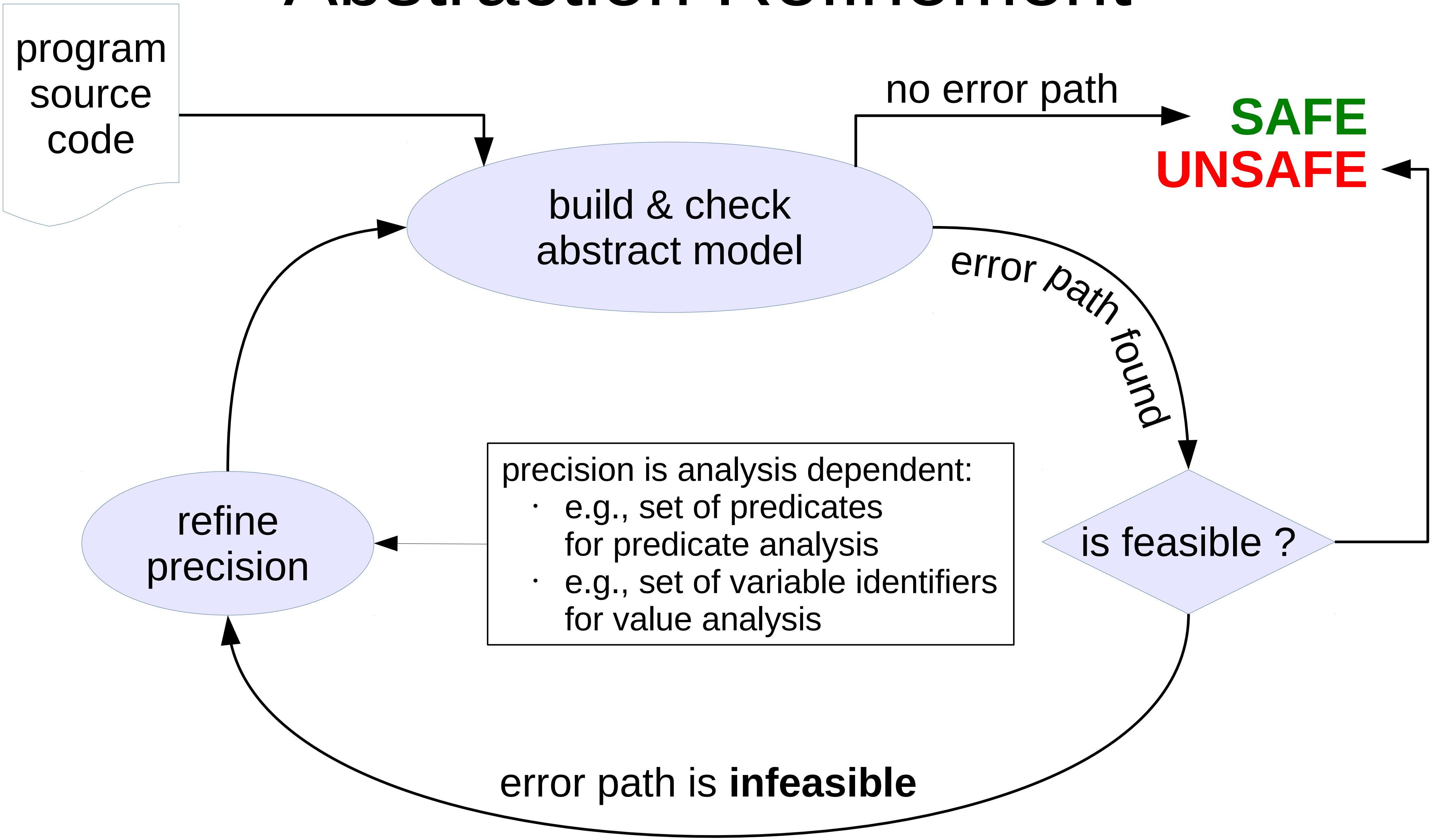
- Disregard irrelevant information
  - Avoids state-space explosion
  - Allows verification of real-world software to scale
  - Success story of **SLAM** project at Microsoft

- But how does a **good** abstraction look like?
  - Too coarse → False alarms
  - Too precise → More timeouts
  - Impossible to do manually → Automation needed

Counterexample-Guided Abstraction Refinement



Counterexample-Guided Abstraction Refinement



program source code

build & check abstract model

no error path

SAFE

UNSAFE

error path found

is feasible ?

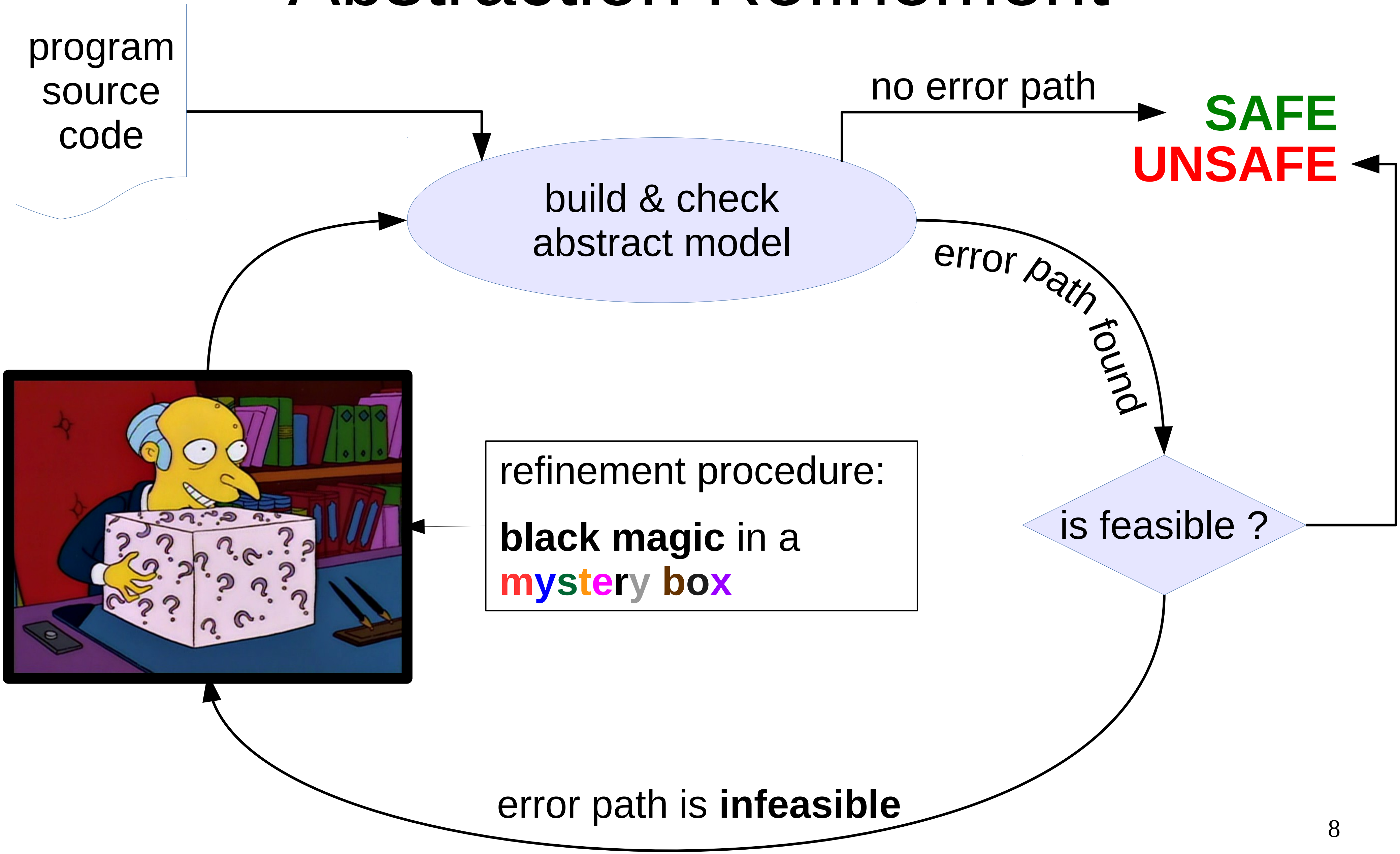
error path is **infeasible**

refine precision

precision is analysis dependent:

- e.g., set of predicates for predicate analysis
- e.g., set of variable identifiers for value analysis

Counterexample-Guided Abstraction Refinement



program source code

build & check abstract model

no error path

SAFE

UNSAFE

error path found

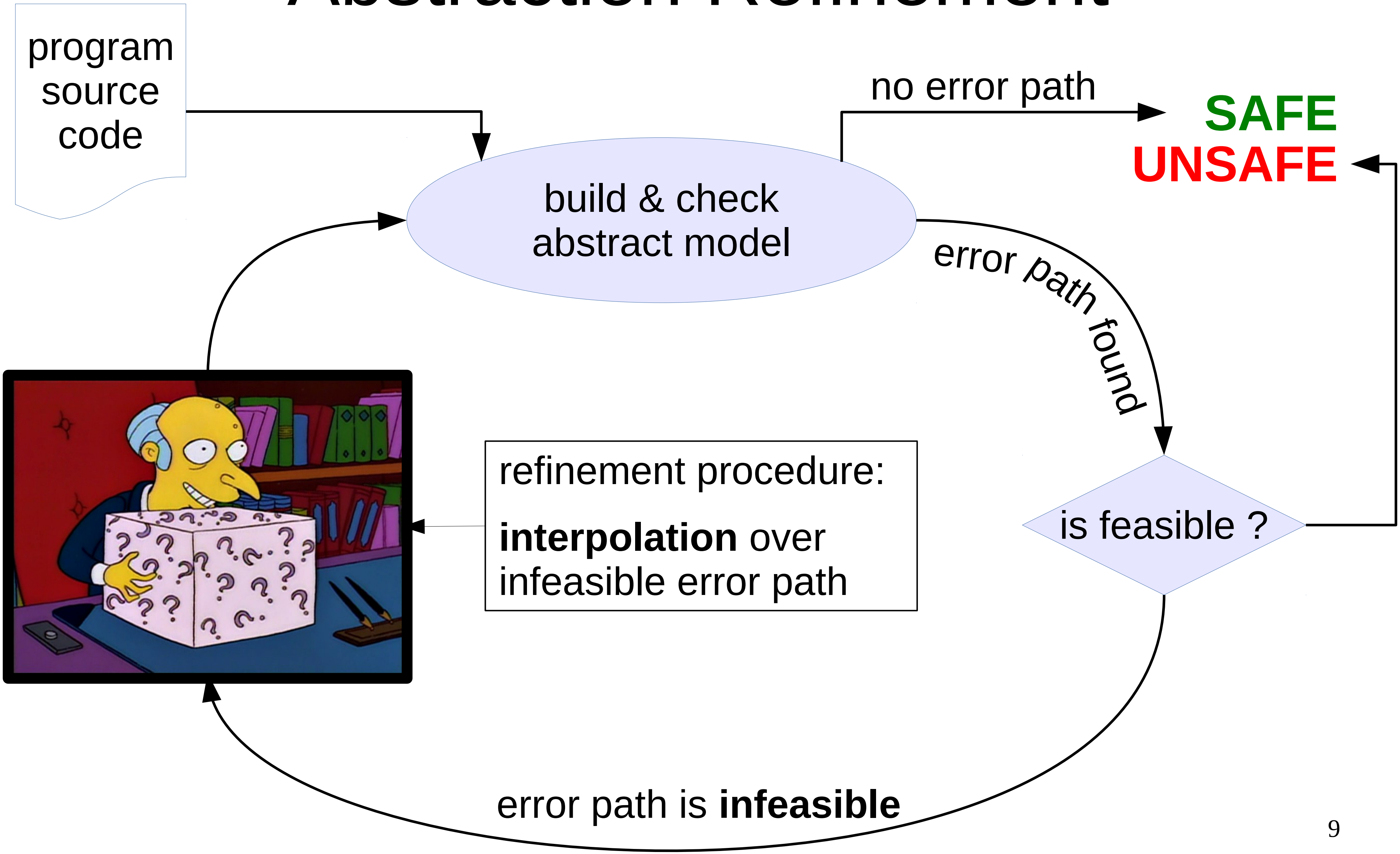
is feasible ?

refinement procedure:

black magic in a mystery box

error path is **infeasible**

Counterexample-Guided Abstraction Refinement



program source code

build & check abstract model

no error path

SAFE

UNSAFE

error path found

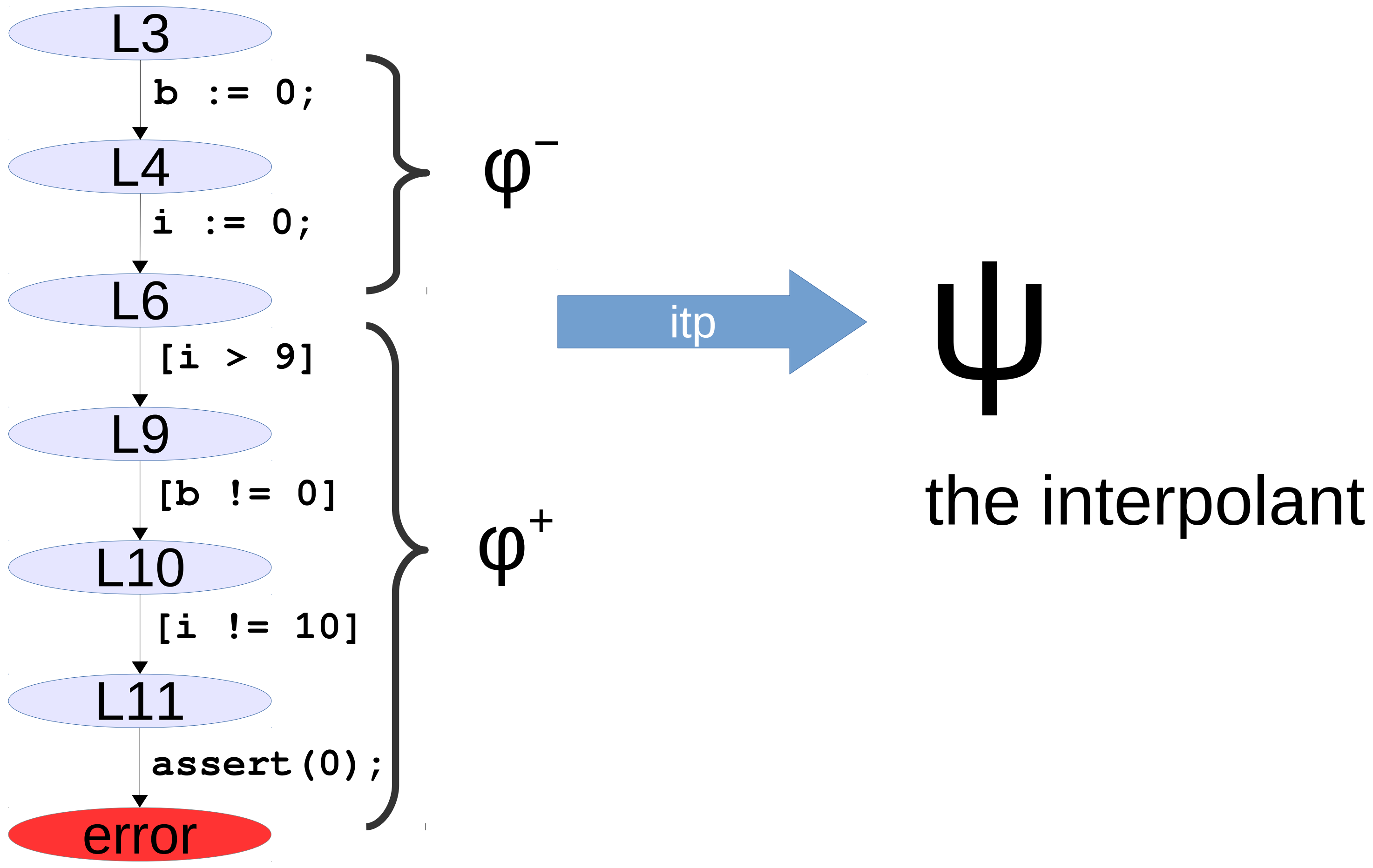
is feasible ?

refinement procedure:

interpolation over infeasible error path

error path is **infeasible**

Craig Interpolation



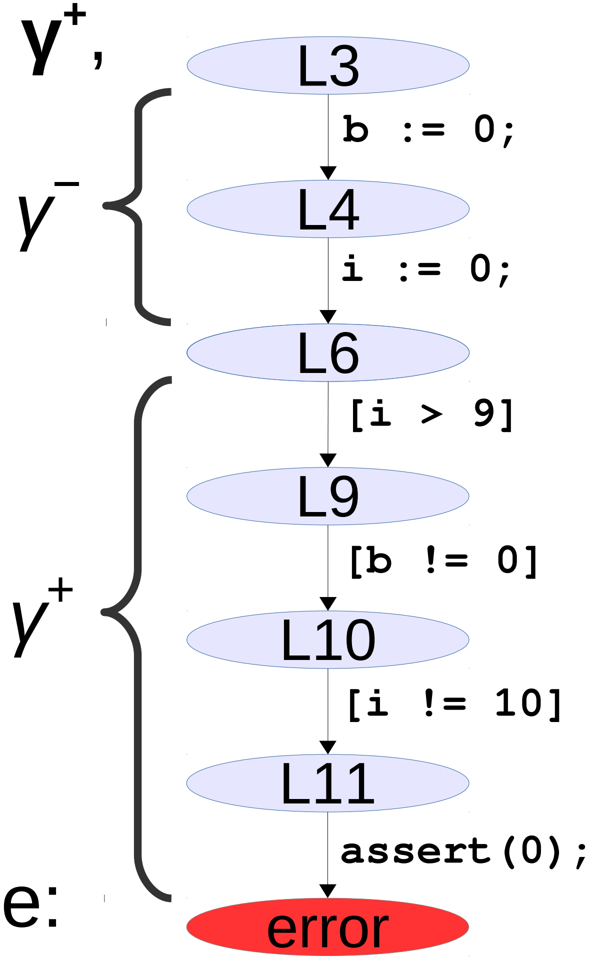
At L6 the interpolant ψ for φ^- and φ^+ could be:
$[b = 0]$, or $[i = 0]$, or $[b = 0 \land i = 0]$, or ...

[Abstractions from Proofs, 2004, Henzinger, Jhala, Majumdar, McMillan]

“Explicit-Value” Interpolation

For a pair of ***constraint sequences*** $\boldsymbol{\gamma^-}$ and $\boldsymbol{\gamma^+}$, such that $\boldsymbol{\gamma^-} \wedge \boldsymbol{\gamma^+}$ is ***contradicting***, ***interpolant ψ*** is a ***constraint sequence*** that fulfills the following requirements:

1) γ^- implies ψ
2) $ψ \wedge \gamma^+$ is unsatisfiable
3) ψ only contains symbols that are common to both γ^- and γ^+



```
b := 0;
i := 0;
[i > 9]
[b != 0]
[i != 10]
assert(0);
```

At L6 the interpolant ψ for γ^- and γ^+ could be:
[b = 0], or [i = 0], or [b = 0 ∧ i = 0], or ...

[Explicit-State Software Model Checking Based on CEGAR and Interpolation, 2013, D. Beyer, S. Löwe]

Interpolants

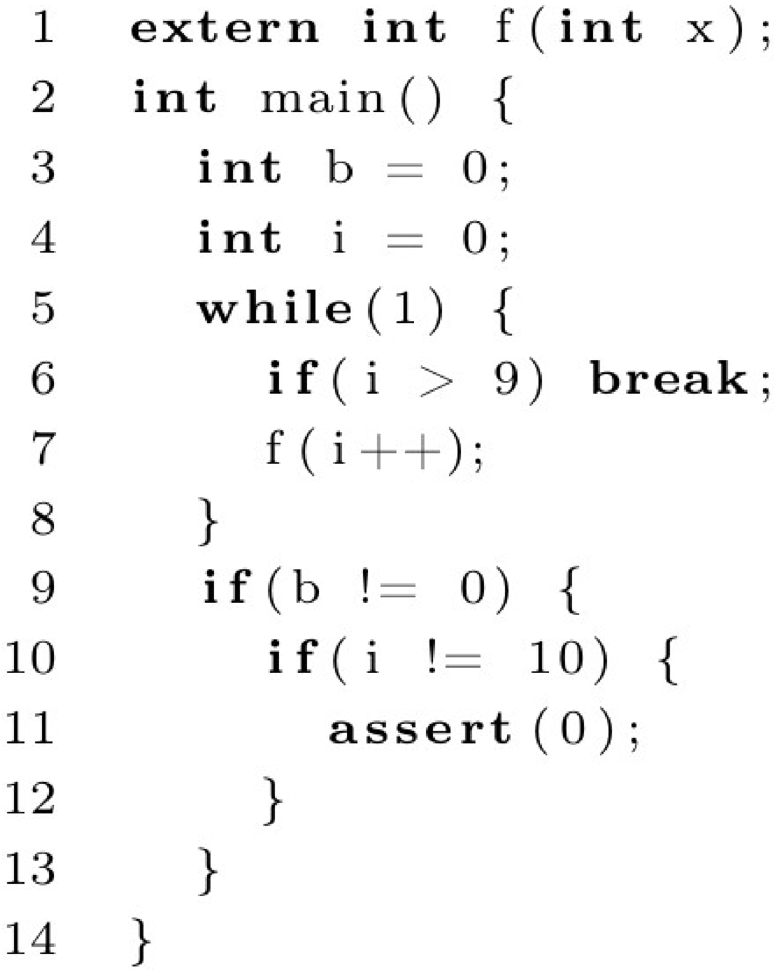
- Represent **concise** explanations for infeasibility of error
- ➢ Therefore, ideally suited for refinement of precision

Interpolation

- Represent **concise** explanations for infeasibility of error
- ~~Therefore, ideally suited for refinement of precision~~
- Theoretically, well suited for refinement of precision

Example – Good vs. Bad Interpolants

Input Program



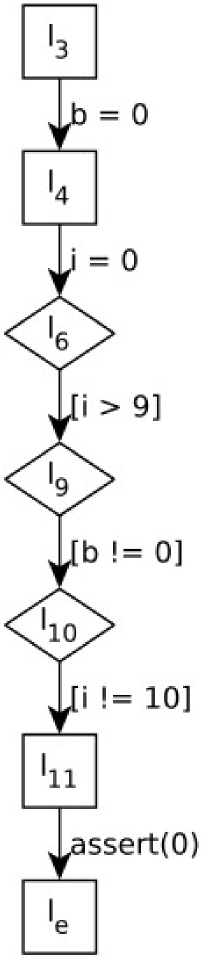
```
extern int f(int x);
int main() {
  int b = 0;
  int i = 0;
  while(1) {
    if(i > 9) break;
    f(i++);
  }
  if(b != 0) {
    if(i != 10) {
      assert(0);
    }
  }
}
```

Example – Good vs. Bad Interpolants

Input Program

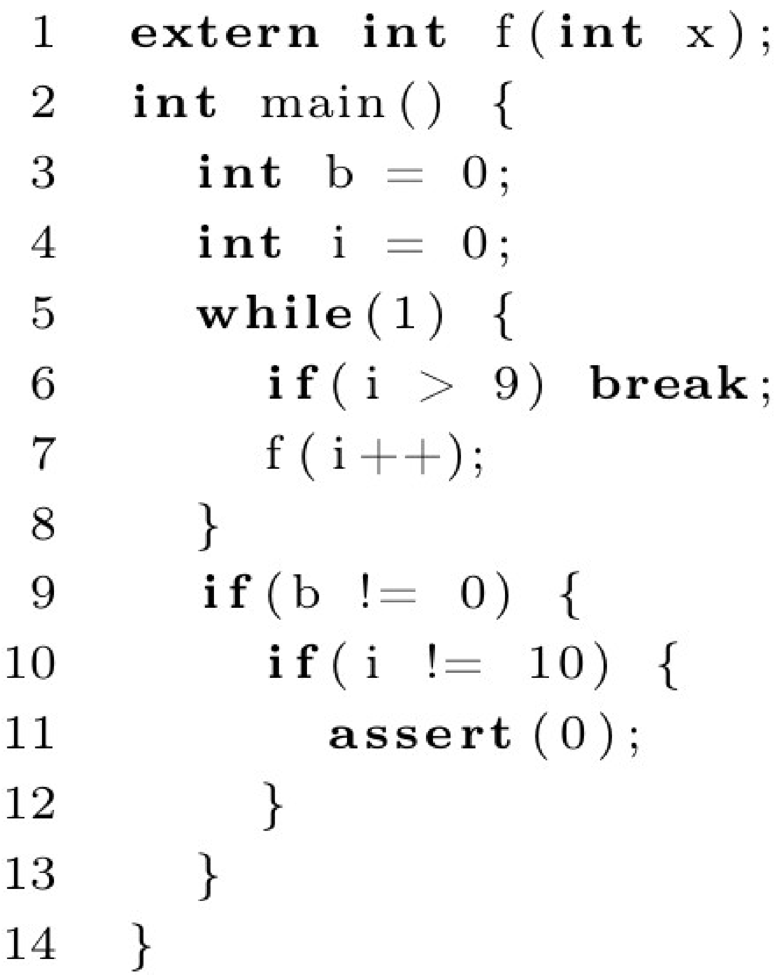
```
 1  extern int f(int x);
 2  int main() {
 3    int b = 0;
 4    int i = 0;
 5    while(1) {
 6      if(i > 9) break;
 7      f(i++);
 8    }
 9    if(b != 0) {
10      if(i != 10) {
11        assert(0);
12      }
13    }
14  }
```

Abstract Error Path



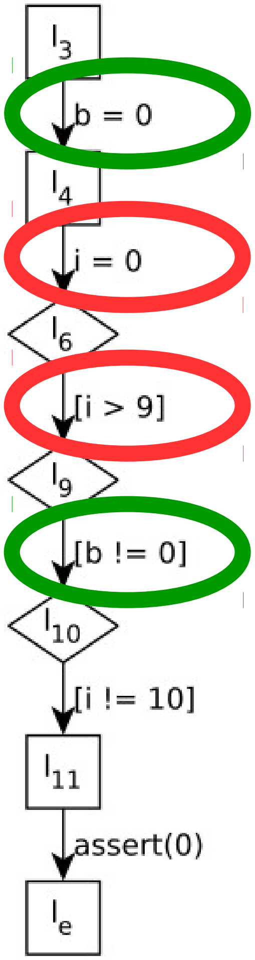
Example – Good vs. Bad Interpolants

Input Program



```
extern int f(int x);
int main() {
  int b = 0;
  int i = 0;
  while(1) {
    if(i > 9) break;
    f(i++);
  }
  if(b != 0) {
    if(i != 10) {
      assert(0);
    }
  }
}
```

Abstract Error Path

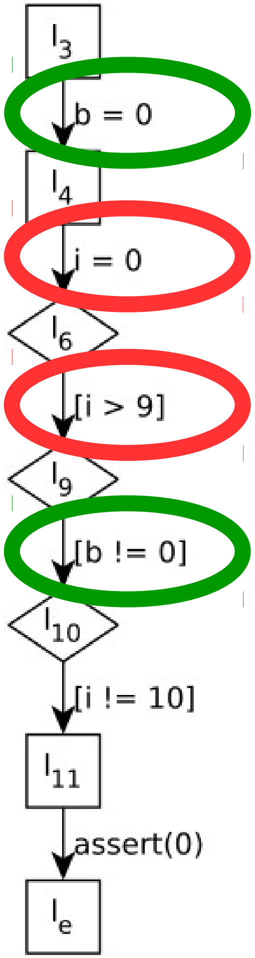


Example – Good vs. Bad Interpolants

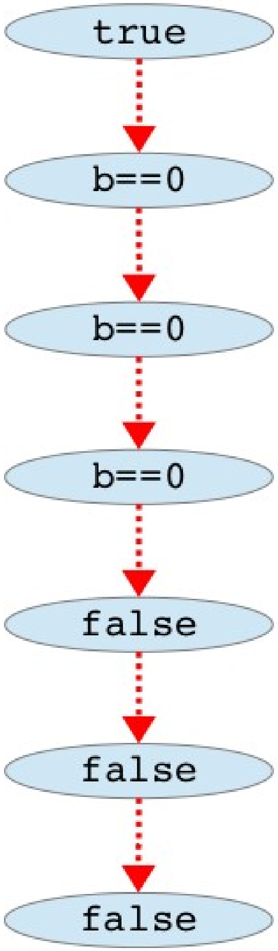
Input Program

```
extern int f(int x);
int main() {
   int b = 0;
   int i = 0;
   while(1) {
      if(i > 9) break;
      f(i++);
   }
   if(b != 0) {
      if(i != 10) {
         assert(0);
      }
   }
}
```

Abstract Error Path



Good Interpolants



true

b==0

b==0

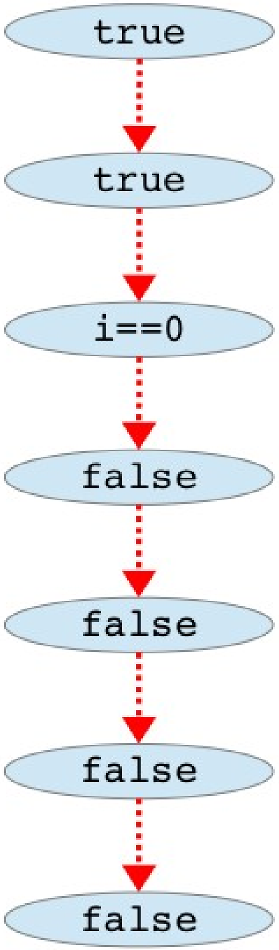
b==0

false

false

false

Bad Interpolants



true

true

i==0

false

false

false

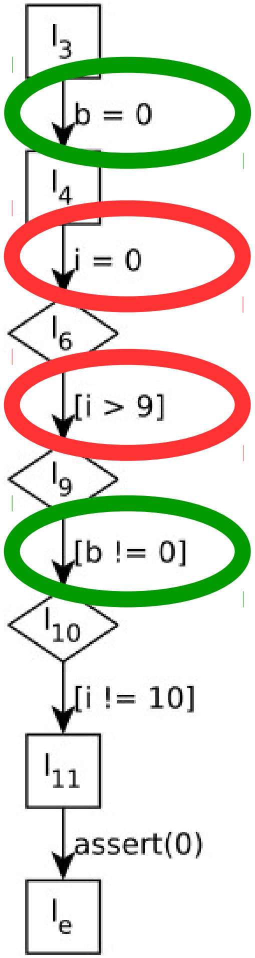
false

Example – Good vs. Bad Interpolants

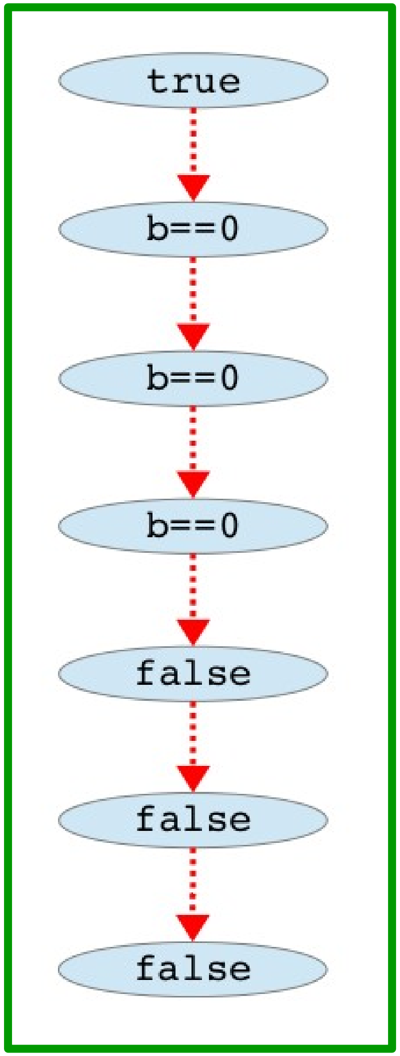
Input Program

```
extern int f(int x);
int main() {
  int b = 0;
  int i = 0;
  while(1) {
    if(i > 9) break;
    f(i++);
  }
  if(b != 0) {
    if(i != 10) {
      assert(0);
    }
  }
}
```

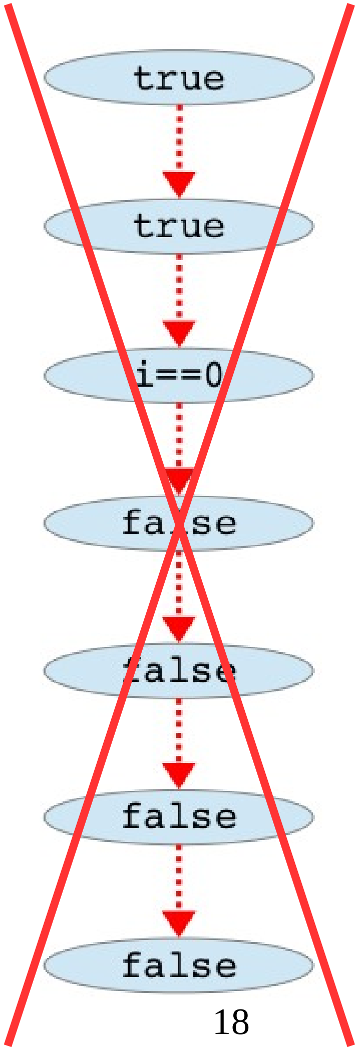
Abstract Error Path



Good Interpolants



Bad Interpolants



Interpolation – Recap

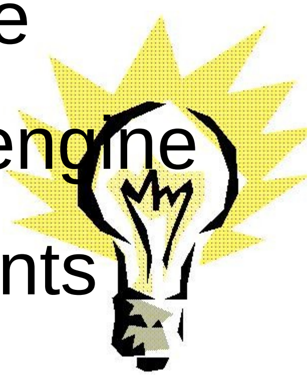
- Represent **concise** explanations for infeasibility of error
- ➢ ~~Therefore, ideally suited for refinement of precision~~
- ➢ Theoretically, well suited for refinement of precision

- Single interpolation problem typically has several solutions
  - Which interpolant do we get?
  - Some are “good”, others might lead to divergence
  - Selection controlled by internals of interpolation engine

Interpolation – Guided

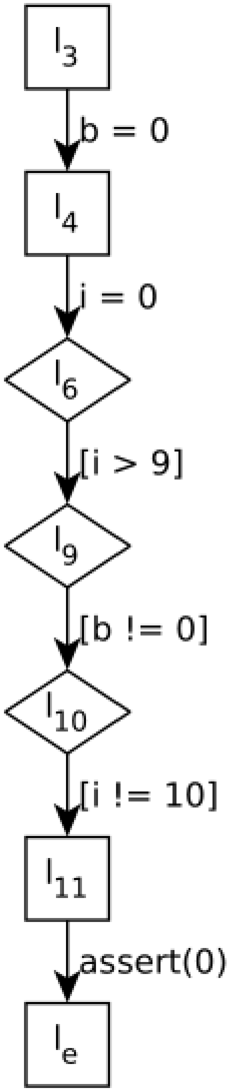
- Represent **concise** explanations for infeasibility of error
- ➢ ~~Therefore, ideally suited for refinement of precision~~
- ➢ Theoretically, well suited for refinement of precision

- Single interpolation problem typically has several solutions
  - Which interpolant do we get?
  - Some are “good”, others might lead to divergence
  - Selection controlled by internals of interpolation engine
- ➢ Guide interpolation, ideally towards good interpolants



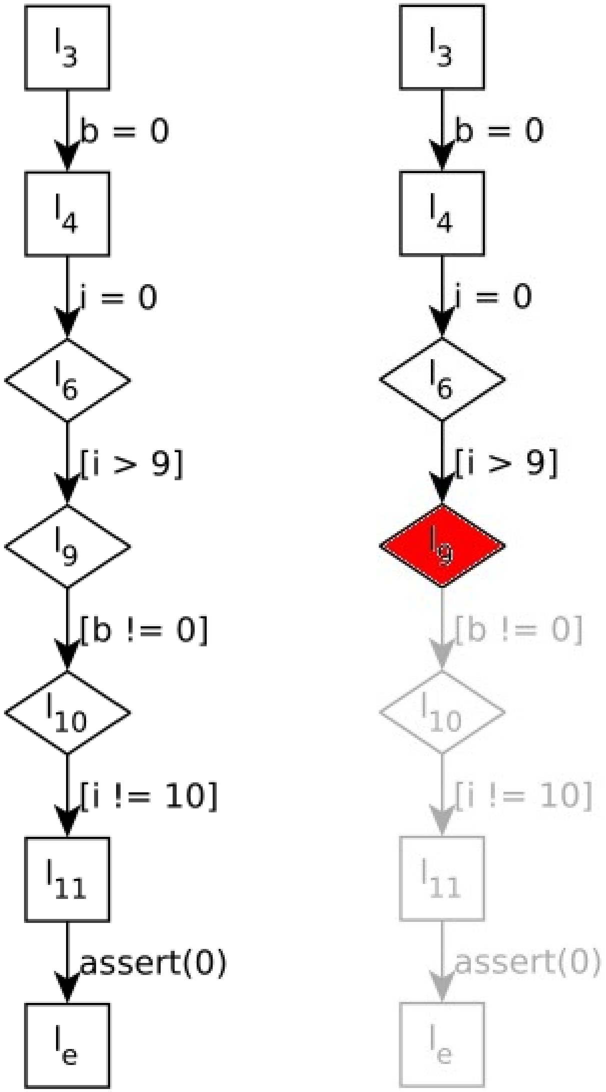
Extraction of Infeasible Sliced Prefixes

Abstract Error Path



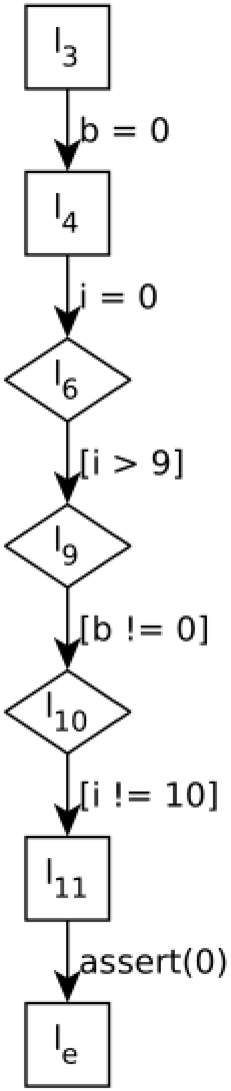
Extraction of Infeasible Sliced Prefixes

Abstract Error Path

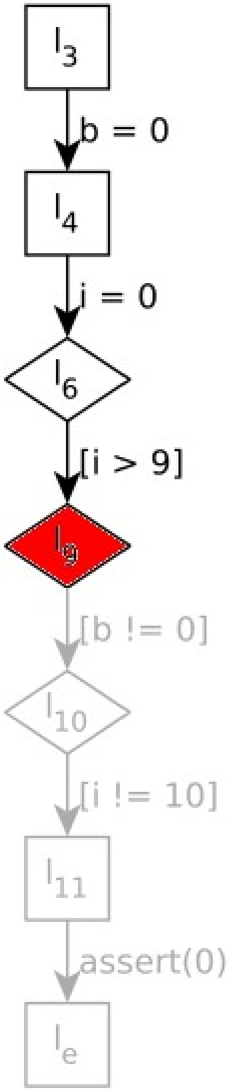


Extraction of Infeasible Sliced Prefixes

Abstract Error Path

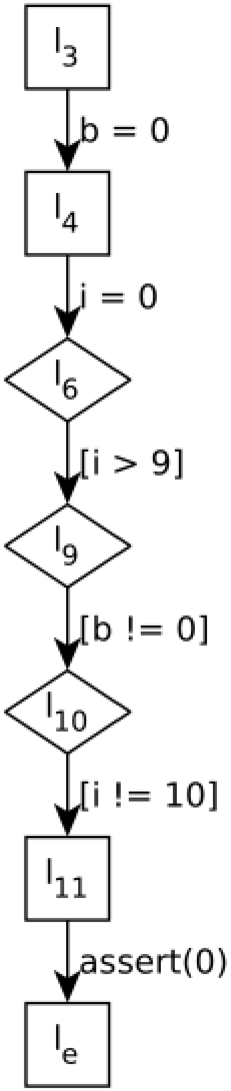


1st Prefix

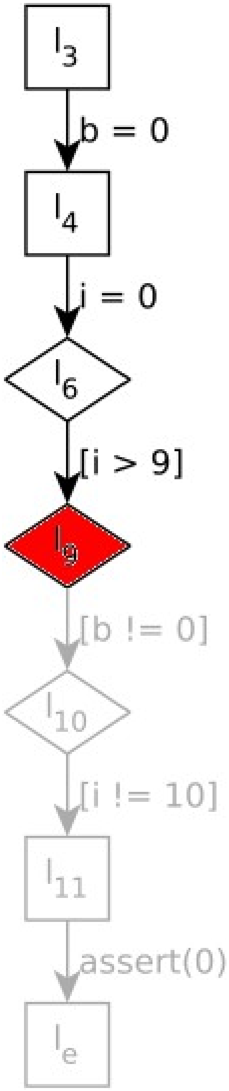


Extraction of Infeasible Sliced Prefixes

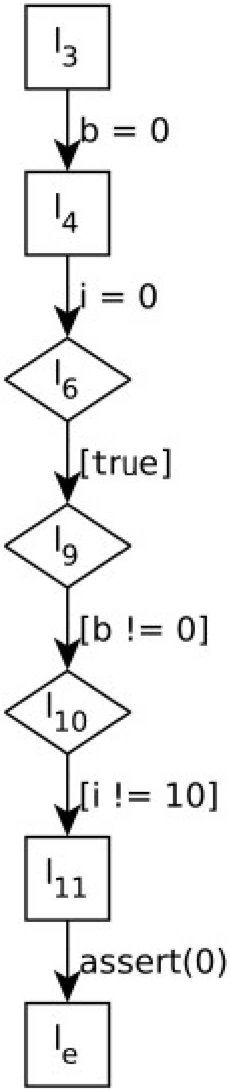
Abstract Error Path



1^{st} Prefix

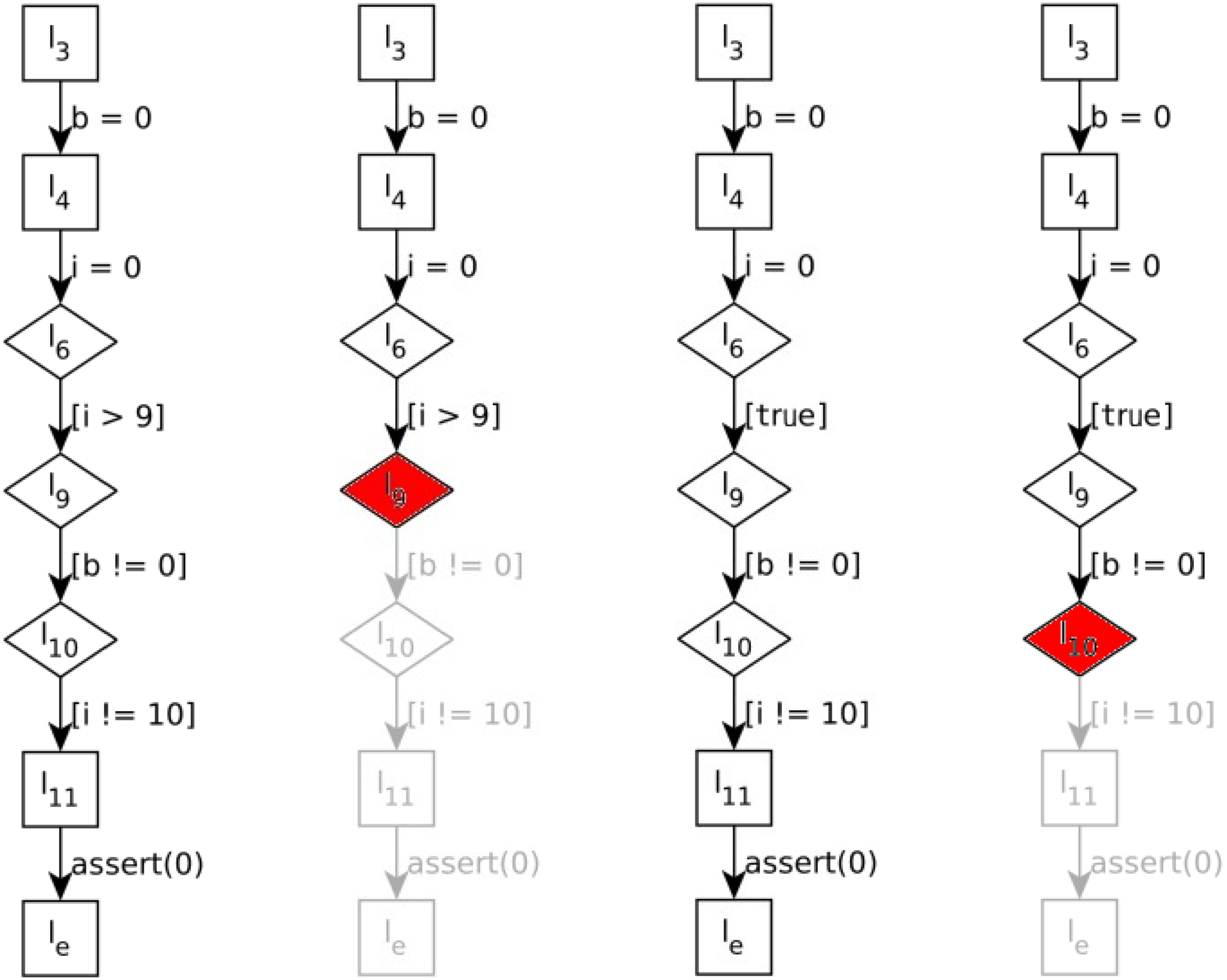


Sliced Error Path



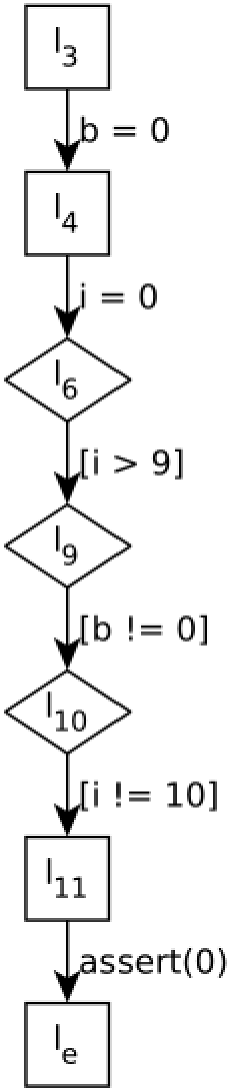
Extraction of Infeasible Sliced Prefixes

Abstract Error Path 1st Prefix Sliced Error Path

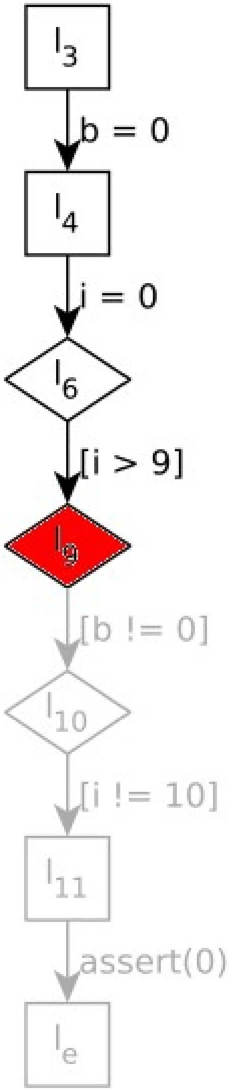


Extraction of Infeasible Sliced Prefixes

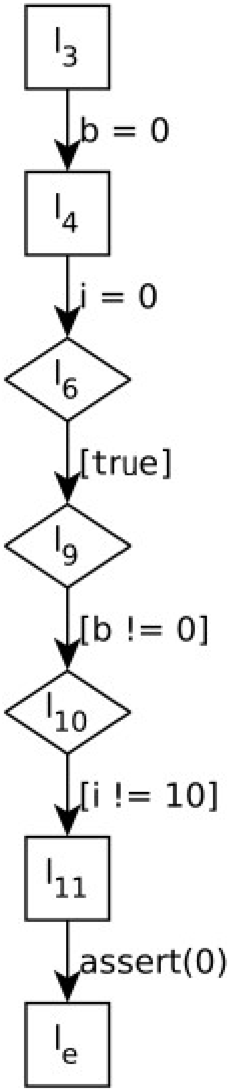
Abstract Error Path



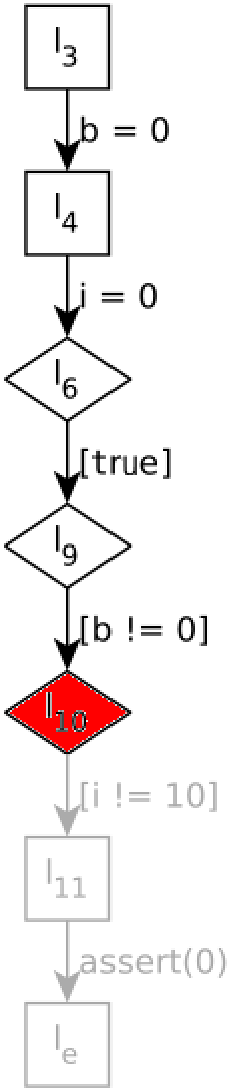
1st Prefix



Sliced Error Path



2nd Prefix

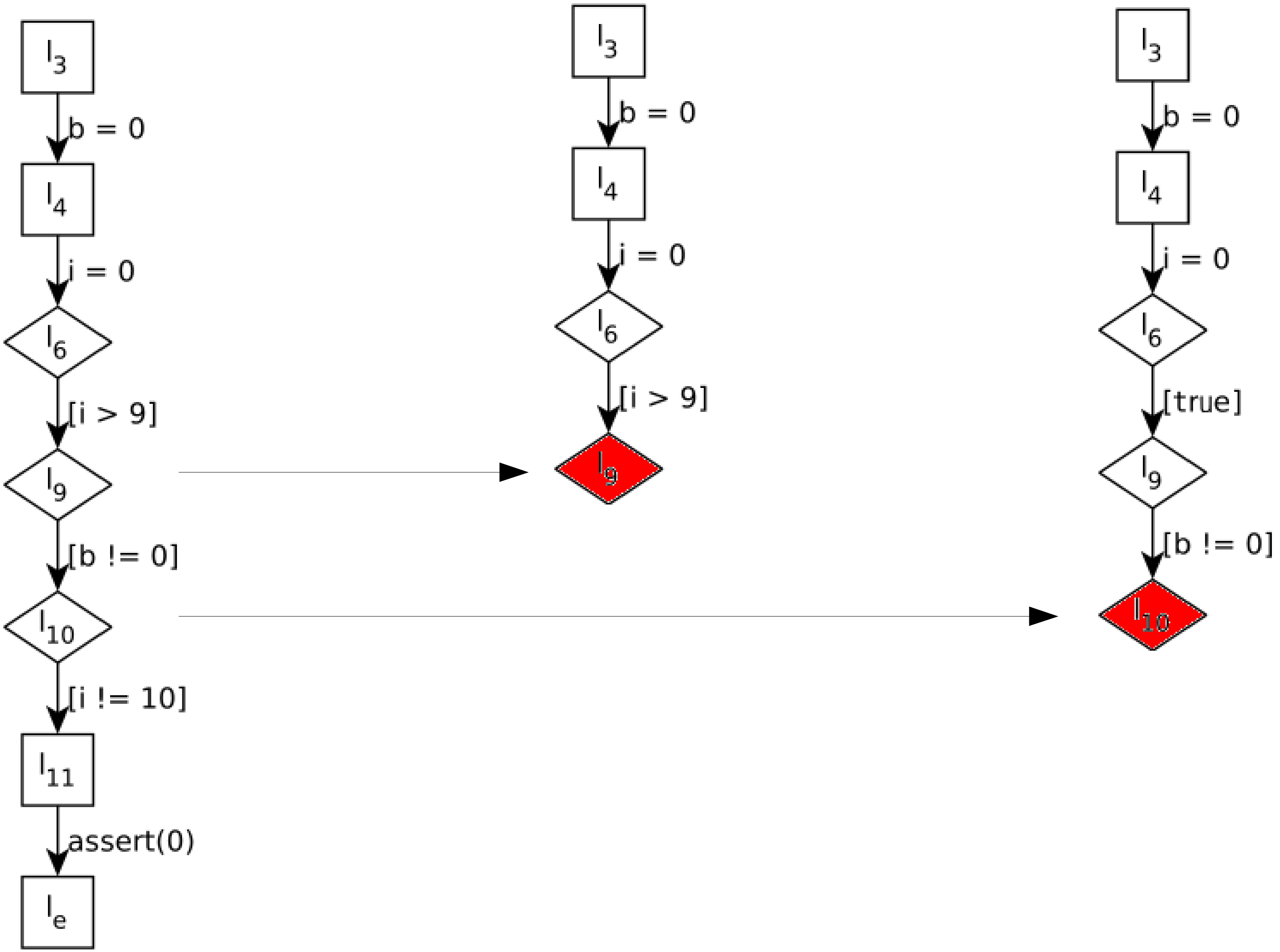


Extraction of Infeasible Sliced Prefixes

Single Abstract Error Path

1st Prefix

2nd Prefix

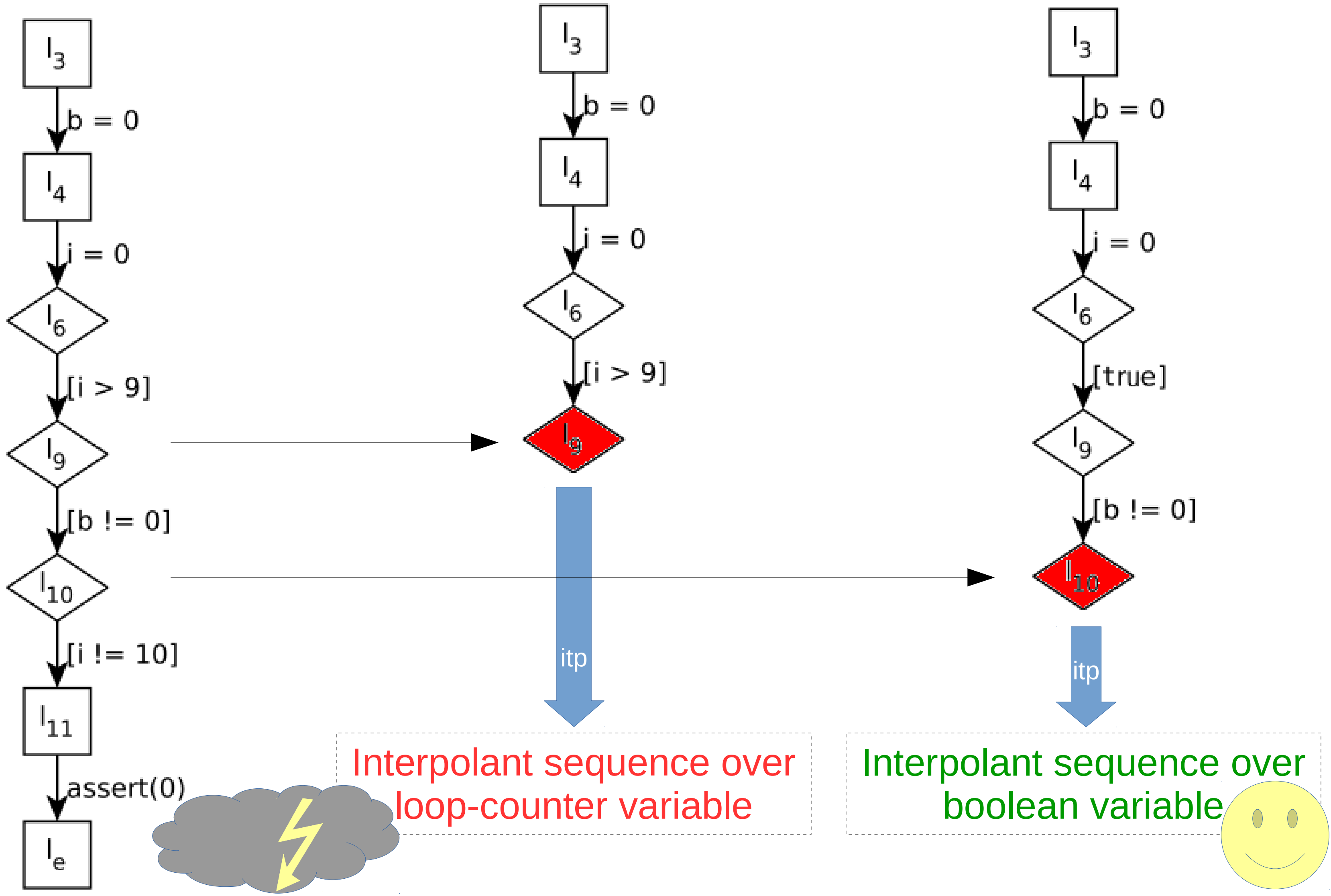


Extraction of Infeasible Sliced Prefixes

Single Abstract Error Path

1st Prefix

2nd Prefix



Interpolant sequence over loop-counter variable

Interpolant sequence over boolean variable

Proposition: Interpolants of Prefixes

Any **infeasible sliced prefix φ**

that is extracted from an **infeasible error path σ**

can be used **for interpolation**

to **exclude the original error path σ**

from **subsequent iterations** of CEGAR loop

Proposition: Interpolants of Prefixes

Any **infeasible sliced prefix φ**

that is extracted from an **infeasible error path σ**

can be used **for interpolation**

to **exclude the original error path σ**

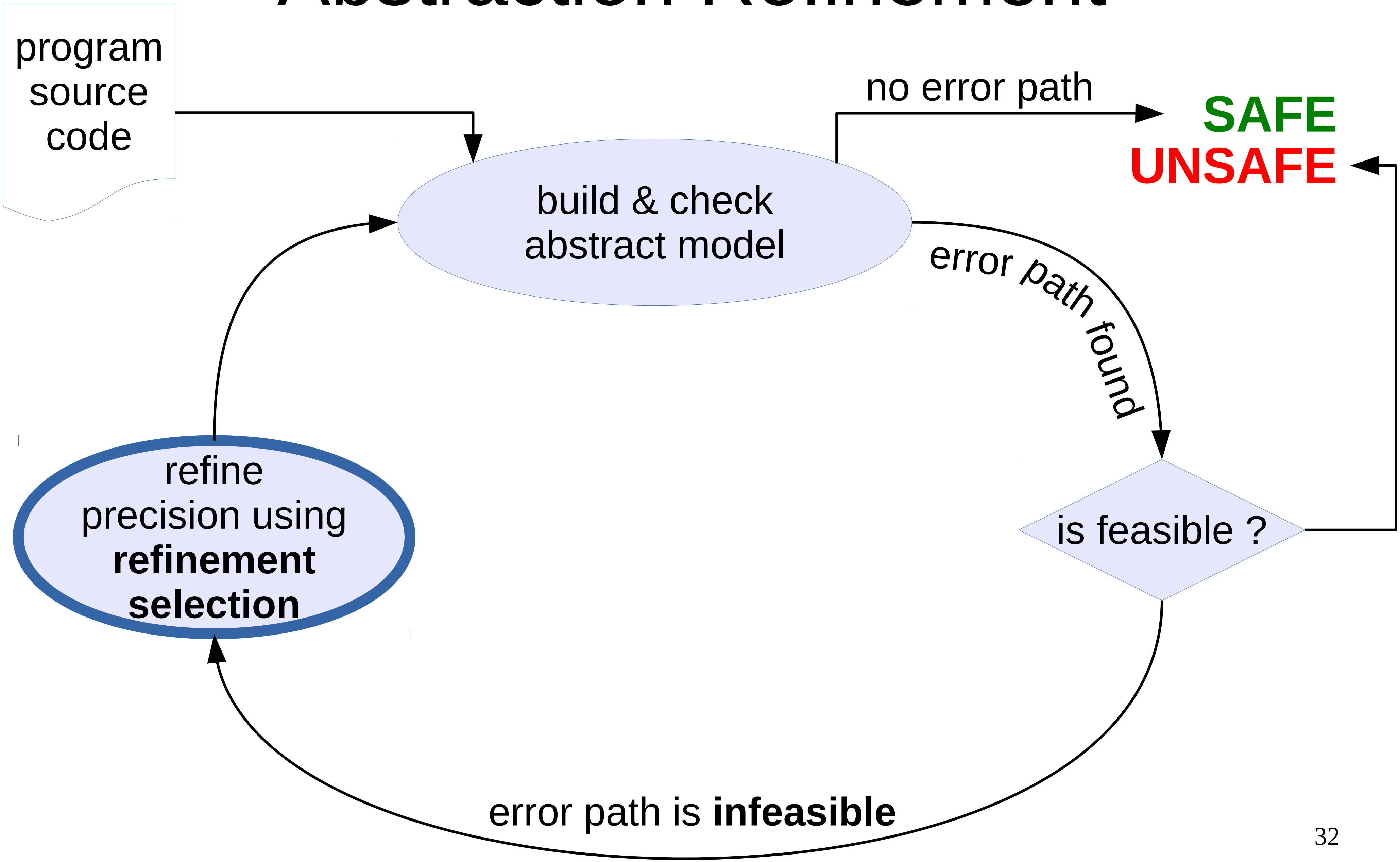
from **subsequent iterations** of CEGAR loop

- **We can use any prefix we want for interpolation !**

refinement selection is now possible

Proof is in the Paper

Couterexample-Guided Abstraction Refinement


program source code
no error path
SAFE
UNSAFE
build & check abstract model
error path found
refine precision using **refinement selection**
is feasible ?
error path is **infeasible**

Refinement Selection

- Refinement is now an optimization problem
- Select any refinement from a set of refinements
- Different heuristics for selecting an refinement
  - Shortest or longest prefix
  - Best or worst score based on Domain-Types
  - Width of precision
  - Depth of refinement root
  - ...

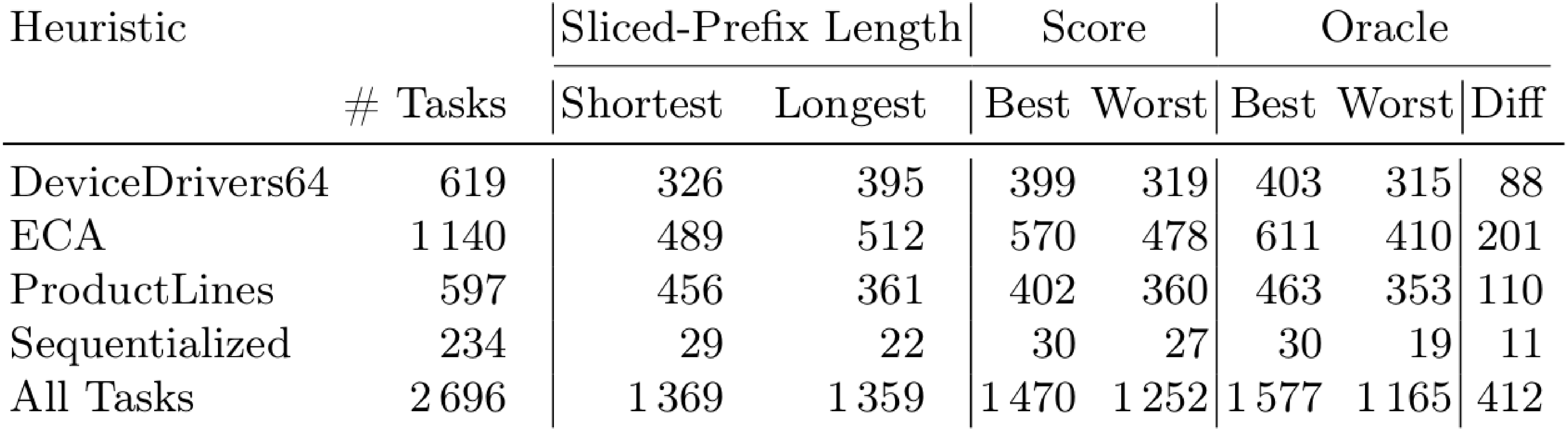
Implementation & Experiments

- Evaluated on over 2500 benchmarks from SV-COMP'15
- Evaluated under rules of SV-COMP'15
  - 15 minutes CPU Time
  - 15 GB RAM
- All ideas and concepts described are implemented
  - Integrated into CPAchecker
  - Extended Value Analysis

CPA✓

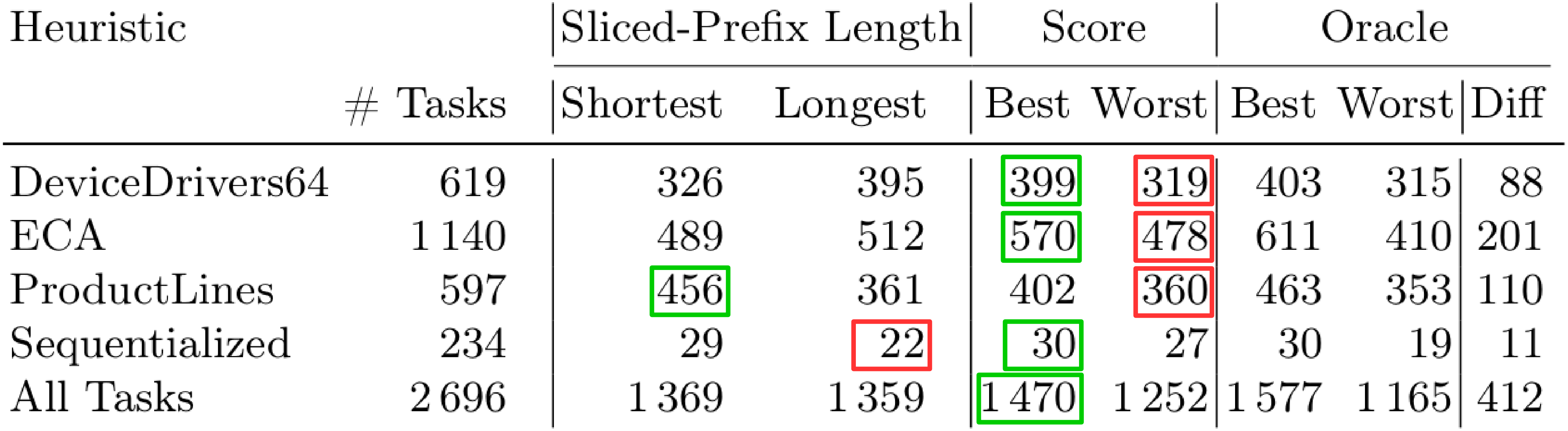
http://cpachecker.sosy-lab.org

Results



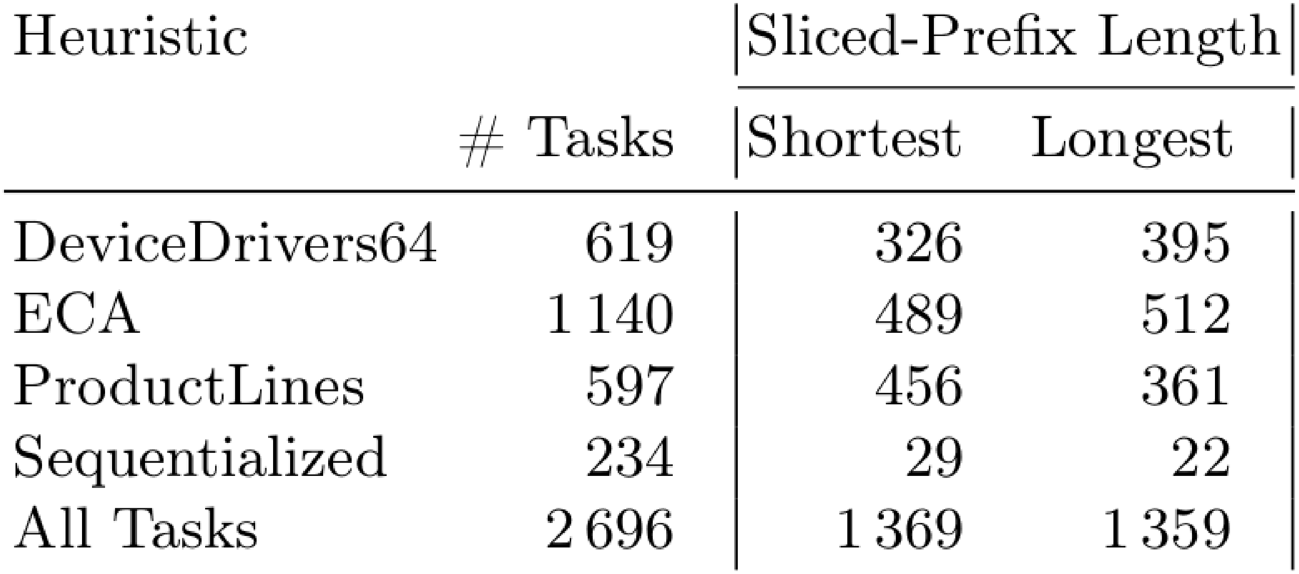
Heuristic		Sliced-Prefix Length		Score		Oracle		
	# Tasks	Shortest	Longest	Best	Worst	Best	Worst	Diff
DeviceDrivers64	619	326	395	399	319	403	315	88
ECA	1 140	489	512	570	478	611	410	201
ProductLines	597	456	361	402	360	463	353	110
Sequentialized	234	29	22	30	27	30	19	11
All Tasks	2 696	1 369	1 359	1 470	1 252	1 577	1 165	412

Results

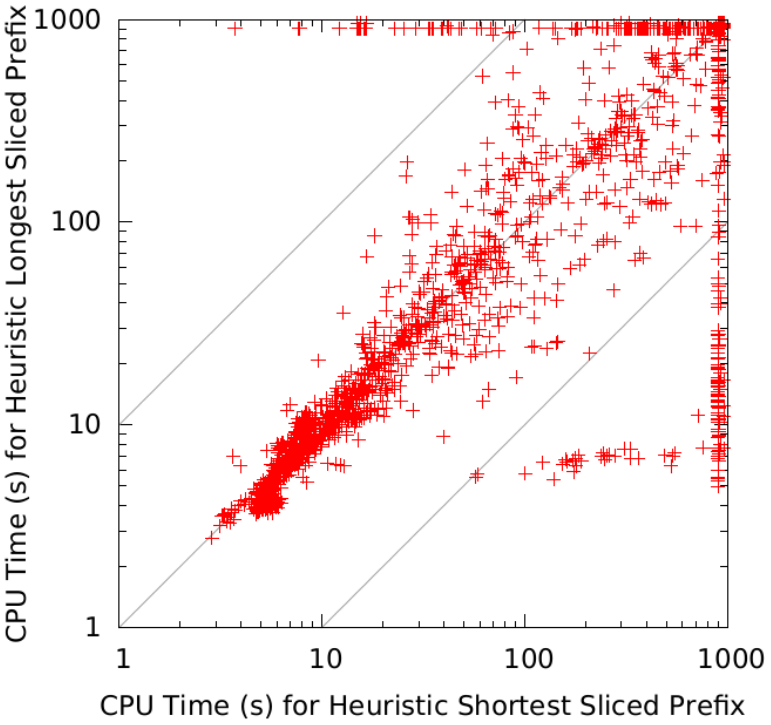


Heuristic		Sliced-Prefix Length		Score		Oracle		
	# Tasks	Shortest	Longest	Best	Worst	Best	Worst	Diff
DeviceDrivers64	619	326	395	399	319	403	315	88
ECA	1 140	489	512	570	478	611	410	201
ProductLines	597	456	361	402	360	463	353	110
Sequentialized	234	29	22	30	27	30	19	11
All Tasks	2 696	1 369	1 359	1 470	1 252	1 577	1 165	412

Results – Sliced-Prefix Length



Heuristic		Sliced-Prefix Length	
	# Tasks	Shortest	Longest
DeviceDrivers64	619	326	395
ECA	1 140	489	512
ProductLines	597	456	361
Sequentialized	234	29	22
All Tasks	2 696	1 369	1 359

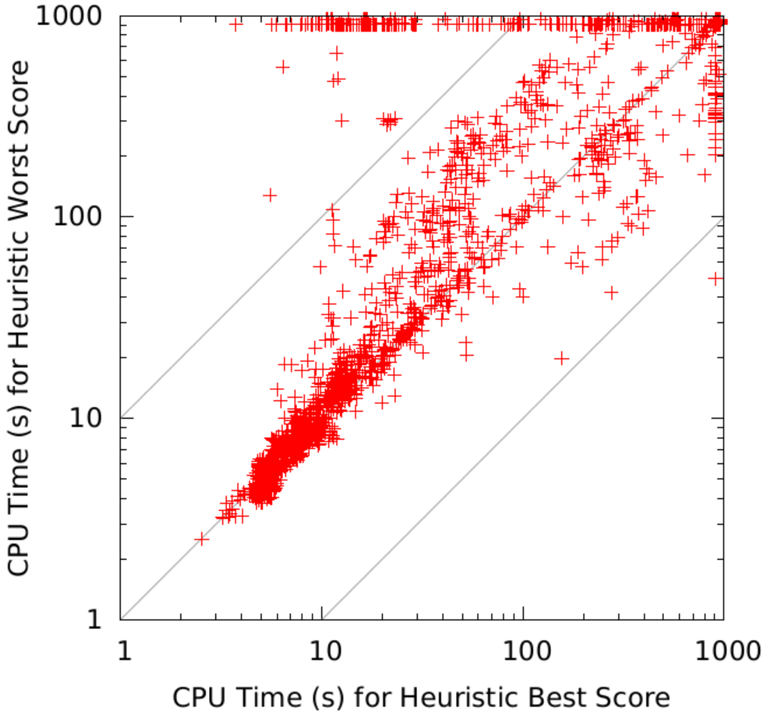


Heuristic “Shortest” vs. “Longest”

Powered by BenchExec - https://github.com/dbeyer/benchexec

Results – Domain-Type Score

Heuristic		Score	
	# Tasks	Best	Worst
DeviceDrivers64	619	399	319
ECA	1 140	570	478
ProductLines	597	402	360
Sequentialized	234	30	27
All Tasks	2 696	1 470	1 252



Heuristic “Best Score” vs. “Worst Score”

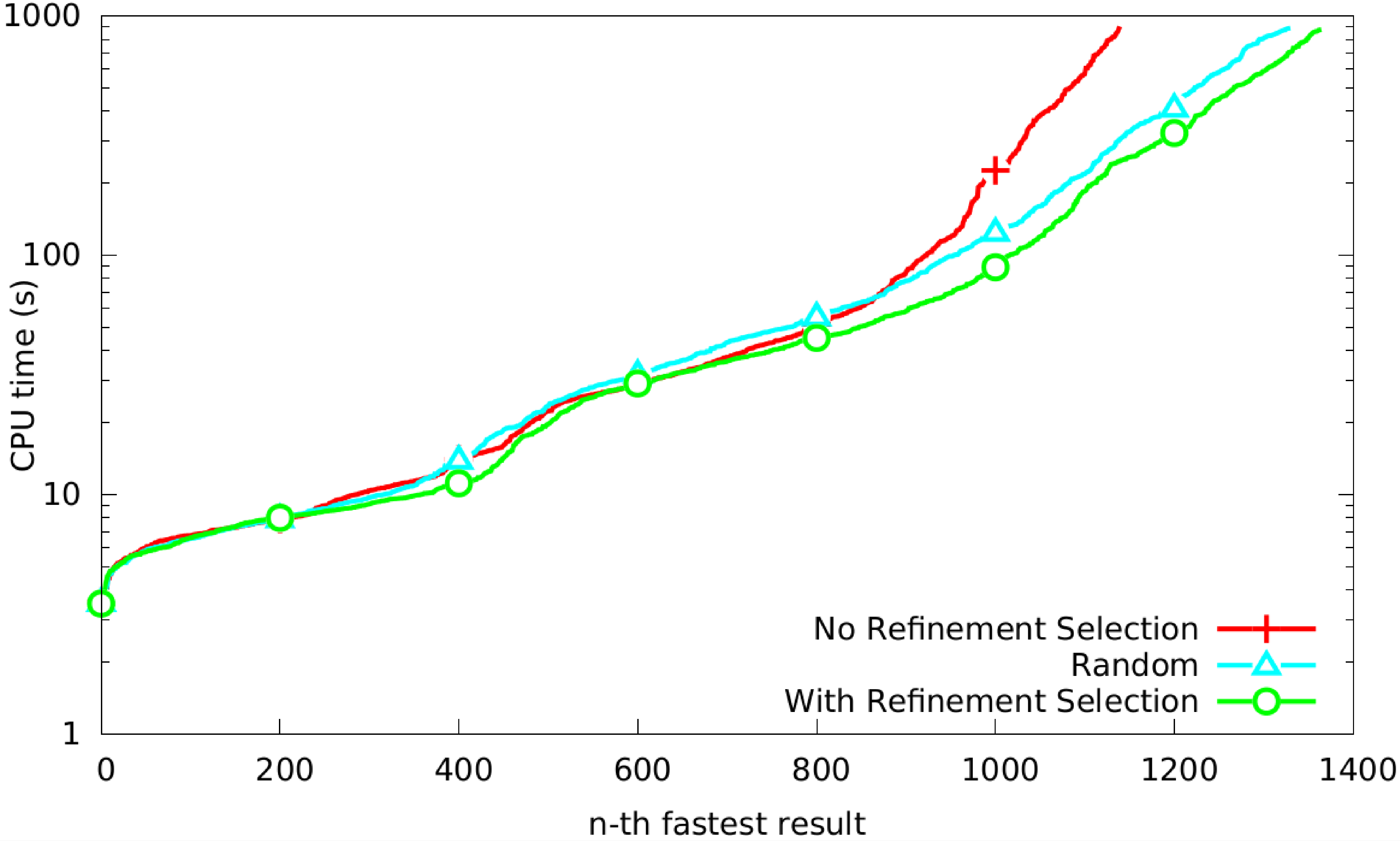
Powered by BenchExec - https://github.com/dbeyer/benchexec

We want Refinement Selection !!!

Because straight-forward interpolation completely and utterly sucks !!!11!

Stefan Löwe

Now also for Predicate Abstraction!



Conclusion

- Defined and implemented for two domains
  - infeasible sliced prefixes
  - precision selection heuristics
- Enables refinement selection

CPA✓

http://cpachecker.sosy-lab.org

- Nice Results
  - Refinement selection matters!
  - More research needed on how to select refinements