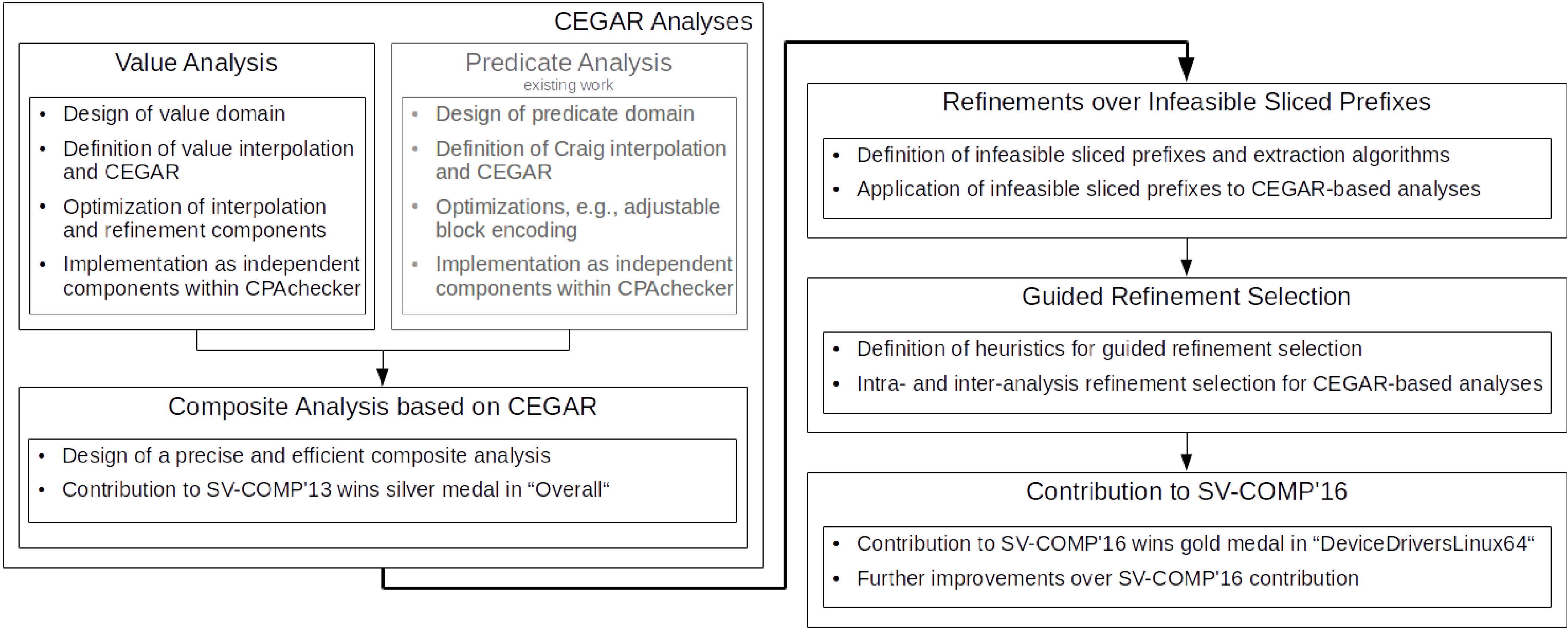
Effective Approaches to Abstraction Refinement for an Explicit Value Analysis

Stefan Löwe

SoSy-Lab
Software Systems

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Outline of my Thesis



CEGAR Analyses

Value Analysis

- Design of value domain
- Definition of value interpolation and CEGAR
- Optimization of interpolation and refinement components
- Implementation as independent components within CPAchecker

Predicate Analysis

existing work

- Design of predicate domain
- Definition of Craig interpolation and CEGAR
- Optimizations, e.g., adjustable block encoding
- Implementation as independent components within CPAchecker

Composite Analysis based on CEGAR

- Design of a precise and efficient composite analysis
- Contribution to SV-COMP'13 wins silver medal in “Overall“

Refinements over Infeasible Sliced Prefixes

- Definition of infeasible sliced prefixes and extraction algorithms
- Application of infeasible sliced prefixes to CEGAR-based analyses

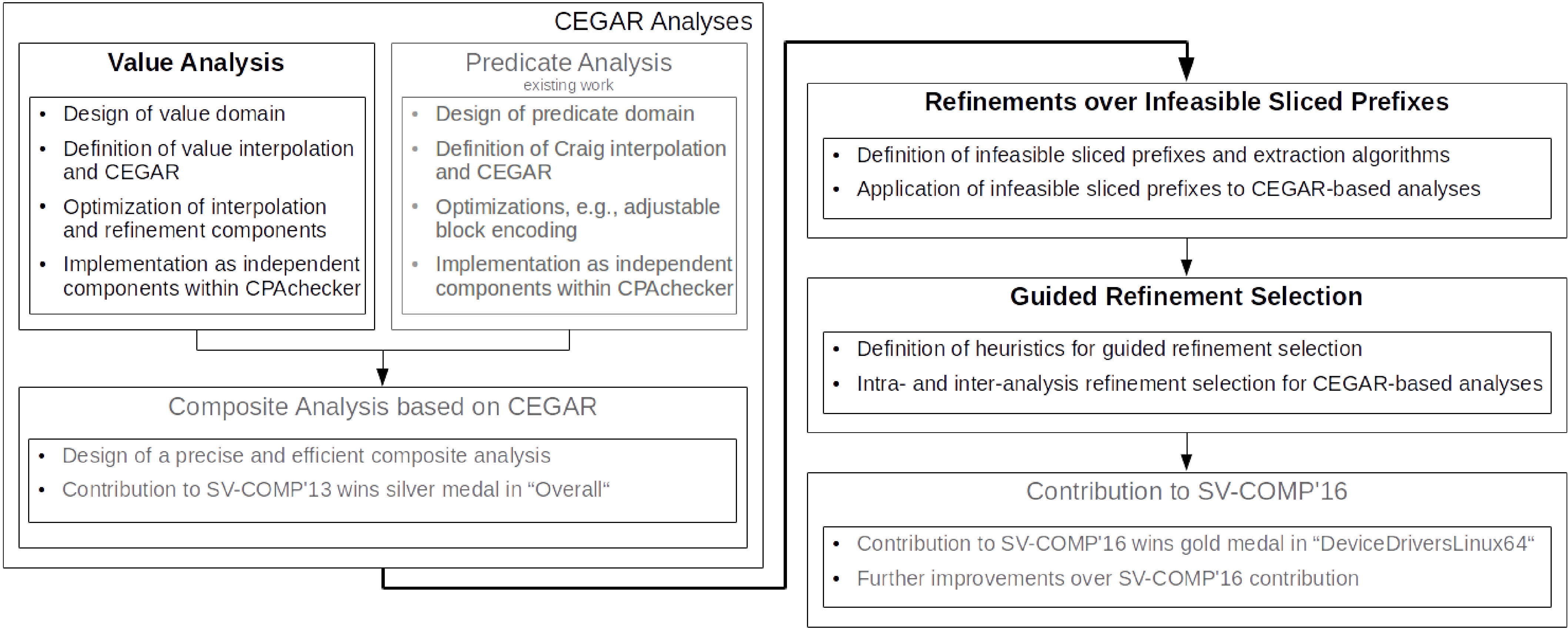
Guided Refinement Selection

- Definition of heuristics for guided refinement selection
- Intra- and inter-analysis refinement selection for CEGAR-based analyses

Contribution to SV-COMP'16

- Contribution to SV-COMP'16 wins gold medal in “DeviceDriversLinux64“
- Further improvements over SV-COMP'16 contribution

Outline of my Talk



CEGAR Analyses

Value Analysis

- Design of value domain
- Definition of value interpolation and CEGAR
- Optimization of interpolation and refinement components
- Implementation as independent components within CPAchecker

Predicate Analysis
existing work

- Design of predicate domain
- Definition of Craig interpolation and CEGAR
- Optimizations, e.g., adjustable block encoding
- Implementation as independent components within CPAchecker

Composite Analysis based on CEGAR

- Design of a precise and efficient composite analysis
- Contribution to SV-COMP'13 wins silver medal in “Overall“

Refinements over Infeasible Sliced Prefixes

- Definition of infeasible sliced prefixes and extraction algorithms
- Application of infeasible sliced prefixes to CEGAR-based analyses

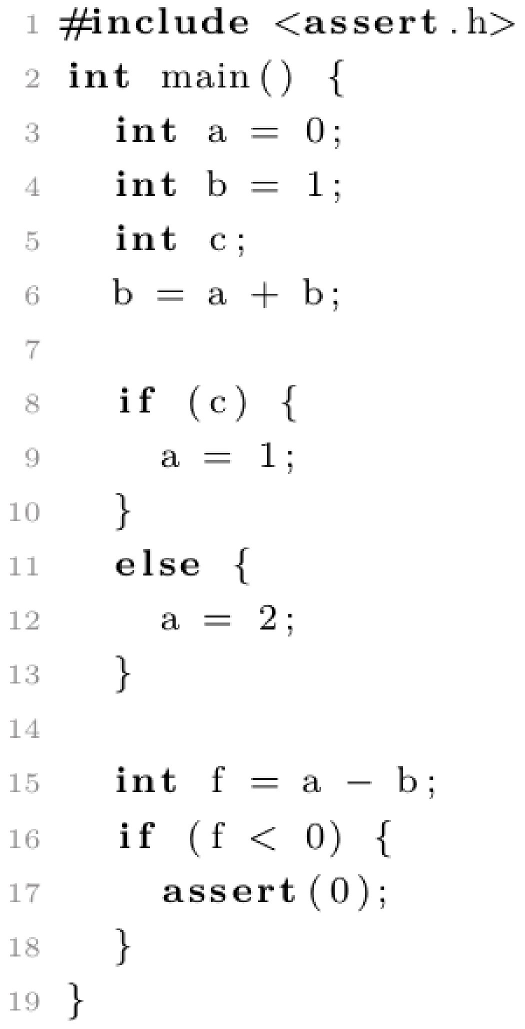
Guided Refinement Selection

- Definition of heuristics for guided refinement selection
- Intra- and inter-analysis refinement selection for CEGAR-based analyses

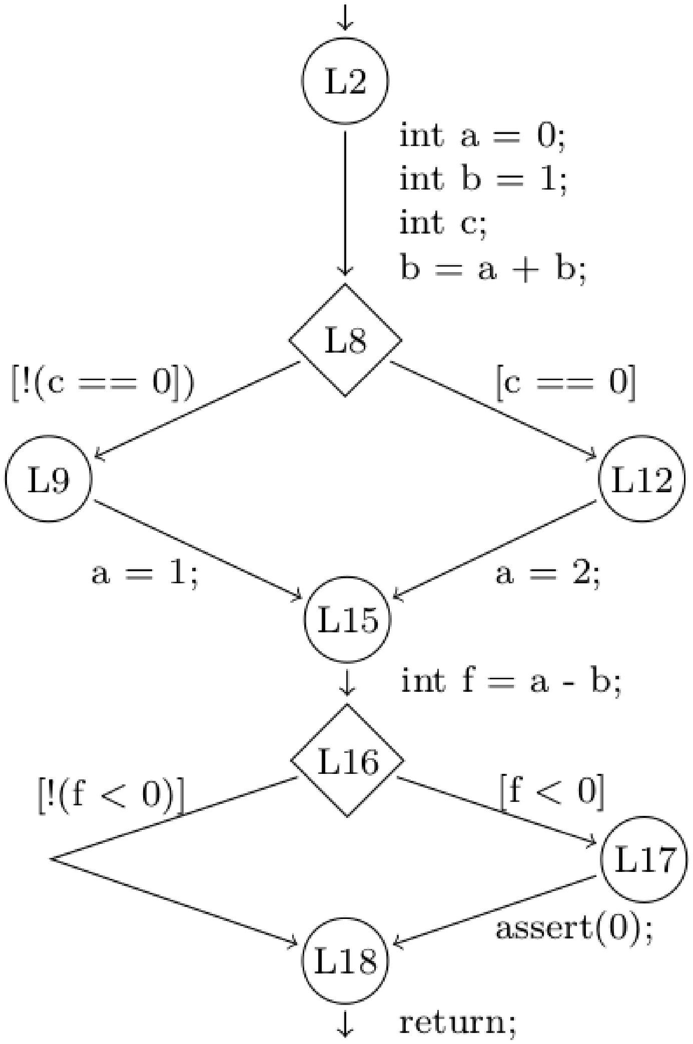
Contribution to SV-COMP'16

- Contribution to SV-COMP'16 wins gold medal in “DeviceDriversLinux64“
- Further improvements over SV-COMP'16 contribution

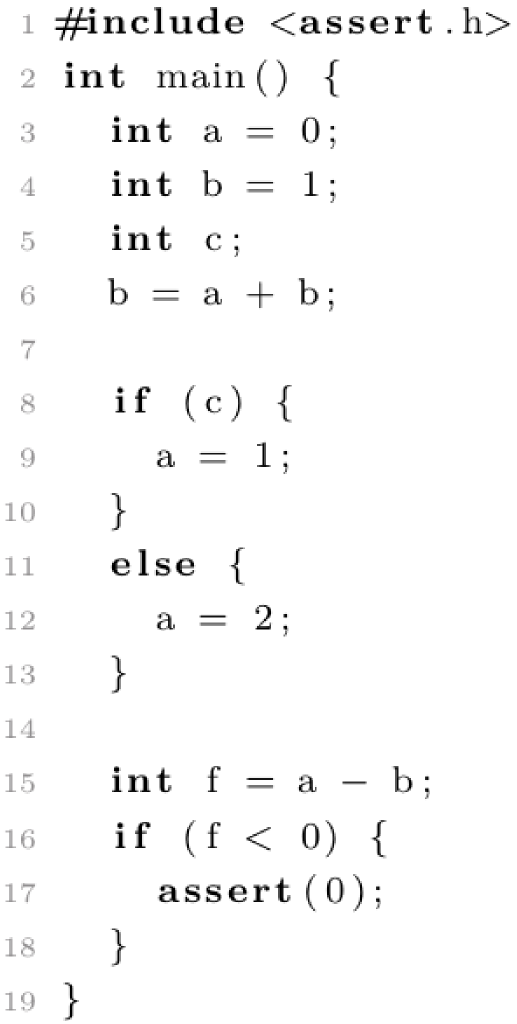
Value Analysis by Example



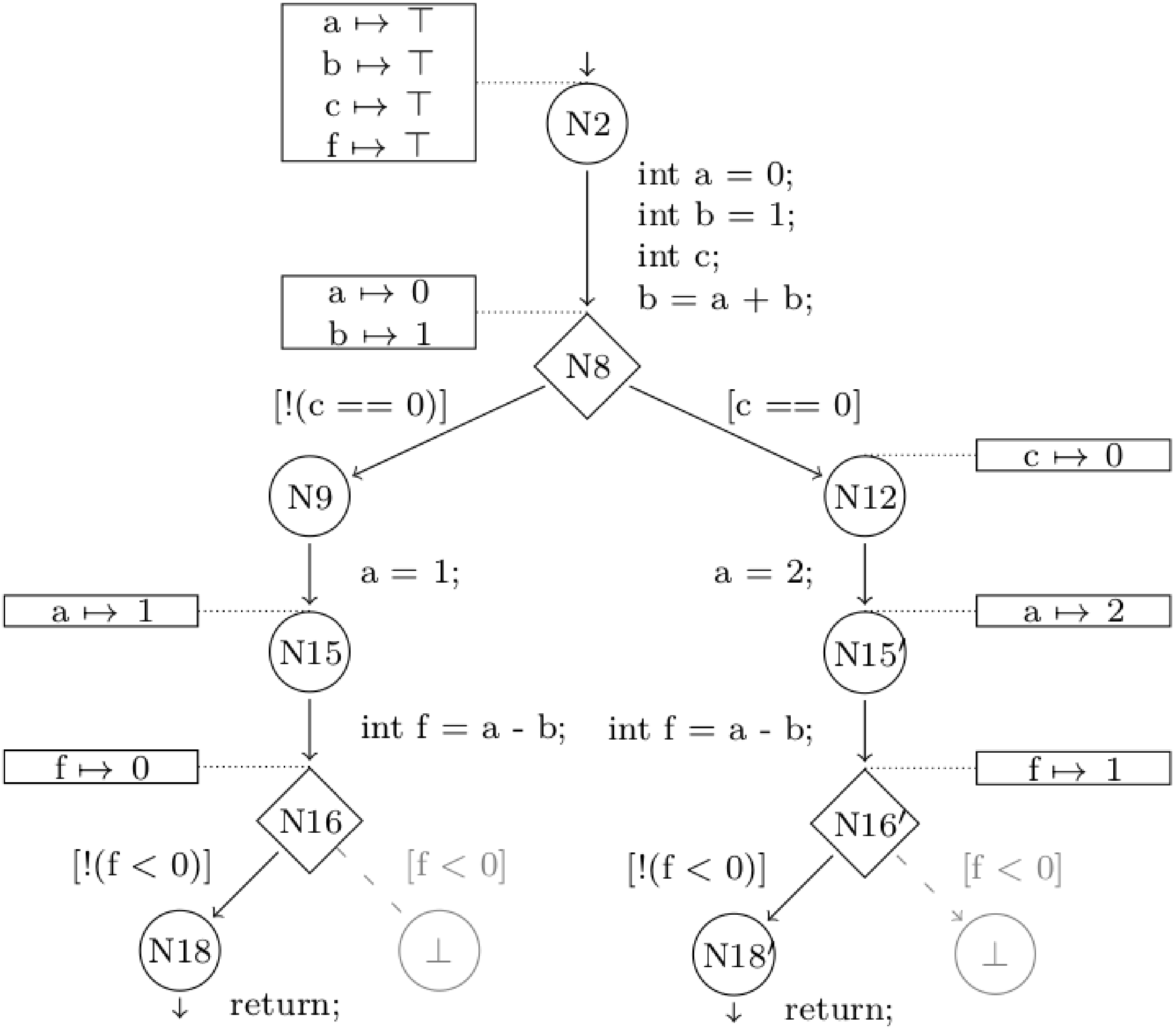
```
1  #include <assert.h>
2  int main() {
3     int a = 0;
4     int b = 1;
5     int c;
6     b = a + b;
7
8     if (c) {
9       a = 1;
10    }
11    else {
12      a = 2;
13    }
14
15    int f = a - b;
16    if (f < 0) {
17      assert(0);
18    }
19 }
```



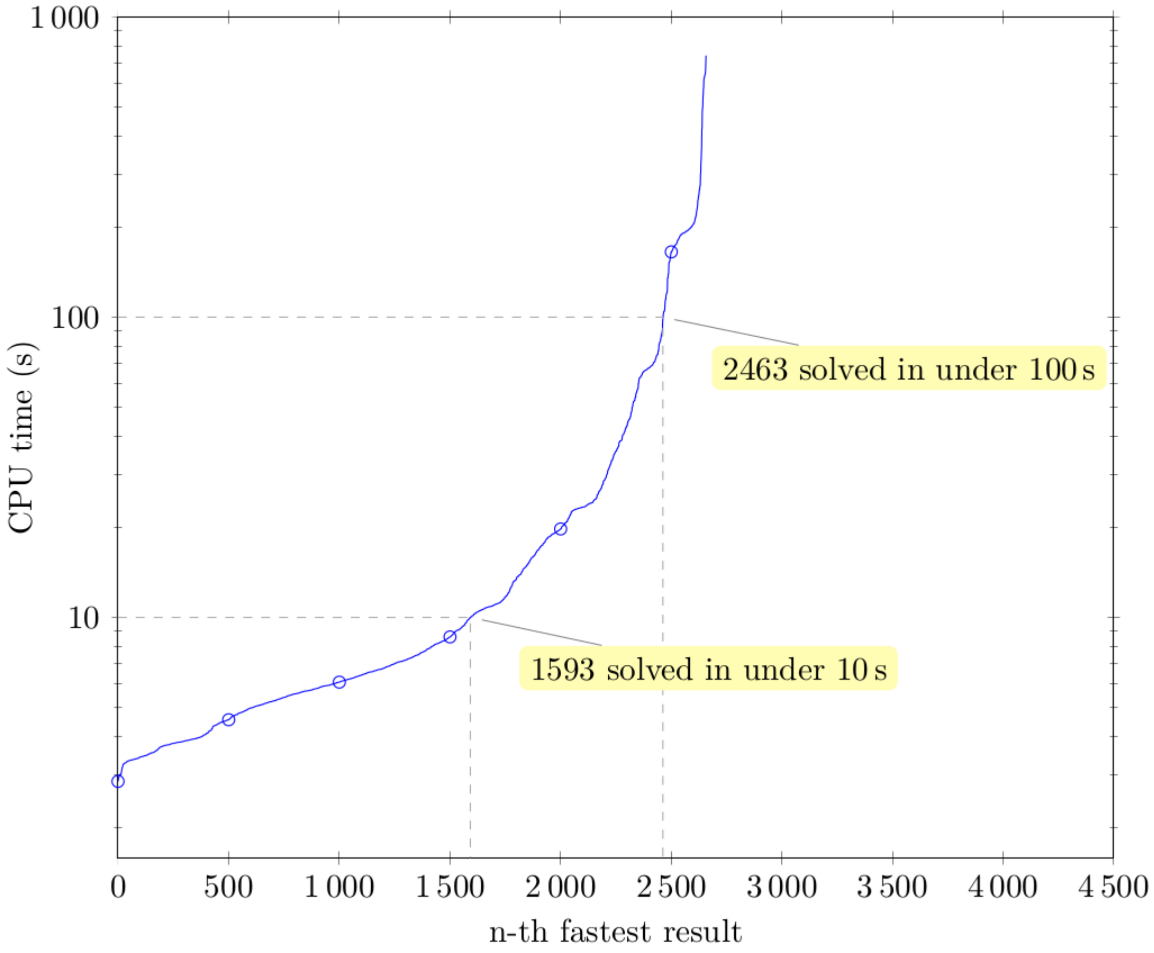
Value Analysis by Example



```
#include <assert.h>
int main() {
  int a = 0;
  int b = 1;
  int c;
  b = a + b;

  if (c) {
    a = 1;
  }
  else {
    a = 2;
  }

  int f = a - b;
  if (f < 0) {
    assert(0);
  }
}
```



Value Analysis by the Numbers

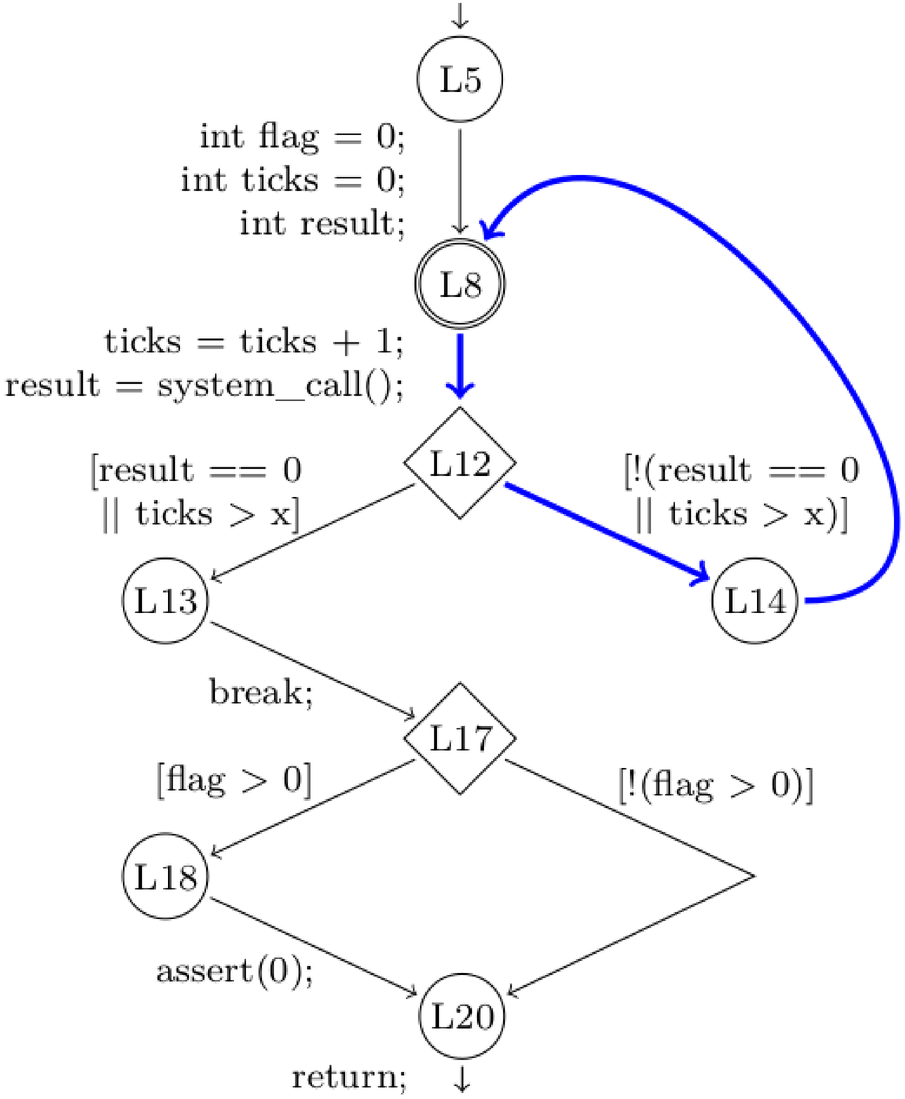


- Well over 4000 verification tasks from SV-COMP'16
- VA solves almost two thirds
- Under SV-COMP'16 rules, complete evaluation takes 440 hours
- 410 hours, or 93%, are wasted for unsolved verification tasks

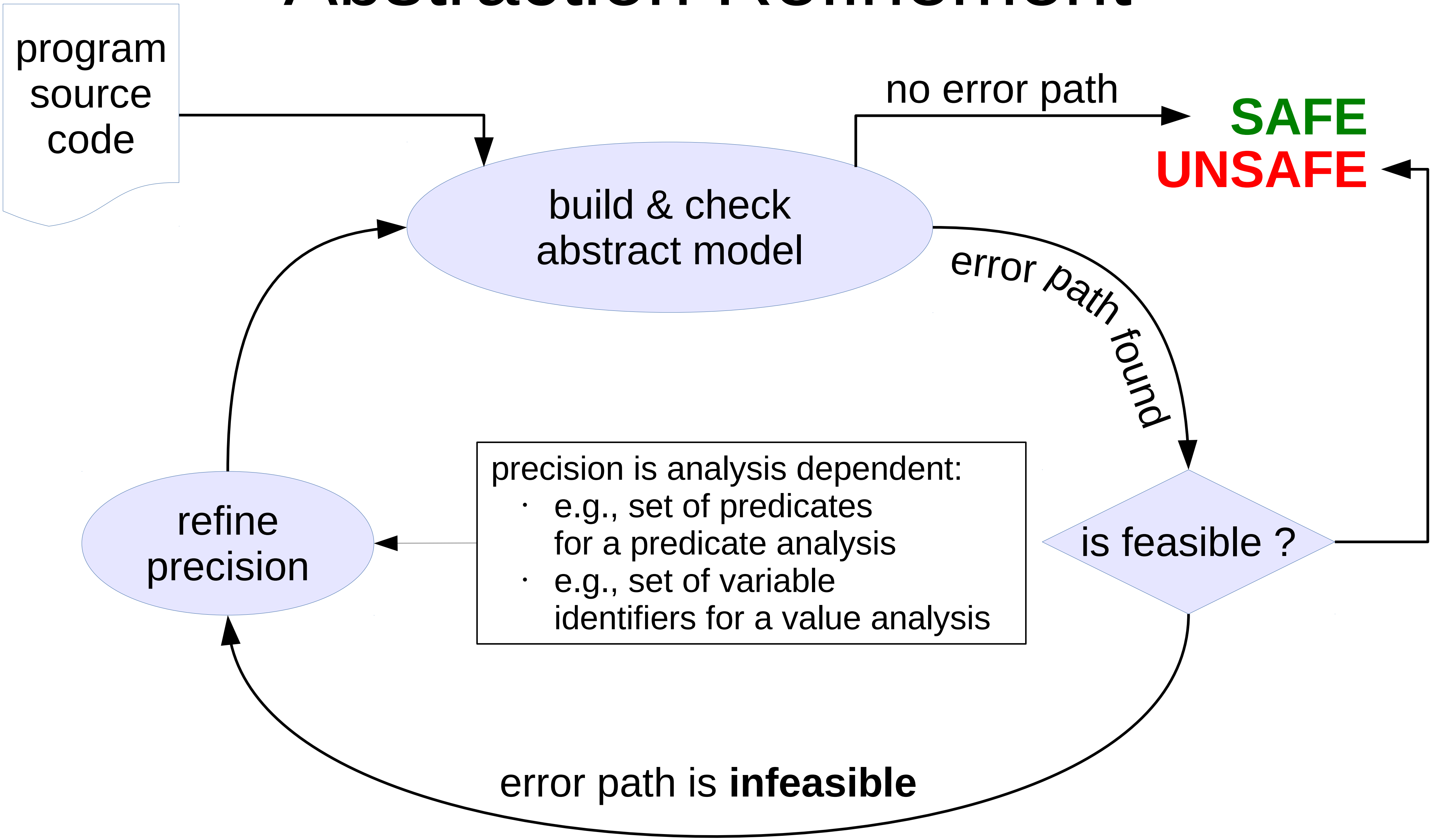
State-space explosion is prime reason for extreme resource consumption

State-Space Explosion

```
 1 #include <assert.h>
 2 extern int system_call();
 3
 4 int main(int x) {
 5   int flag = 0, ticks = 0;
 6   int result;
 7
 8   while(1) {
 9     ticks  = ticks + 1;
10     result = system_call();
11
12     if(result == 0 || ticks > x) {
13       break;
14     }
15   }
16
17   if(flag > 0) {
18     assert(0);
19   }
20 }
```



Counterexample-Guided Abstraction Refinement



program source code

build & check abstract model

no error path

SAFE

UNSAFE

error path found

is feasible ?

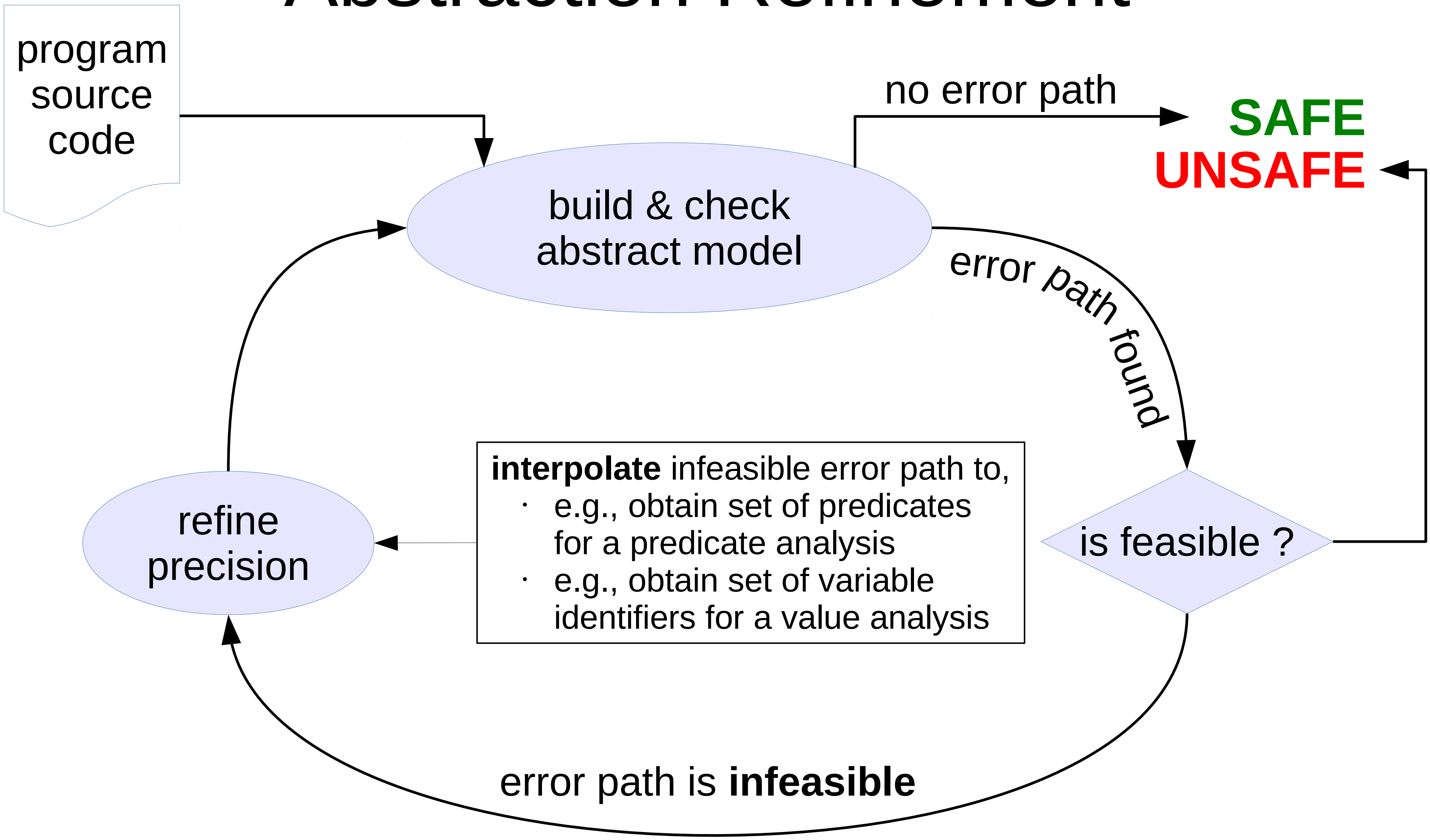
refine precision

precision is analysis dependent:

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

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 ?

interpolate infeasible error path to,

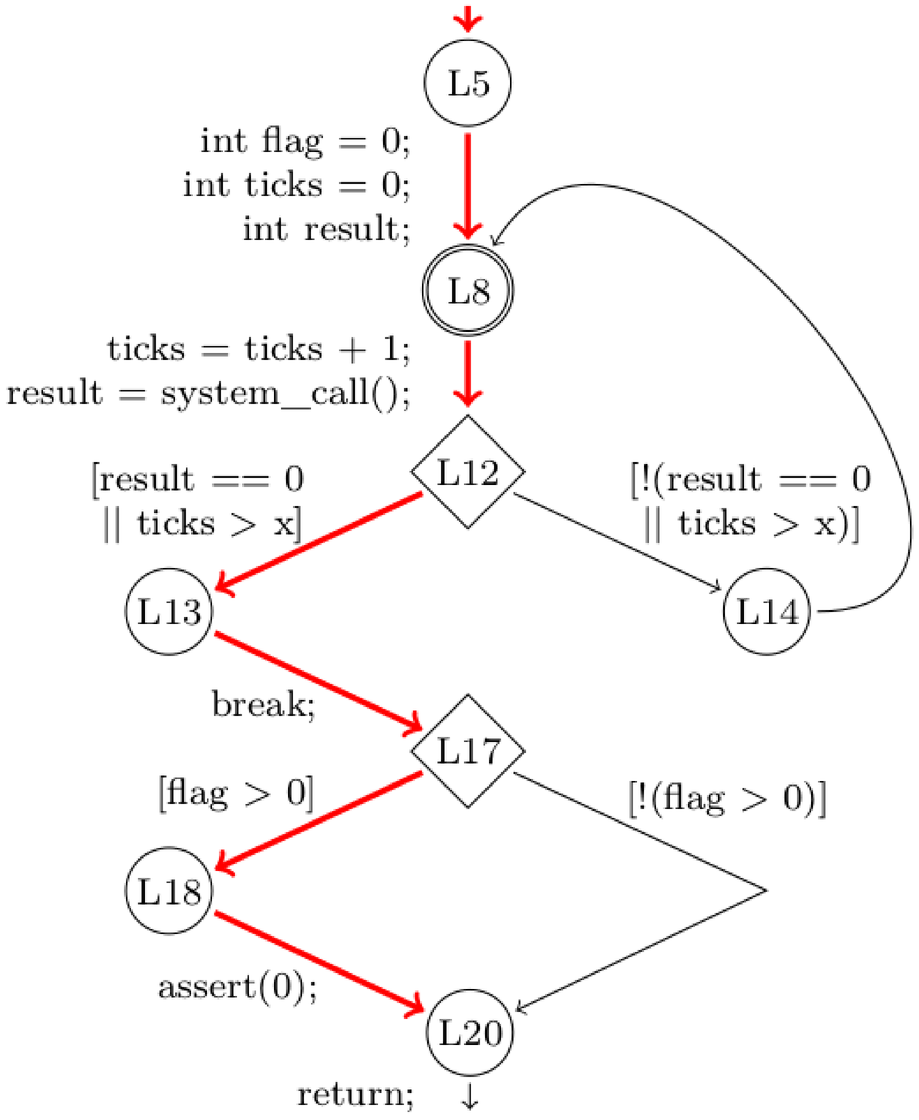
- e.g., obtain set of predicates for a predicate analysis
- e.g., obtain set of variable identifiers for a value analysis

refine precision

error path is **infeasible**

Craig Interpolation

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



φ^-

φ^+

itp

ψ

the interpolant

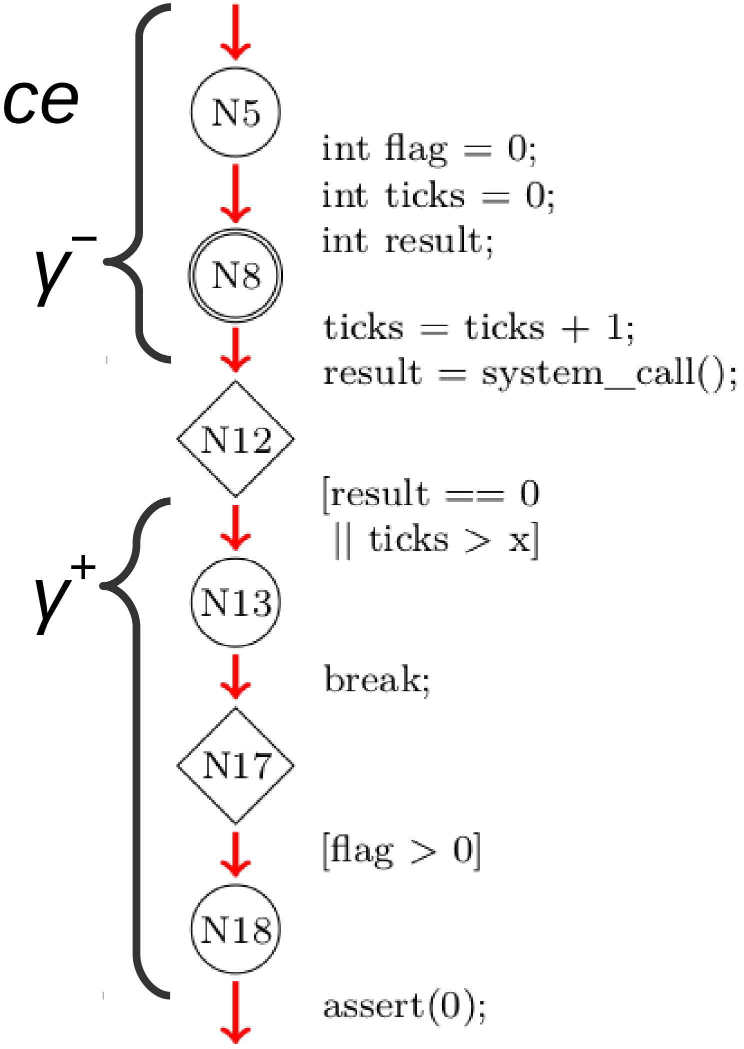
At L12 the interpolant ψ for φ^- and φ^+ could be: [flag = 0], or [flag ≤ 0], or ...

Value Interpolation

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

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

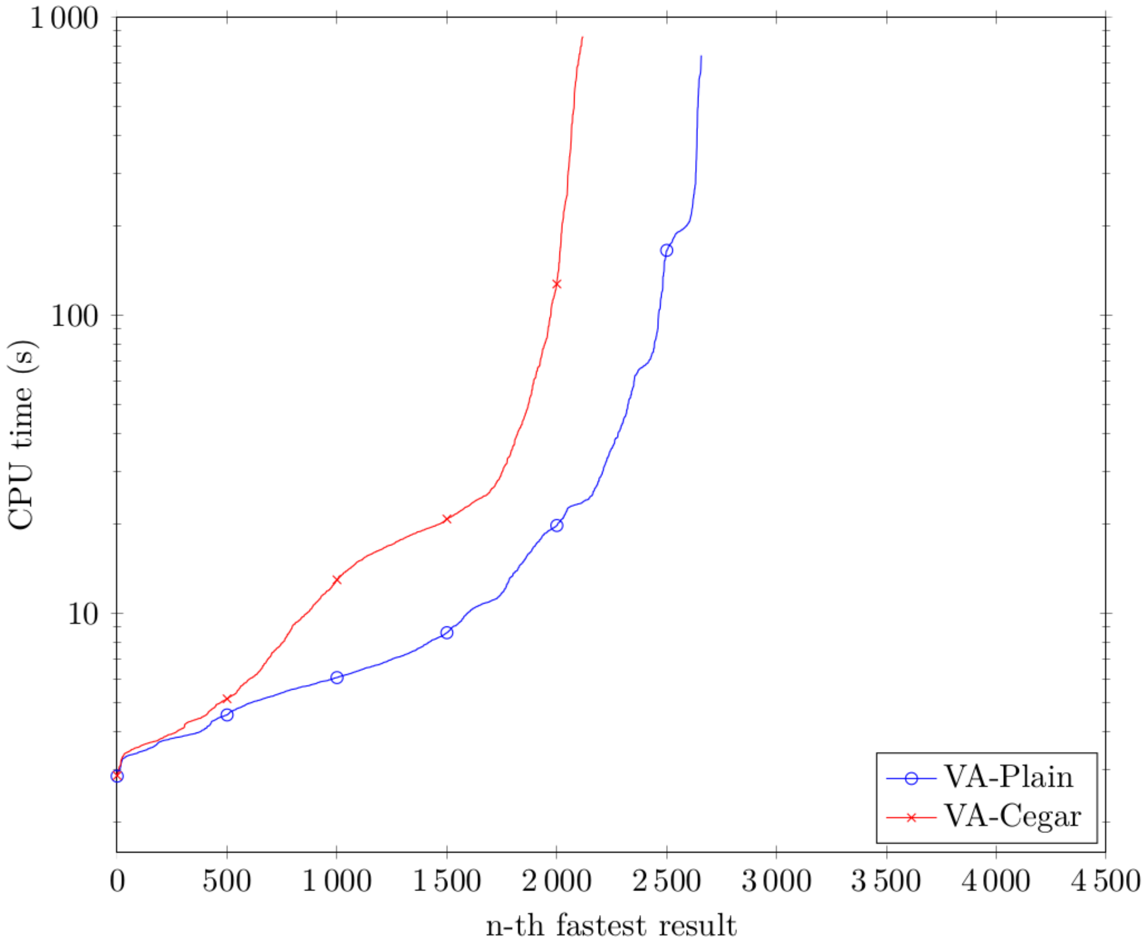
1) γ^- implies ψ
2) $\psi \wedge \gamma^+$ is unsatisfiable
3) ψ only contains symbols that are common to both γ^- and γ^+



```
int flag = 0;
int ticks = 0;
int result;

ticks = ticks + 1;
result = system_call();

[result == 0
 || ticks > x]

break;

[flag > 0]

assert(0);
```

A L12 the interpolant ψ for φ^- and φ^+ can only be: [flag = 0]

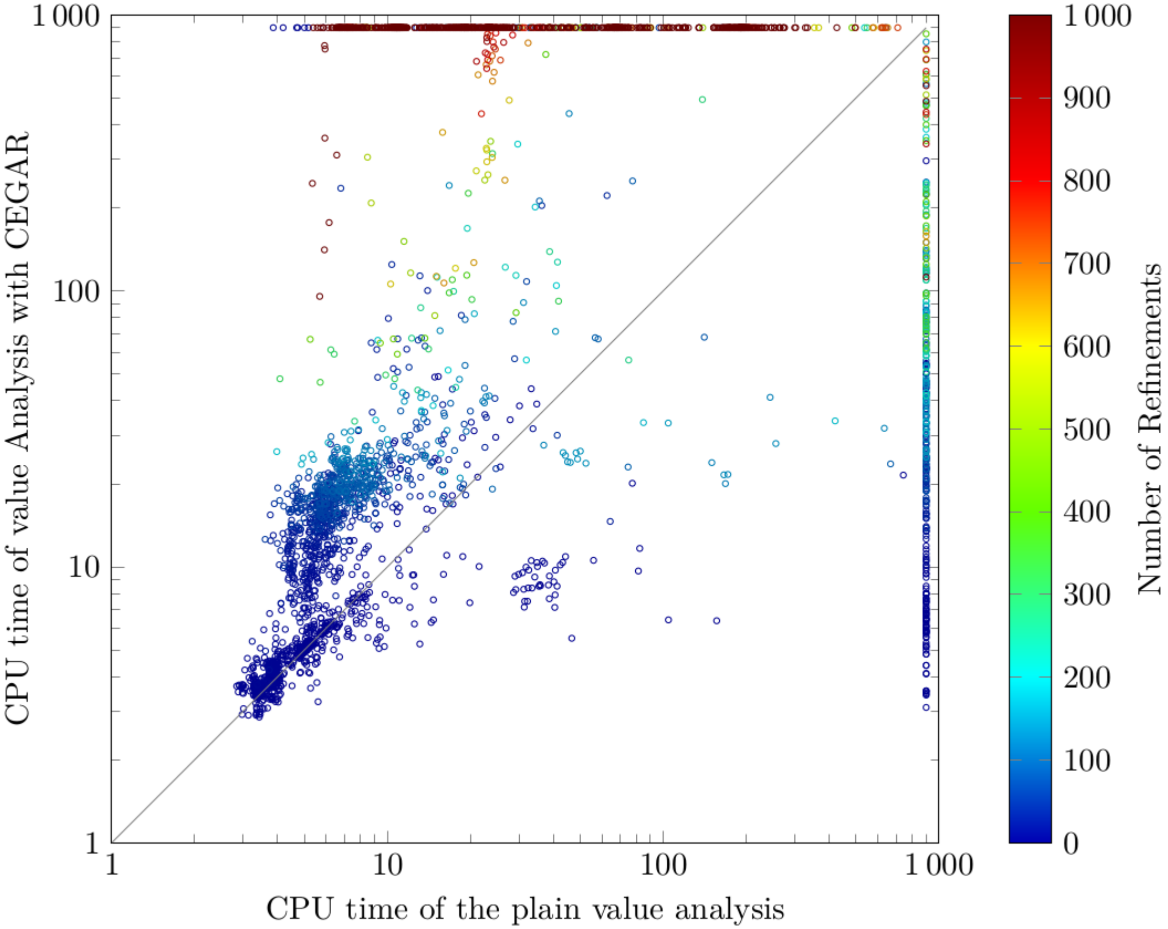
Comparison to Plain Value Analysis



- Significant improvements in DeviceDrivers64Linux
- Significant regressions in ECA and ProductLines
- In total solves around 500 verification task less

High number of refinements is prime reason for overall regression

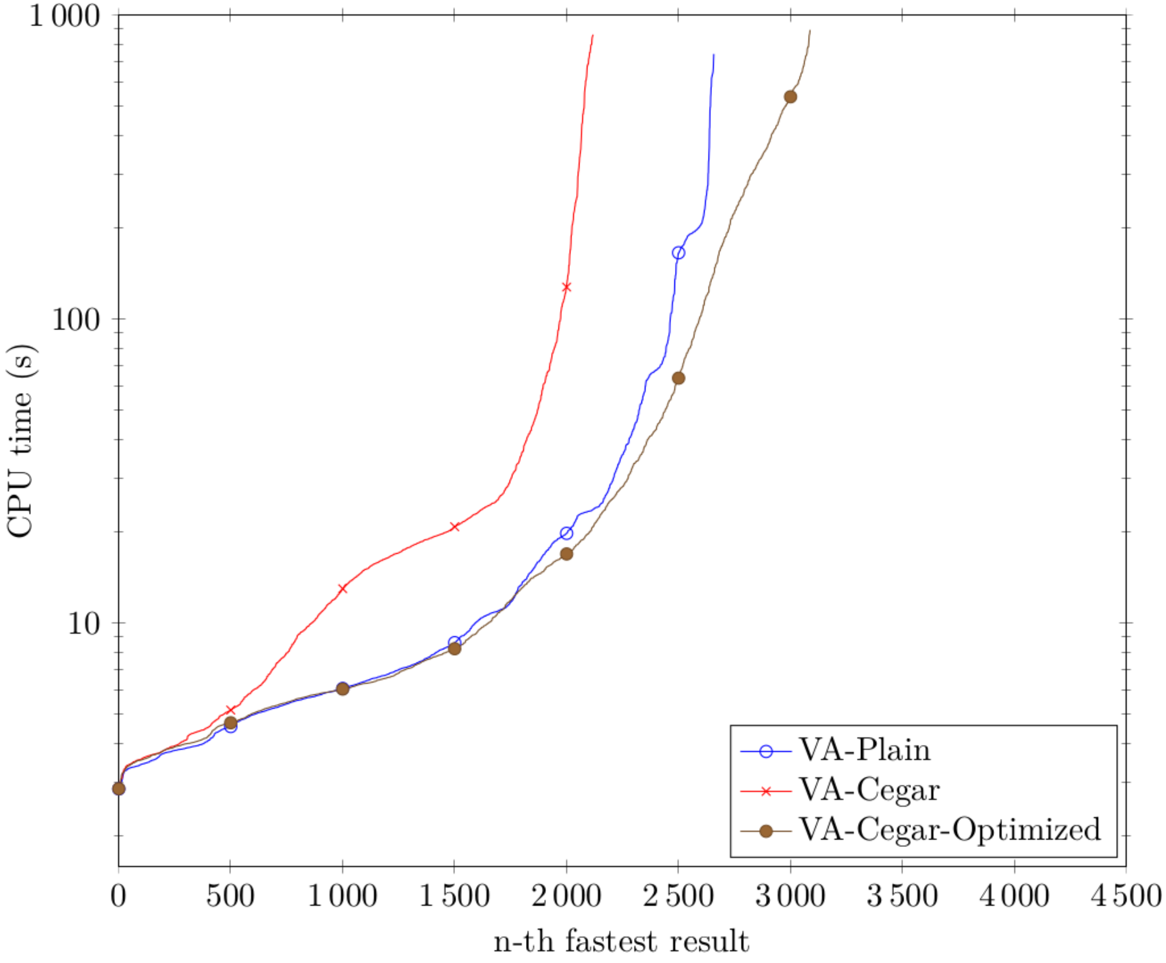
Inspecting Number of Refinements



At least three clusters distinguishable

- Solved by both #refinements < 200
- Solved only by VA-Cegar #refinements < 500
- Solved only by VA-Plain #refinements > 1000

Reducing Time for Refinements

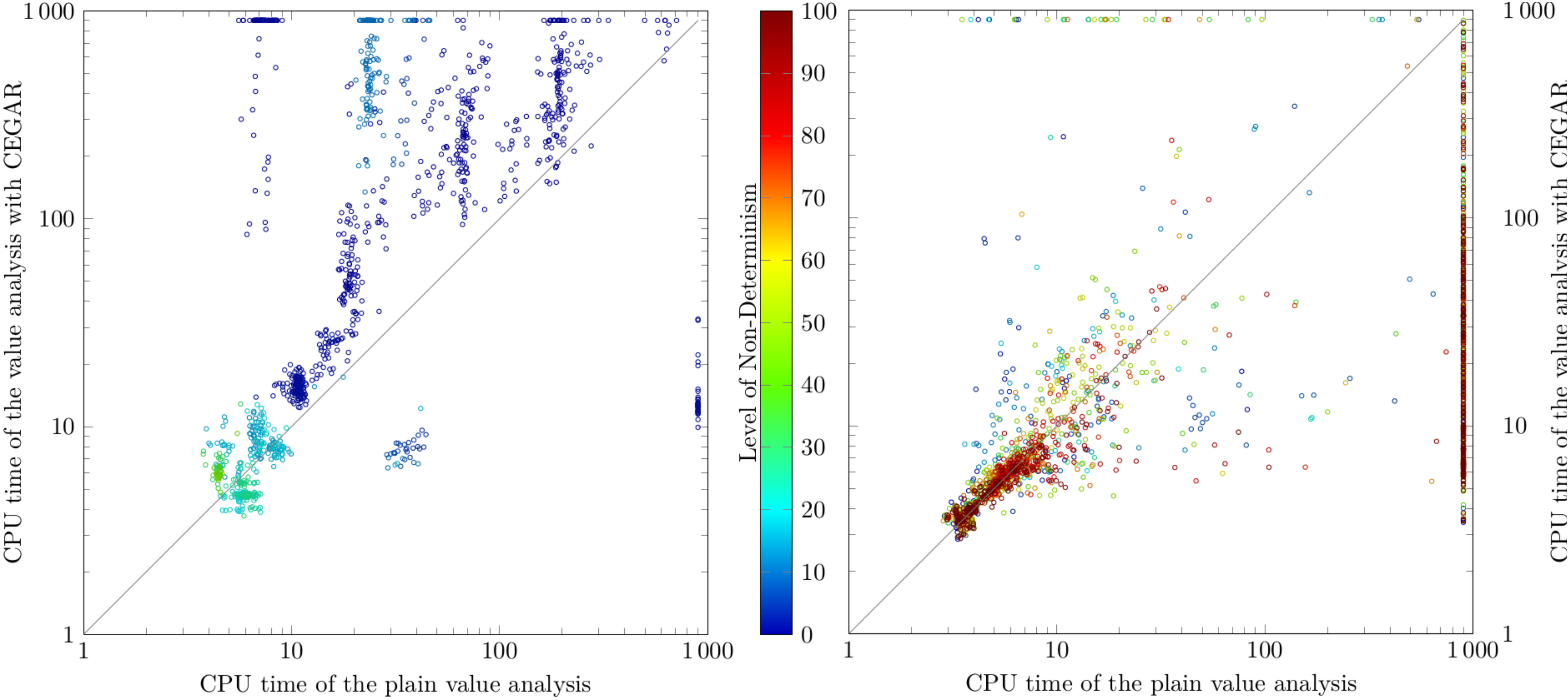


- Optimized Interpolation
    - Deepest Infeasible Suffix
    - Interpolant-Equality
- Optimized Refinement
    - “Scoped” Precision
    - Eager Restart

➢ CEGAR pays off, solving well over 400 tasks more

➢ Lazy abstraction is not well-suited for the Value Analysis

Level of Non-Determinism



Low level of non-determinism:
Use Plain Value Analysis

High level of non-determinism:
Use Value Analysis with CEGAR

- Valid indicator whether to perform abstraction or not

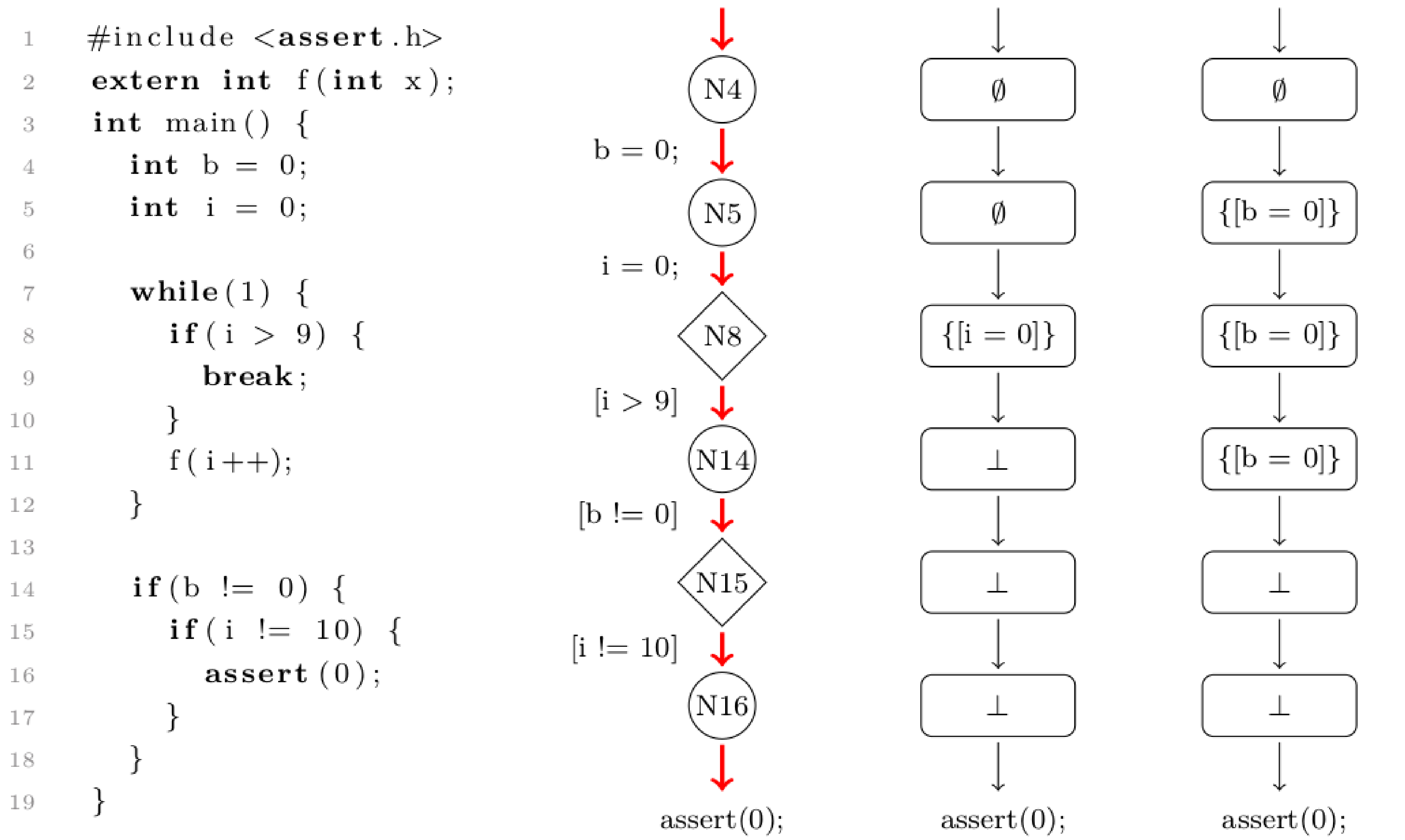
Versatility of Value Interpolation

- Applicable to other analyses
    - Octagon analysis
    - Symbolic execution analysis
- Enables regression verification
- Parallel composition with Predicate Analysis

➢ Availablilty of several effective analyses based on CEGAR

➢ Next: Techniques that may benefit **all such analyses**

Infeasible Sliced Prefixes and Refinement Selection



```
#include <assert.h>
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);
    }
  }
}
```

(a) verification task

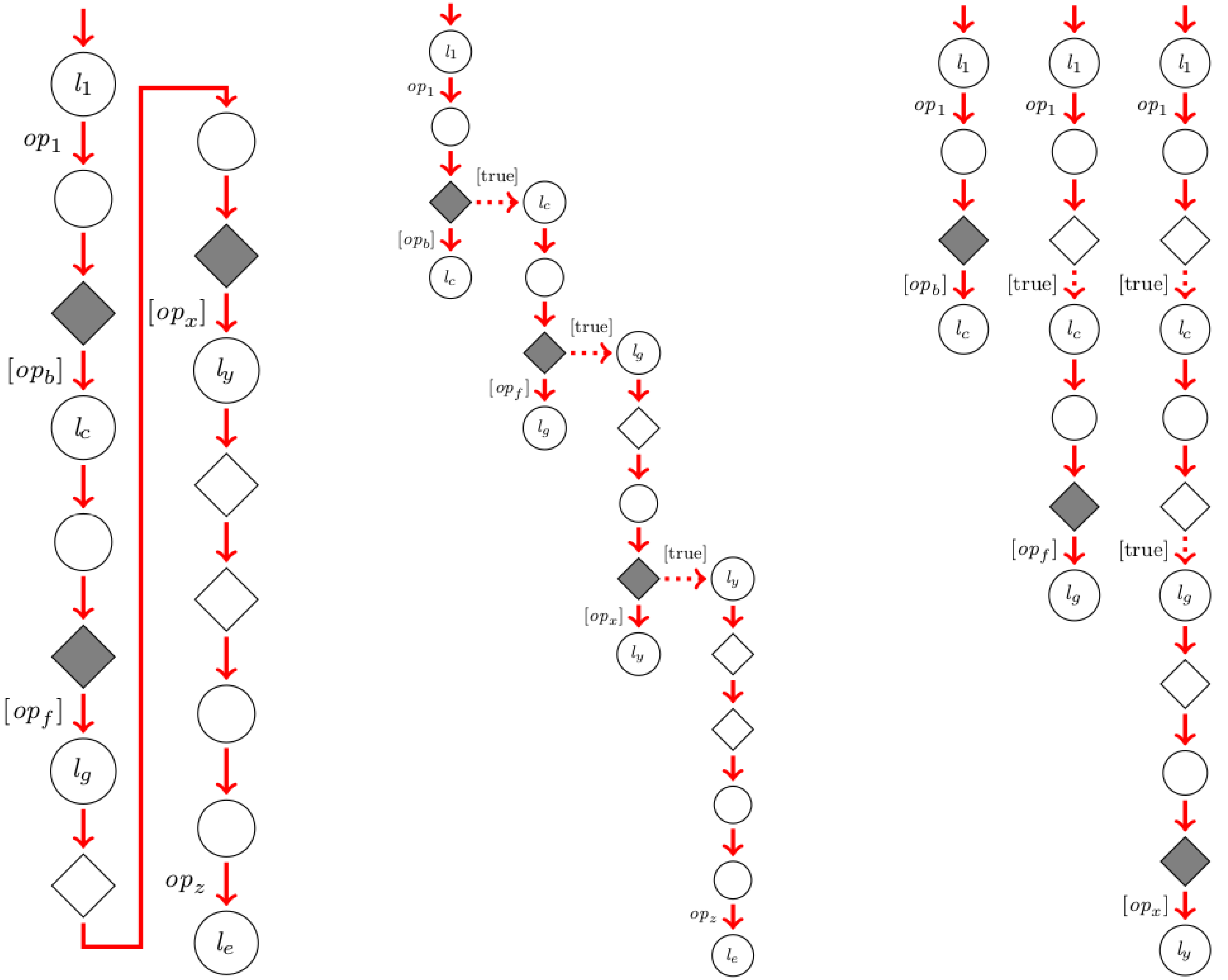
(b) error path

(c) bad sequence

(d) good sequence

Extraction of Infeasible Sliced Prefixes

[Sliced Path Prefixes: An Effective Method to Enable Refinement Selection, 2015, Beyer, Löwe, Wendler]



(a) Infeasible error path

(b) Cascade of sliced prefixes

(c) Infeasible sliced prefixes

Main Message

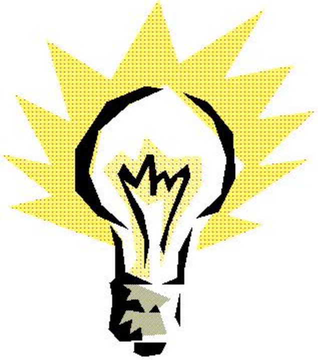
Any **infeasible sliced prefix φ,**

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.

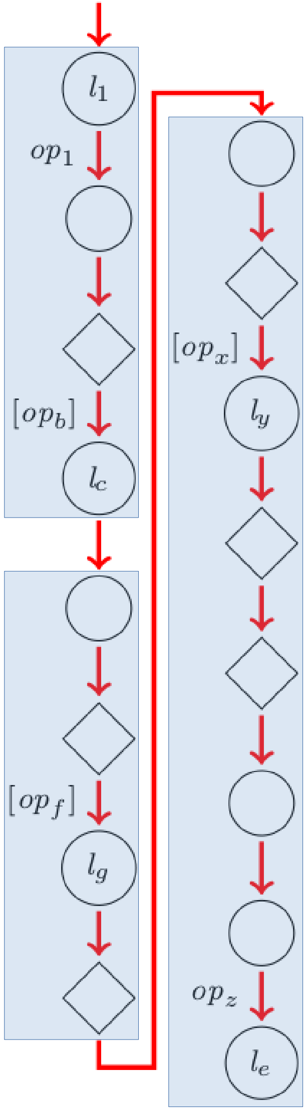


- We can use any prefix we want for interpolation !

Sliced Prefixes - Further Applications

- Enables guided refinement selection
- Improves effectiveness and efficiency of static refinement
- Speeds up Value Interpolation significantly
- Impressive results in combination with symbolic execution
- Better control for global refinement
  - All target states at once
  - Each target state with an unique refinement
- Infeasible Sliced Prefixes for ABE?

Infeasible Sliced Prefixes for ABE?



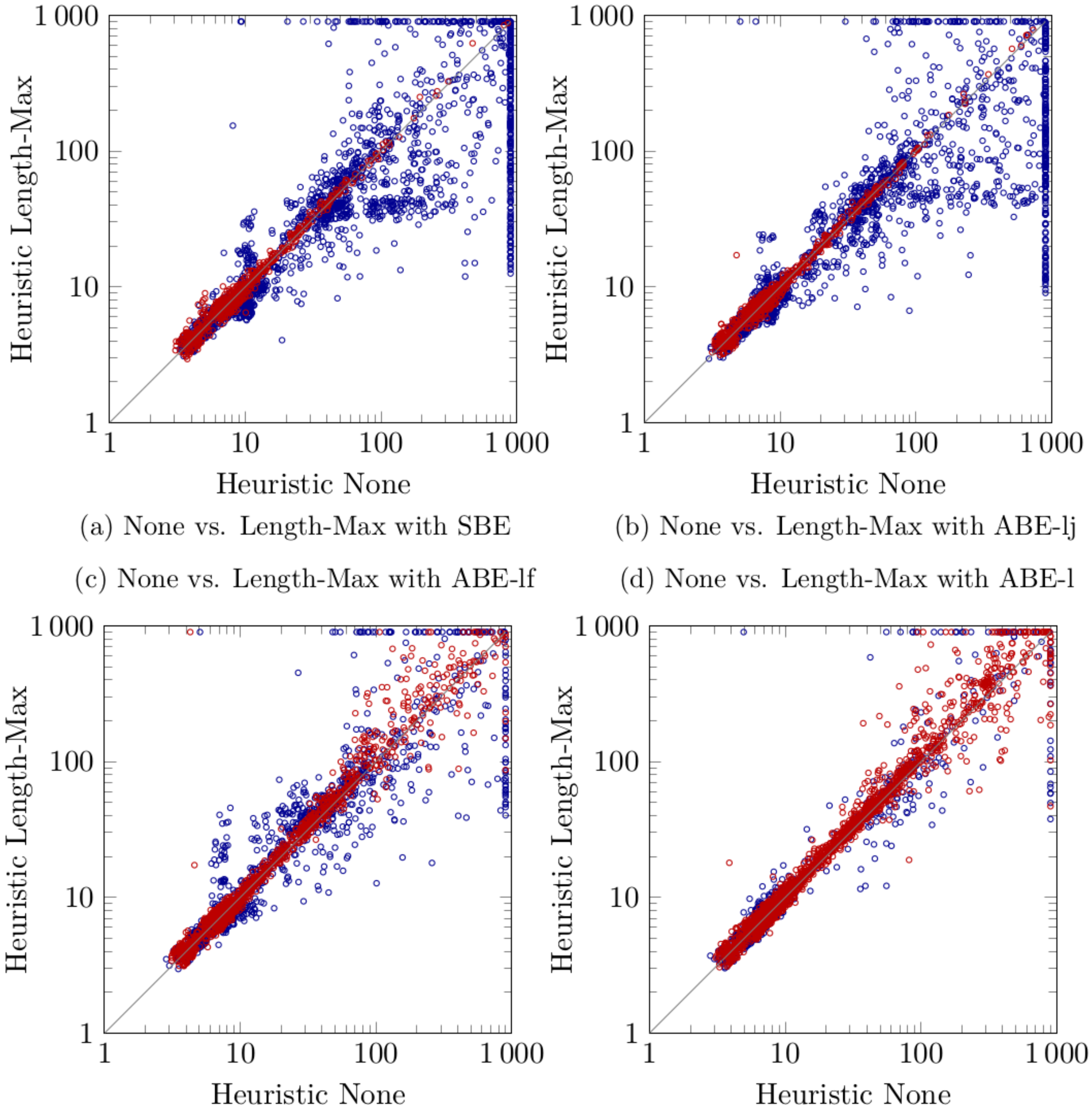
- ABE: block size can have any size
- ABE-encoded path represent different paths
  - Simply pick one? No!
  - Simply pick all? No!

➢Just think in blocks

- SBE-encoded paths also are made of blocks
- SBE: each block contains a single statement

➢For ABE: apply same approach as for SBE / Value Analysis

Infeasible Sliced Prefixes for ABE



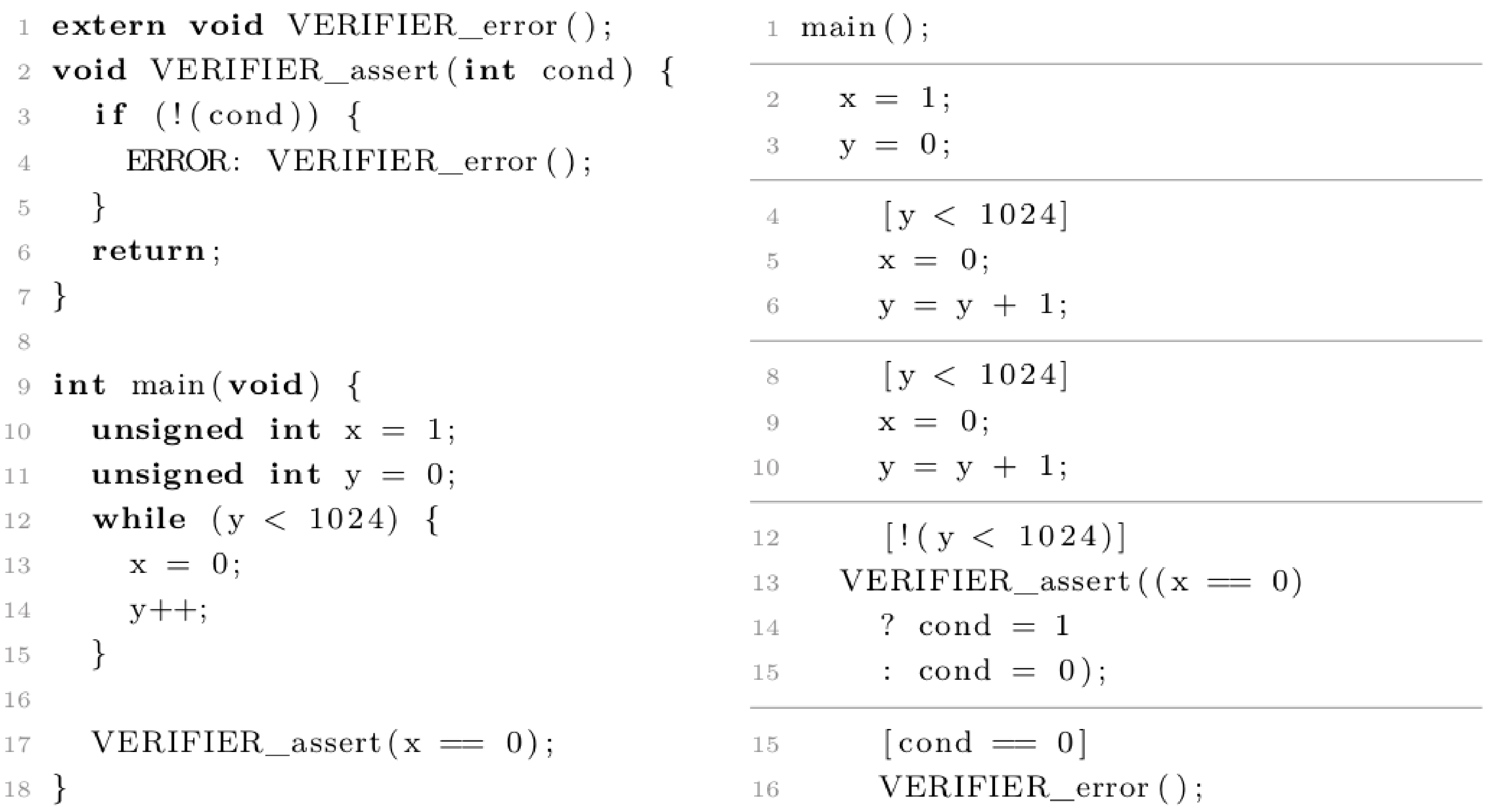
(a) None vs. Length-Max with SBE

(b) None vs. Length-Max with ABE-lj

(c) None vs. Length-Max with ABE-lf

(d) None vs. Length-Max with ABE-l

Elimination of Infeasible Sliced Prefixes !



```
extern void VERIFIER_error();
void VERIFIER_assert(int cond) {
  if (!(cond)) {
    ERROR: VERIFIER_error();
  }
  return;
}

int main(void) {
  unsigned int x = 1;
  unsigned int y = 0;
  while (y < 1024) {
    x = 0;
    y++;
  }

  VERIFIER_assert(x == 0);
}
```

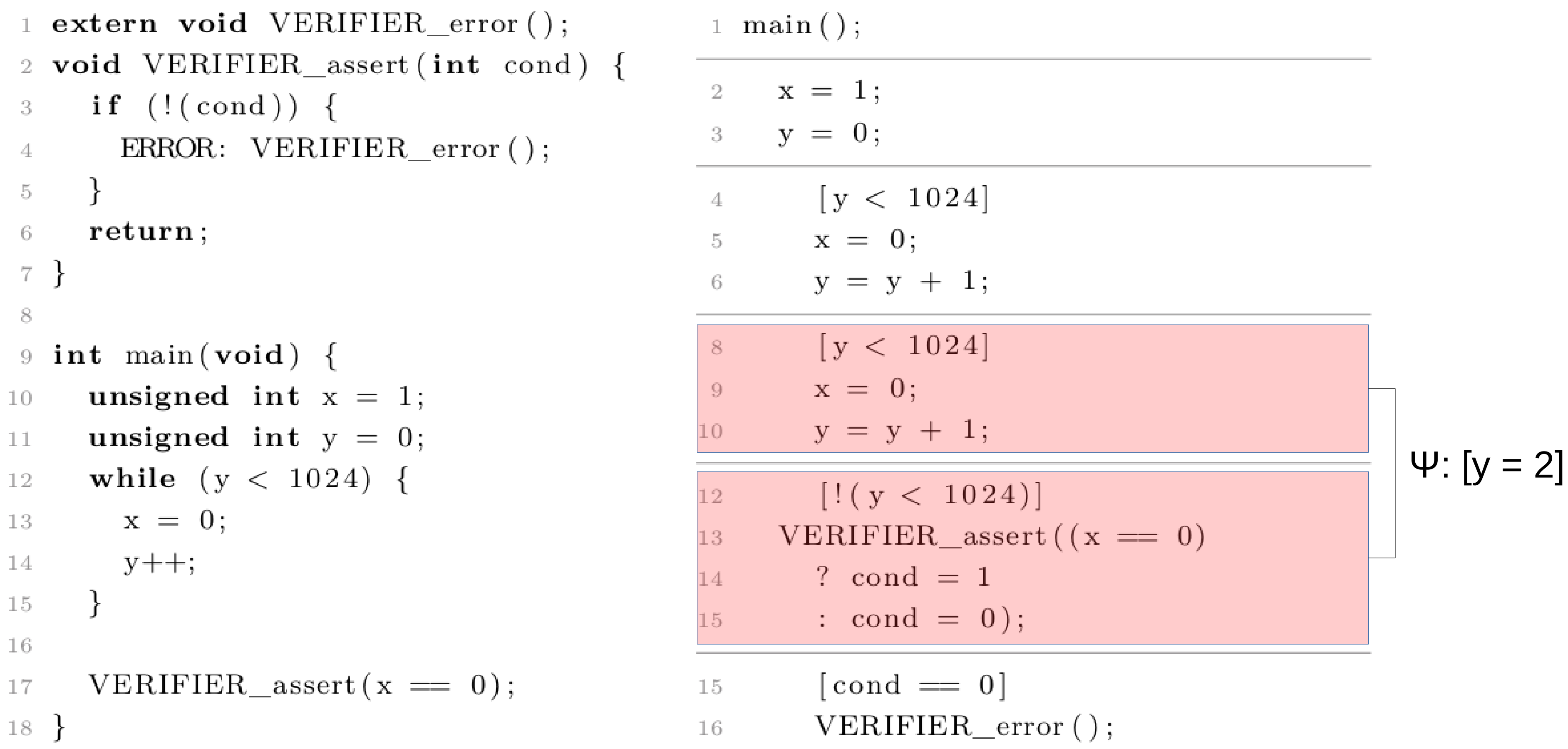
(a) Source code of verification task

```
main();
--------------------------------
  x = 1;
  y = 0;
--------------------------------
    [y < 1024]
    x = 0;
    y = y + 1;
--------------------------------
    [y < 1024]
    x = 0;
    y = y + 1;
--------------------------------
    [!(y < 1024)]
  VERIFIER_assert((x == 0)
    ? cond = 1
    : cond = 0);
--------------------------------
    [cond == 0]
    VERIFIER_error();
```

(b) Error path over two loop iteration

Verification task `const_true-unreach-call1.c` from the official SVCOMP'16 repository, and a possible infeasible error path when analyzing the task with ABE-lf

Elimination of Infeasible Sliced Prefixes !



```
extern void VERIFIER_error();
void VERIFIER_assert(int cond) {
  if (!(cond)) {
    ERROR: VERIFIER_error();
  }
  return;
}

int main(void) {
  unsigned int x = 1;
  unsigned int y = 0;
  while (y < 1024) {
    x = 0;
    y++;
  }

  VERIFIER_assert(x == 0);
}
```

(a) Source code of verification task

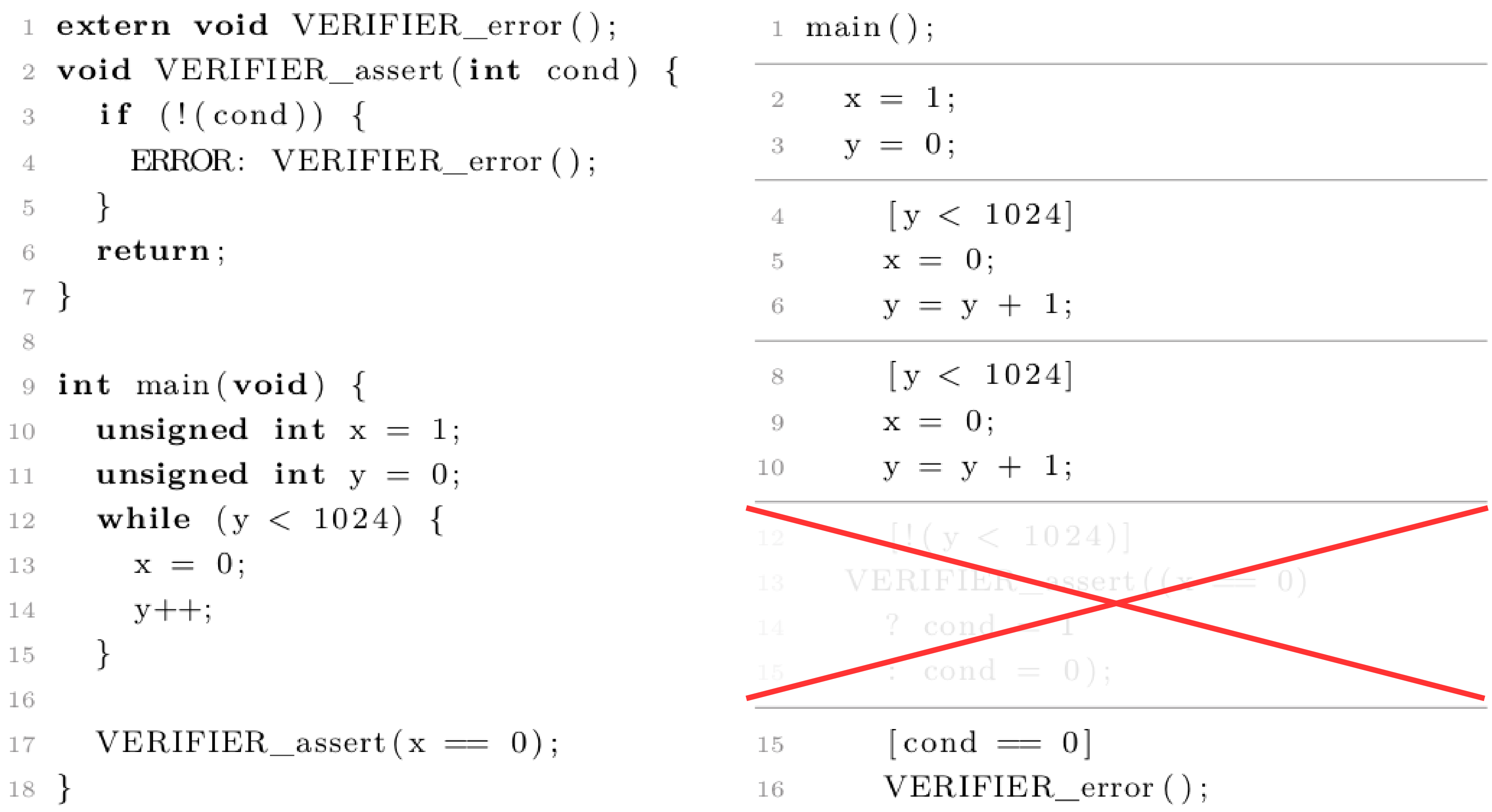
```
main();

  x = 1;
  y = 0;

      [y < 1024]
      x = 0;
      y = y + 1;

      [y < 1024]
      x = 0;
      y = y + 1;

      [!(y < 1024)]
  VERIFIER_assert((x == 0)
      ? cond = 1
      : cond = 0);

      [cond == 0]
      VERIFIER_error();
```

Ψ: [y = 2]

(b) Error path over two loop iteration

Verification task `const_true-unreach-call1.c` from the official SVCOMP’16 repository, and a possible infeasible error path when analyzing the task with ABE-lf

Elimination of Infeasible Sliced Prefixes !



```
extern void VERIFIER_error();
void VERIFIER_assert(int cond) {
  if (!(cond)) {
    ERROR: VERIFIER_error();
  }
  return;
}

int main(void) {
  unsigned int x = 1;
  unsigned int y = 0;
  while (y < 1024) {
    x = 0;
    y++;
  }

  VERIFIER_assert(x == 0);
}
```

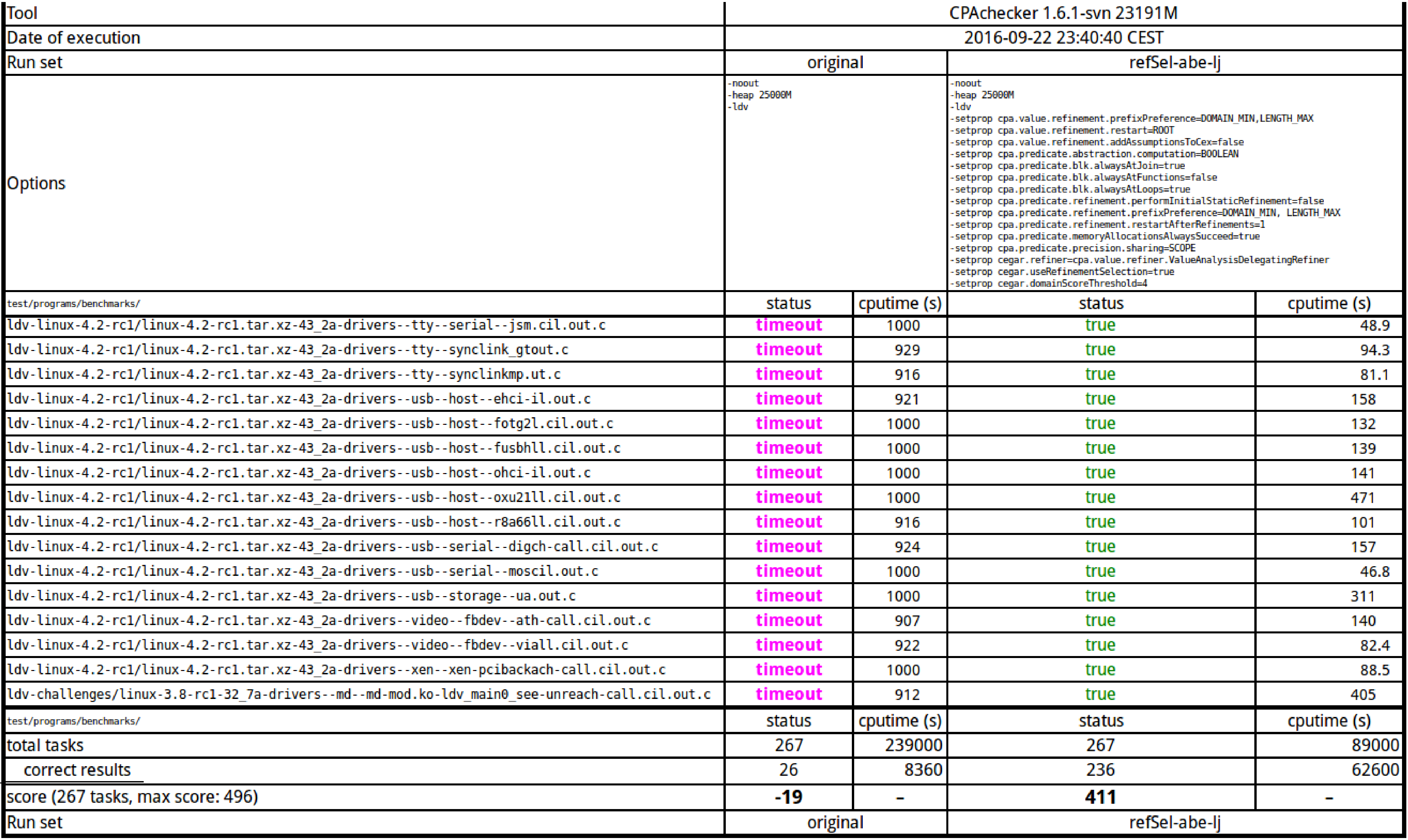
(a) Source code of verification task

```
main();
  x = 1;
  y = 0;
    [y < 1024]
    x = 0;
    y = y + 1;
    [y < 1024]
    x = 0;
    y = y + 1;
    [!(y < 1024)]
  VERIFIER_assert((x == 0)
    ? cond = 1
    : cond = 0);
    [cond == 0]
    VERIFIER_error();
```

(b) Error path over two loop iteration

- For ABE: this approach is also not perfect
- Any other ideas?

Quite good for LDV



Tool	CPAchecker 1.6.1-svn 23191M			
Date of execution	2016-09-22 23:40:40 CEST			
Run set	original		refSel-abe-lj	
Options	-noout -heap 25000M -ldv		-noout -heap 25000M -ldv -setprop cpa.value.refinement.prefixPreference=DOMAIN_MIN,LENGTH_MAX -setprop cpa.value.refinement.restart=ROOT -setprop cpa.value.refinement.addAssumptionsToCex=false -setprop cpa.predicate.abstraction.computation=BOOLEAN -setprop cpa.predicate.blk.alwaysAtJoin=true -setprop cpa.predicate.blk.alwaysAtFunctions=false -setprop cpa.predicate.blk.alwaysAtLoops=true -setprop cpa.predicate.refinement.performInitialStaticRefinement=false -setprop cpa.predicate.refinement.prefixPreference=DOMAIN_MIN, LENGTH_MAX -setprop cpa.predicate.refinement.restartAfterRefinements=1 -setprop cpa.predicate.memoryAllocationsAlwaysSucceed=true -setprop cpa.predicate.precision.sharing=SCOPE -setprop cegar.refiner=cpa.value.refiner.ValueAnalysisDelegatingRefiner -setprop cegar.useRefinementSelection=true -setprop cegar.domainScoreThreshold=4	
test/programs/benchmarks/	status	cputime (s)	status	cputime (s)
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--tty--serial--jsm.cil.out.c	timeout	1000	true	48.9
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--tty--synclink_gtout.c	timeout	929	true	94.3
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--tty--synclinkmp.ut.c	timeout	916	true	81.1
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--usb--host--ehci-il.out.c	timeout	921	true	158
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--usb--host--fotg2l.cil.out.c	timeout	1000	true	132
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--usb--host--fusbhll.cil.out.c	timeout	1000	true	139
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--usb--host--ohci-il.out.c	timeout	1000	true	141
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--usb--host--oxu21ll.cil.out.c	timeout	1000	true	471
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--usb--host--r8a66ll.cil.out.c	timeout	916	true	101
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--usb--serial--digch-call.cil.out.c	timeout	924	true	157
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--usb--serial--moscil.out.c	timeout	1000	true	46.8
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--usb--storage--ua.out.c	timeout	1000	true	311
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--video--fbdev--ath-call.cil.out.c	timeout	907	true	140
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--video--fbdev--viall.cil.out.c	timeout	922	true	82.4
ldv-linux-4.2-rc1/linux-4.2-rc1.tar.xz-43_2a-drivers--xen--xen-pcibackach-call.cil.out.c	timeout	1000	true	88.5
ldv-challenges/linux-3.8-rc1-32_7a-drivers--md--md-mod.ko-ldv_main0_see-unreach-call.cil.out.c	timeout	912	true	405
test/programs/benchmarks/	status	cputime (s)	status	cputime (s)
total tasks	267	239000	267	89000
correct results	26	8360	236	62600
score (267 tasks, max score: 496)	**-19**	-	**411**	-
Run set	original		refSel-abe-lj	

Questions ?