C for Yourself:
Comparison of Front-End Techniques for Formal Verification

Levente Bajczi, *Zsófia Ádám, Vince Molnár*

Budapest University of Technology and Economics, Budapest, Hungary

Department of Measurement and Information Systems

This research was partially funded by the European Commission and the Hungarian Authorities (NKFIH) through the Arrowhead Tools project (EU grant agreement No. 826452 (https://cordis.europa.eu/project/id/826452), NKFIH grant 2019-2.1.3-NEMZ ECSEL-2019- 00003); and by the ÚNKP-21-2-I-BME-142 New National Excellence Program of the Ministry for Innovation and Technology from the Source of the National Research, Development and Innovation Fund.

MŰEGYETEM 1782

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATICS • BME

ftsrg **Critical Systems Research Group**

Motivation

Motivation

- We are the developers of Θtheta

https://github.com/ftsrg/theta/

Motivation

- We are the developers of Θtheta
  - Highly configurable CEGAR framework

https://github.com/ftsrg/theta/

Motivation

- We are the developers of Θtheta
  - Highly configurable CEGAR framework
- 2021: Debut at SV-COMP using **LLVM**

https://github.com/ftsrg/theta/

Motivation

- We are the developers of Θtheta
  - Highly configurable CEGAR framework
- 2021: Debut at SV-COMP using **LLVM**
  - Subpar performance for CEGAR

https://github.com/ftsrg/theta/

Motivation

- We are the developers of Θtheta
  - Highly configurable CEGAR framework
- 2021: Debut at SV-COMP using **LLVM**
  - Subpar performance for CEGAR
  - Most tasks solved by BMC

https://github.com/ftsrg/theta/

Motivation

- We are the developers of Θtheta
  - Highly configurable CEGAR framework
- 2021: Debut at SV-COMP using **LLVM**
  - Subpar performance for CEGAR
  - Most tasks solved by BMC
- 2022: 2nd year at SV-COMP using a custom C parser

ANTLR

https://github.com/ftsrg/theta/

ftsrg

Motivation

- We are the developers of Θtheta
  - Highly configurable CEGAR framework
- 2021: Debut at SV-COMP using **LLVM**
  - Subpar performance for CEGAR
  - Most tasks solved by BMC
- 2022: 2nd year at SV-COMP using a custom C parser
  - Strong improvement for CEGAR

ANTLR

https://github.com/ftsrg/theta/

ftsrg

Motivation

- We are the developers of Θtheta
  - Highly configurable CEGAR framework
- 2021: Debut at SV-COMP using **LLVM**
  - Subpar performance for CEGAR
  - Most tasks solved by BMC

- 2022: 2nd year at SV-COMP using a custom C parser
  - Strong improvement for CEGAR
  - **Why?**

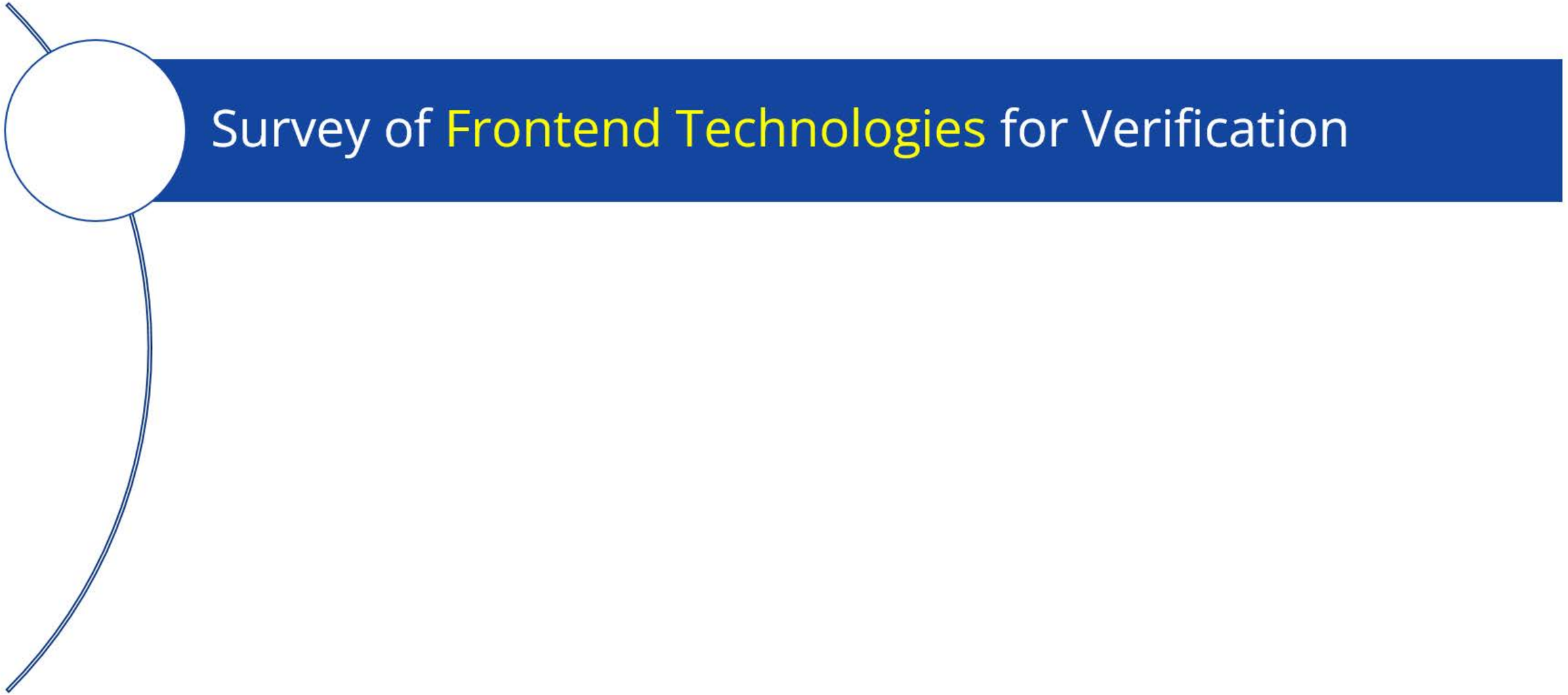
ANTLR

https://github.com/ftsrg/theta/

ftsrg

Contributions

Contributions



Contributions

Contributions

- Survey of Frontend Technologies for Verification
- Theoretical Justification of Observed Patterns
- Proposal of a Verification-First Frontend Project

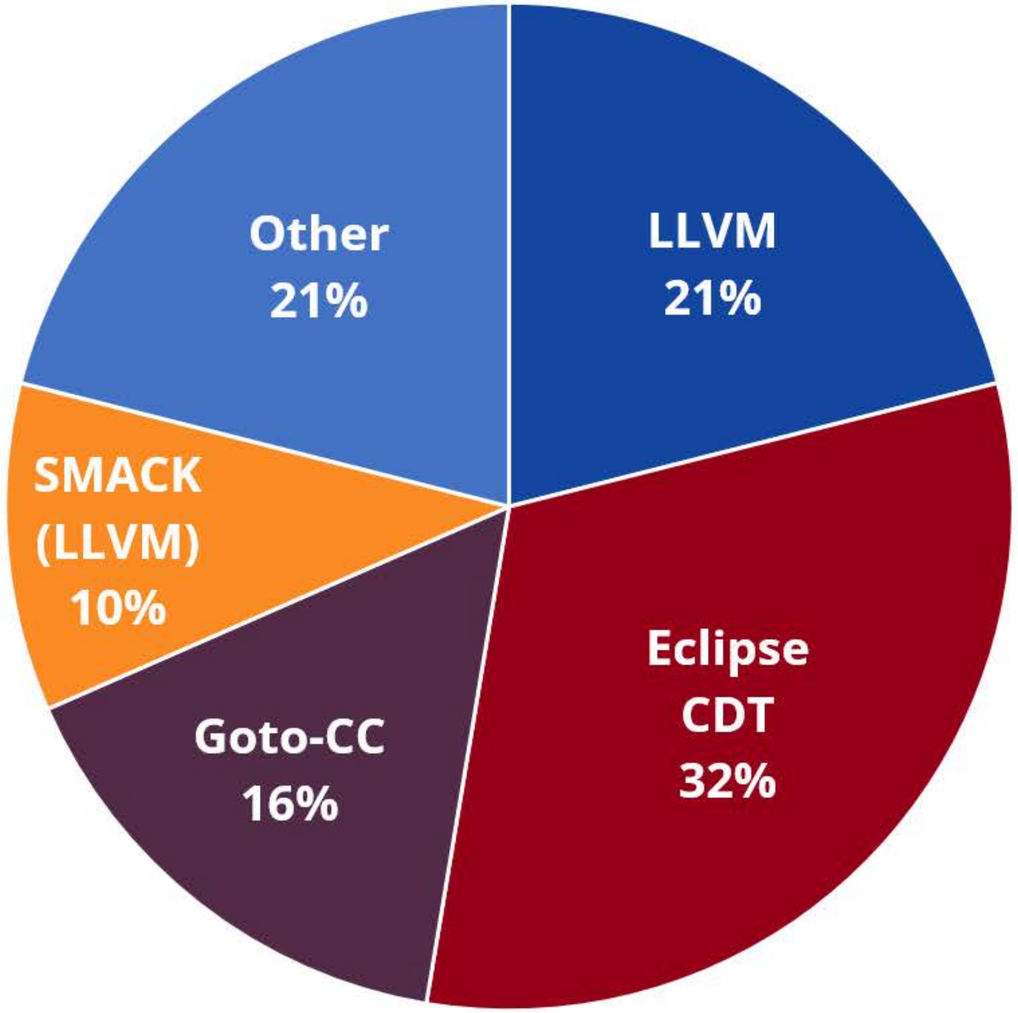
Frontend Technologies

Survey

What Frontends do Tools Use?

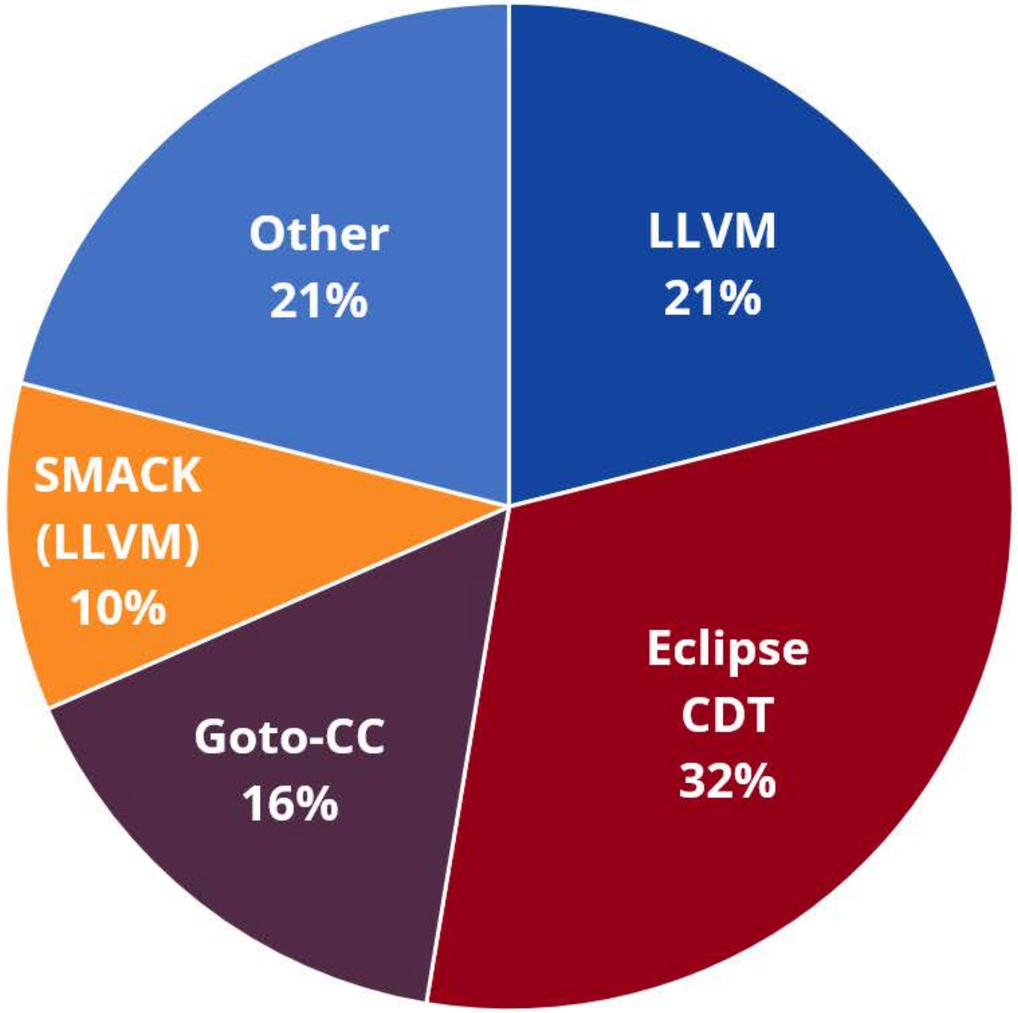
Source: SV-COMP 2021

What Frontends do Tools Use?



Source: SV-COMP 2021

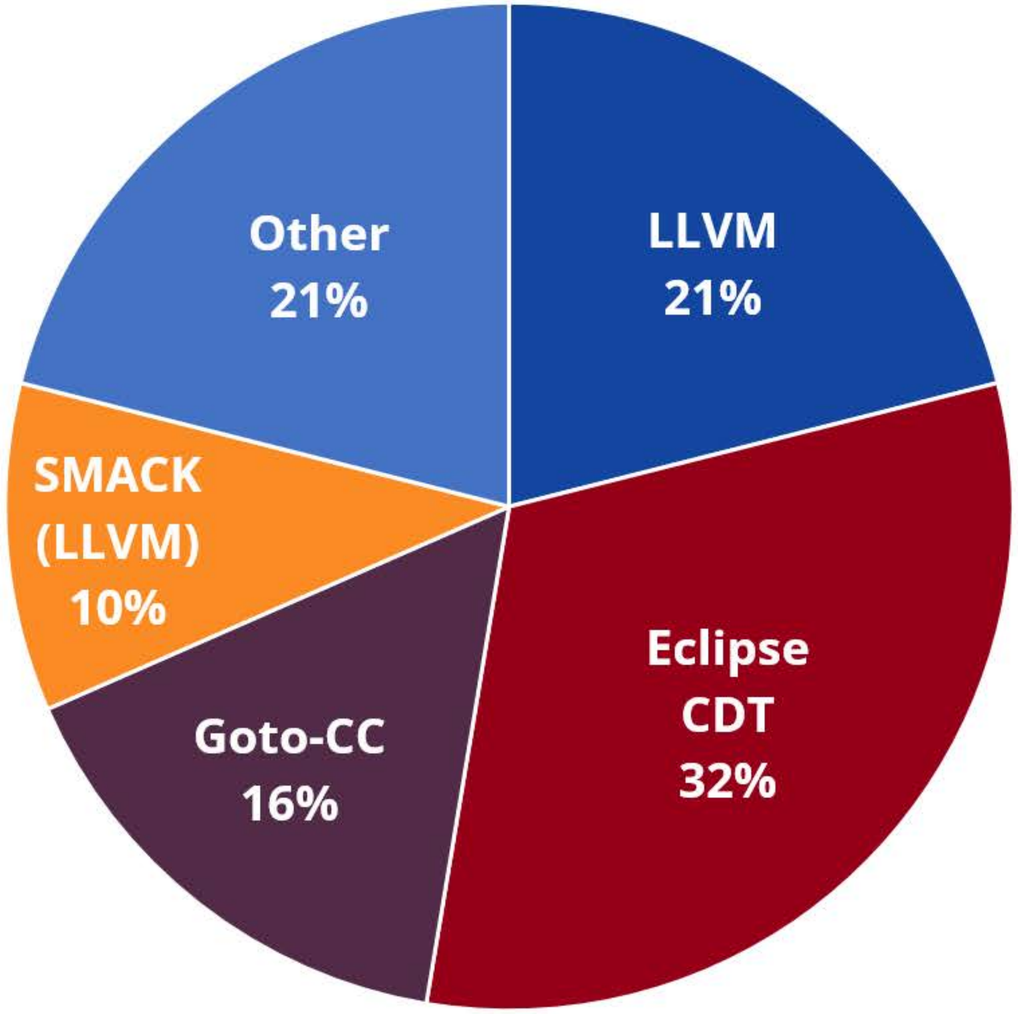
What Frontends do Tools Use?



No best framework?

Source: SV-COMP 2021

What Frontends do Tools Use?



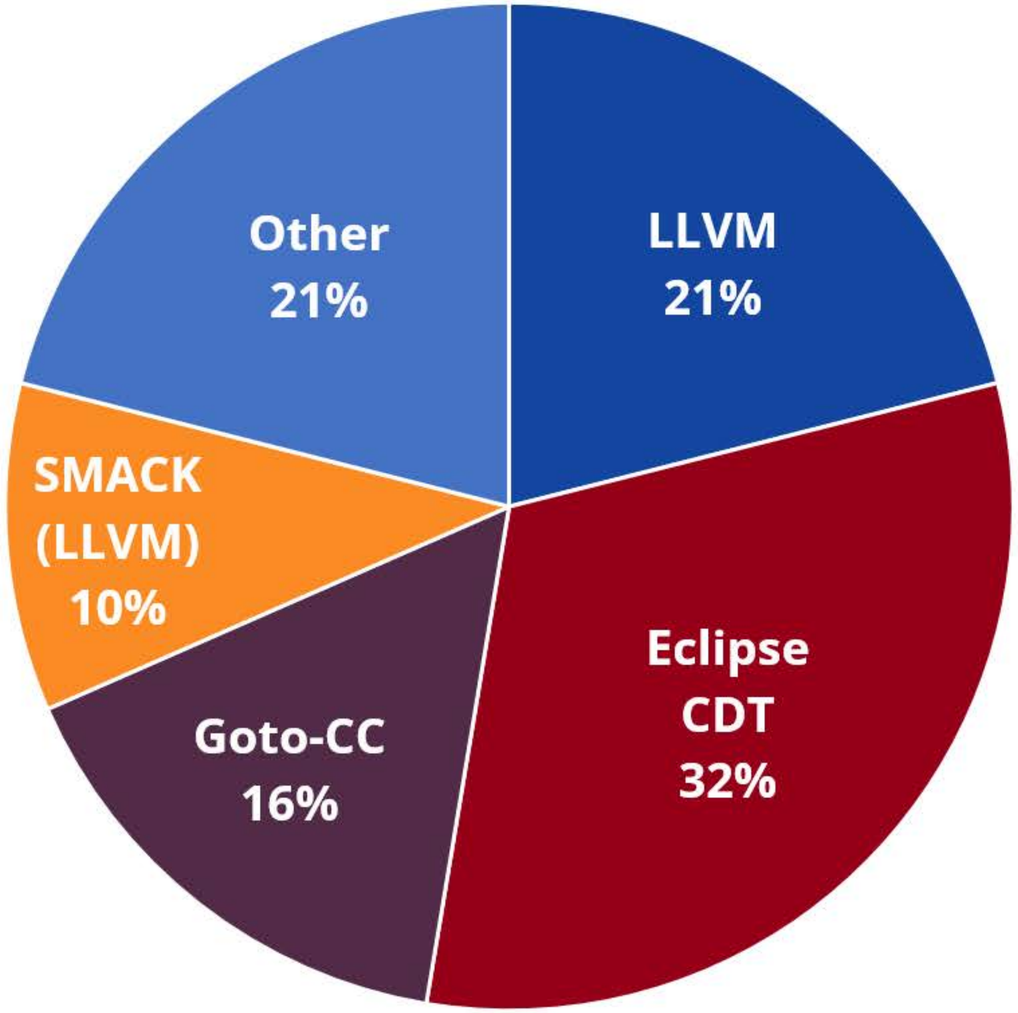
No best framework?

LLVM (+SMACK)

- ✓ Easy to integrate
- ✓ Decades of development

Source: SV-COMP 2021

What Frontends do Tools Use?

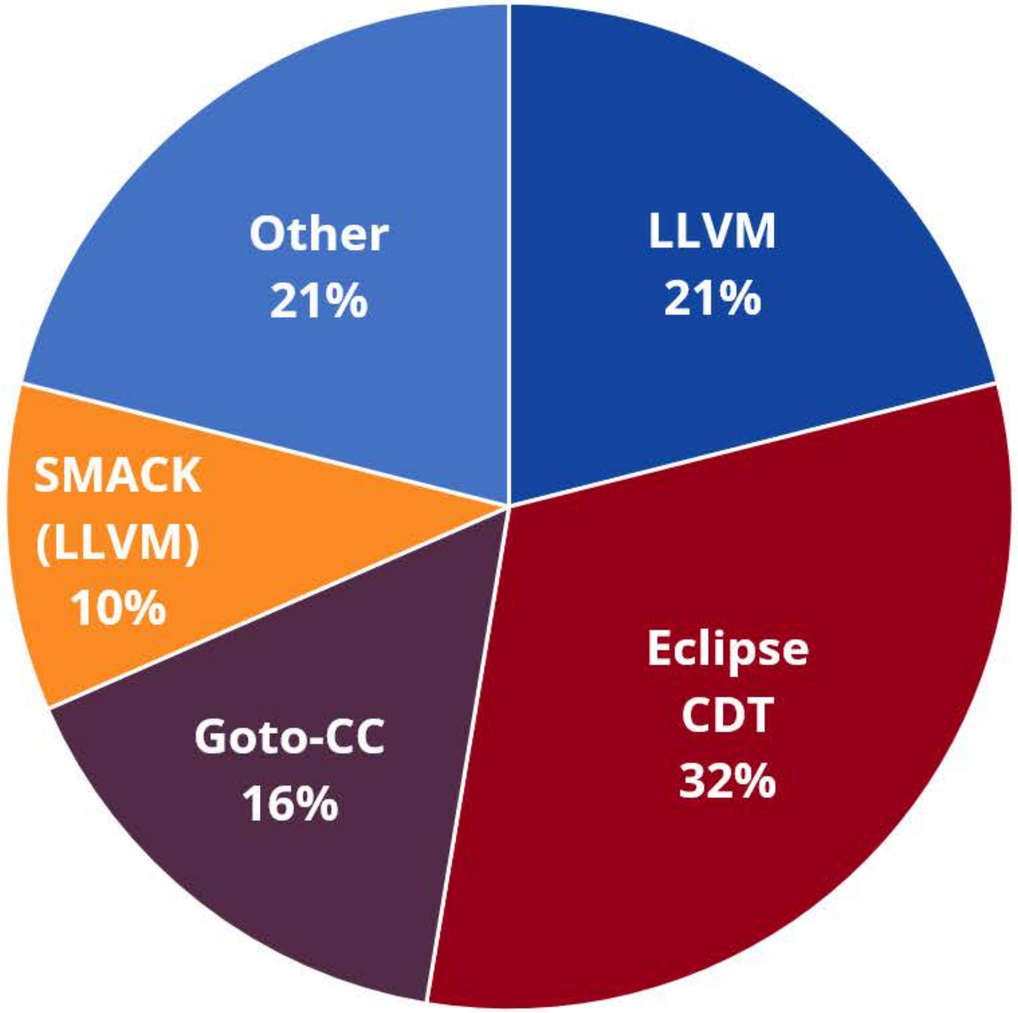


No best framework?

LLVM (+SMACK)	Eclipse CDT
✓Easy to integrate ✓Decades of development	✓Low-level access to syntax ✓Widely used

Source: SV-COMP 2021

What Frontends do Tools Use?



No best framework?

LLVM (+SMACK)	Eclipse CDT	Goto-CC
✓Easy to integrate ✓Decades of development	✓Low-level access to syntax ✓Widely used	✓Verification-centric ✓Proven to work well (CBMC)

Source: SV-COMP 2021

What Frontends do Tools Use?

Source: SV-COMP 2021

What Frontends do Tools Use?

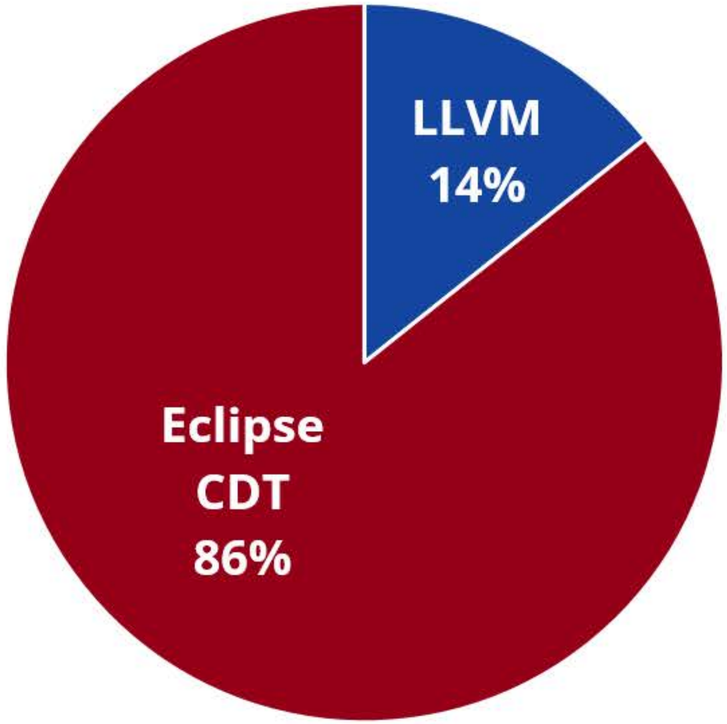
Uses CEGAR

No CEGAR

Source: SV-COMP 2021

What Frontends do Tools Use?

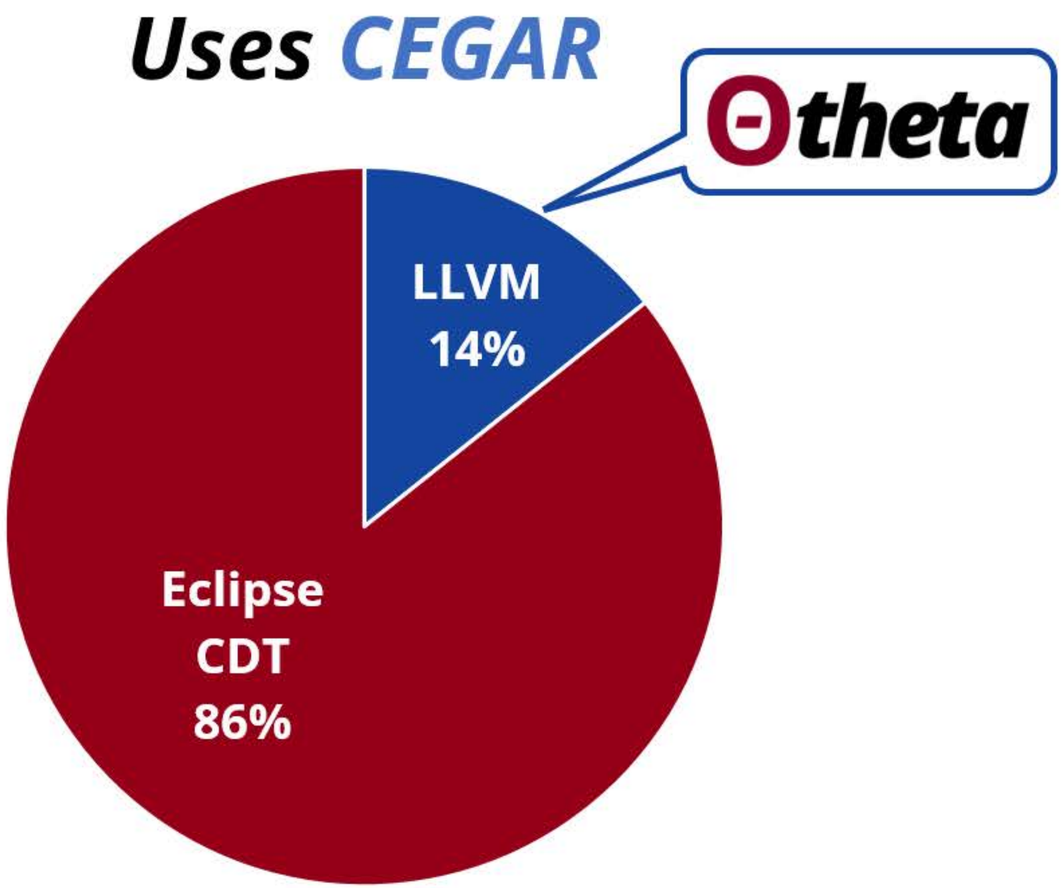
Uses CEGAR



No CEGAR

Source: SV-COMP 2021

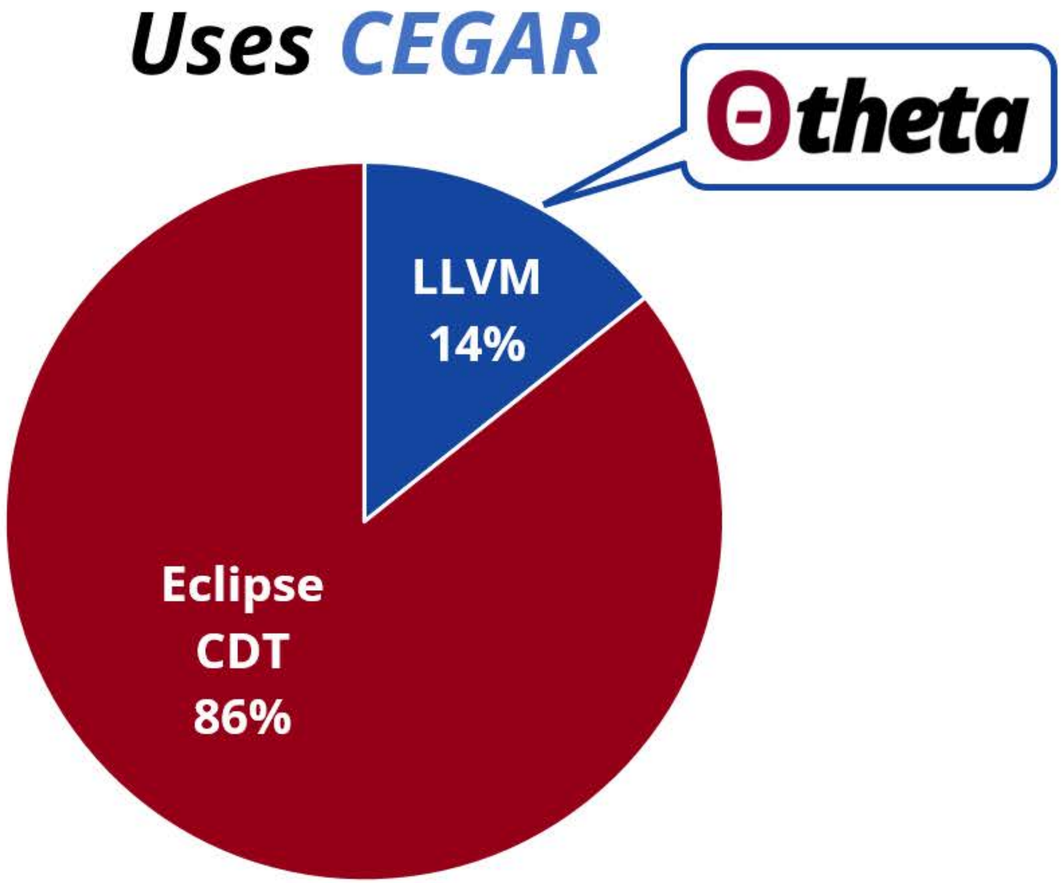
What Frontends do Tools Use?

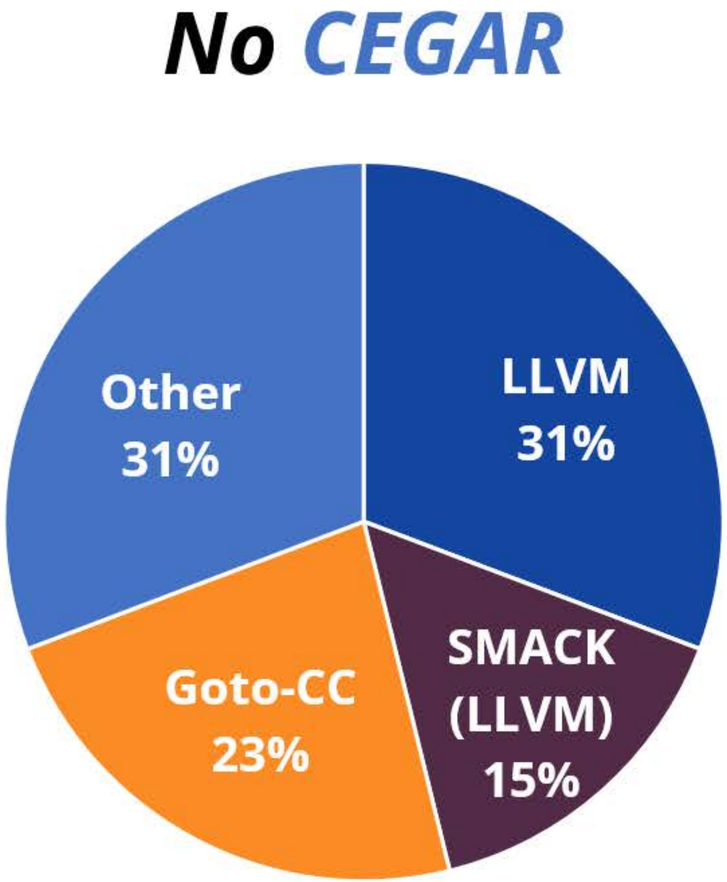


No CEGAR

Source: SV-COMP 2021

What Frontends do Tools Use?





Source: SV-COMP 2021

Software Model Checking Overview

Software Model Checking Overview

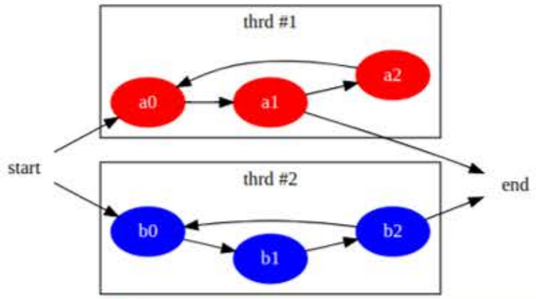
```
18  int main(int argc, char** argv){
19      int i = 0;
20      char* mode      = DEFAULT_MODE;
21      char* client    = DEFAULT_CLIENT;
22      directory       = DEFAULT_DIRECTORY
23      keyfiles        = DEFAULT_KEYS;
24      serverip        = DEFAULT_SERVER;
25      portnum         = DEFAULT_PORT;
26      connection      = DEFAULT_CONNECTIC
27      partner         = DEFAULT_PARTNER;
28
29
30
31
32
33
```

Program source

- High-Level
- Non-formal

Software Model Checking Overview

```
18  int main(int argc, char** argv){
19      int i = 0;
20      char* mode      = DEFAULT_MODE;
21      char* client    = DEFAULT_CLIENT;
22      directory       = DEFAULT_DIRECTORY
23      keyfiles        = DEFAULT_KEYS;
24      serverip        = DEFAULT_SERVER;
25      portnum         = DEFAULT_PORT;
26      connection      = DEFAULT_CONNECTIC
27      partner         = DEFAULT_PARTNER;
28
29
30
31
32
33
```



Program source

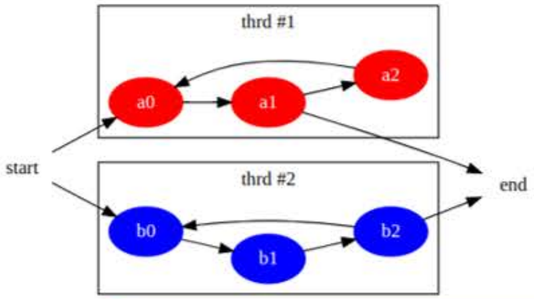
- High-Level
- Non-formal

Formal model

- Low-level
- Precise

Software Model Checking Overview

```
18  int main(int argc, char** argv){
19      int i = 0;
20      char* mode      = DEFAULT_MODE;
21      char* client    = DEFAULT_CLIENT;
22      directory       = DEFAULT_DIRECTORY
23      keyfiles        = DEFAULT_KEYS;
24      serverip        = DEFAULT_SERVER;
25      portnum         = DEFAULT_PORT;
26      connection      = DEFAULT_CONNECTIC
27      partner         = DEFAULT_PARTNER;
28
29
30
31
32
33
```



Program source

- High-Level
- Non-formal

Formal model

- Low-level
- Precise

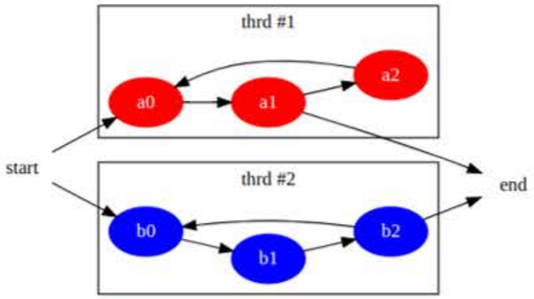
Result

- Unsafe: CEx
- Safe: Proof

Software Model Checking Overview

Frontend

```
18  int main(int argc, char** argv){
19      int i = 0;
20      char* mode      = DEFAULT_MODE;
21      char* client    = DEFAULT_CLIENT;
22      directory       = DEFAULT_DIRECTORY
23      keyfiles        = DEFAULT_KEYS;
24      serverip        = DEFAULT_SERVER;
25      portnum         = DEFAULT_PORT;
26      connection      = DEFAULT_CONNECTIC
27      partner         = DEFAULT_PARTNER;
28
29
30
31
32
33
```



Program source

- High-Level
- Non-formal

Formal model

- Low-level
- Precise

Result

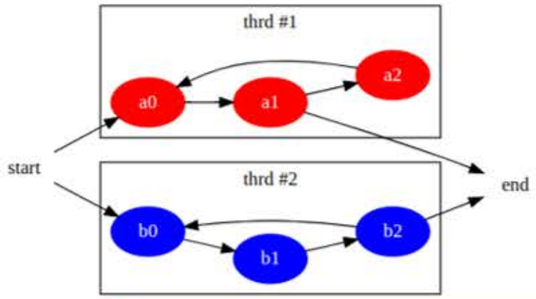
- Unsafe: CEx
- Safe: Proof

Software Model Checking Overview

Frontend

```
18  int main(int argc, char** argv){
19      int i = 0;
20      char* mode      = DEFAULT_MODE;
21      char* client    = DEFAULT_CLIENT;
22      directory       = DEFAULT_DIRECTORY
23      keyfiles        = DEFAULT_KEYS;
24      serverip        = DEFAULT_SERVER;
25      portnum         = DEFAULT_PORT;
26      connection      = DEFAULT_CONNECTIC
27      partner         = DEFAULT_PARTNER;
28
29
30
31
32
33
```

Analysis



Program source

- High-Level
- Non-formal

Formal model

- Low-level
- Precise

Result

- Unsafe: CEx
- Safe: Proof

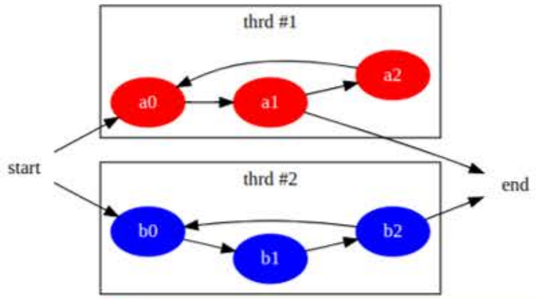
+semantics

Software Model Checking Overview

Frontend

Analysis

```
18  int main(int argc, char** argv){
19      int i = 0;
20      char* mode      = DEFAULT_MODE;
21      char* client    = DEFAULT_CLIENT;
22      directory       = DEFAULT_DIRECTORY
23      keyfiles        = DEFAULT_KEYS;
24      serverip        = DEFAULT_SERVER;
25      portnum         = DEFAULT_PORT;
26      connection      = DEFAULT_CONNECTIC
27      partner         = DEFAULT_PARTNER;
28
29
30
31
32
33
```



Program source

- High-Level
- Non-formal

Formal model

- Low-level
- Precise

Result

- Unsafe: CEx
- Safe: Proof

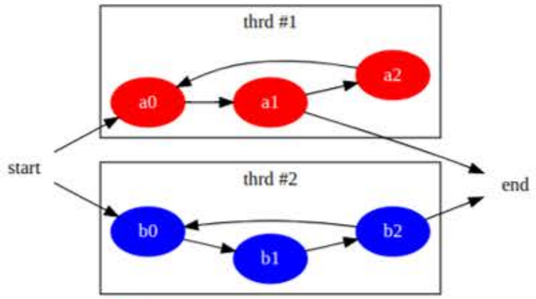
+semantics

- **SMT-based**

Software Model Checking Overview

Frontend

```
18  int main(int argc, char** argv){
19      int i = 0;
20      char* mode      = DEFAULT_MODE;
21      char* client    = DEFAULT_CLIENT;
22      directory       = DEFAULT_DIRECTORY
23      keyfiles        = DEFAULT_KEYS;
24      serverip        = DEFAULT_SERVER;
25      portnum         = DEFAULT_PORT;
26      connection      = DEFAULT_CONNECTIC
27      partner         = DEFAULT_PARTNER;
28
29
30
31
32
33
```



Analysis

Program source

- High-Level
- Non-formal

Formal model

- Low-level
- Precise

Result

- Unsafe: CEx
- Safe: Proof

+semantics

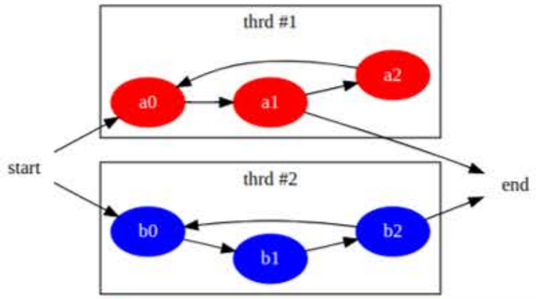
- **SMT-based**
- Stateless

Software Model Checking Overview

Frontend

Analysis

```
18  int main(int argc, char** argv){
19      int i = 0;
20      char* mode      = DEFAULT_MODE;
21      char* client    = DEFAULT_CLIENT;
22      directory       = DEFAULT_DIRECTORY
23      keyfiles        = DEFAULT_KEYS;
24      serverip        = DEFAULT_SERVER;
25      portnum         = DEFAULT_PORT;
26      connection      = DEFAULT_CONNECTIC
27      partner         = DEFAULT_PARTNER;
28
29
30
31
32
33
```



Program source

- High-Level
- Non-formal

Formal model

- Low-level
- Precise

Result

- Unsafe: CEx
- Safe: Proof

+semantics

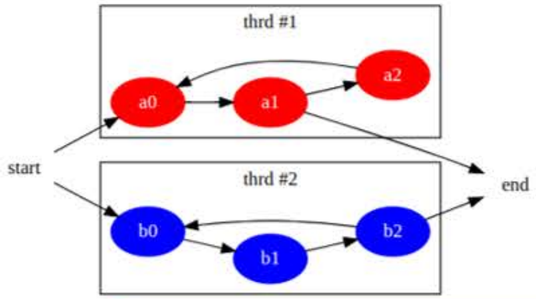
- **SMT-based**
- Stateless
- Symbolic execution

Software Model Checking Overview

Frontend

Analysis

```
18  int main(int argc, char** argv){
19      int i = 0;
20      char* mode      = DEFAULT_MODE;
21      char* client    = DEFAULT_CLIENT;
22      directory       = DEFAULT_DIRECTORY
23      keyfiles        = DEFAULT_KEYS;
24      serverip        = DEFAULT_SERVER;
25      portnum         = DEFAULT_PORT;
26      connection      = DEFAULT_CONNECTIC
27      partner         = DEFAULT_PARTNER;
28
29
30
31
32
33
```



Program source

- High-Level
- Non-formal

Formal model

- Low-level
- Precise

Result

- Unsafe: CEx
- Safe: Proof

+semantics

- **SMT-based**
- Stateless
- Symbolic execution
- ....

SMT-Based Model Checking

SMT-Based Model Checking

Bounded Model Checking (**BMC**)

Counterexample-Guided Abstraction Refinement (**CEGAR**)

SMT-Based Model Checking

Bounded Model Checking (**BMC**)

Counterexample-Guided Abstraction Refinement (**CEGAR**)

- Up to a bound **k**:

SMT-Based Model Checking

Bounded Model Checking (**BMC**)

Counterexample-Guided Abstraction Refinement (**CEGAR**)

- Up to a bound **k**:
  - Traverse all paths

SMT-Based Model Checking

Bounded Model Checking (**BMC**)

Counterexample-Guided Abstraction Refinement (**CEGAR**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query: φ_P

SMT-Based Model Checking

Bounded Model Checking (**BMC**)

Counterexample-Guided Abstraction Refinement (**CEGAR**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query: $\boldsymbol{\varphi_P}$
  - Encode error property: $\boldsymbol{\varphi_{Err}}$

SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query: $\boldsymbol{\varphi_P}$
  - Encode error property: $\boldsymbol{\varphi_{Err}}$
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

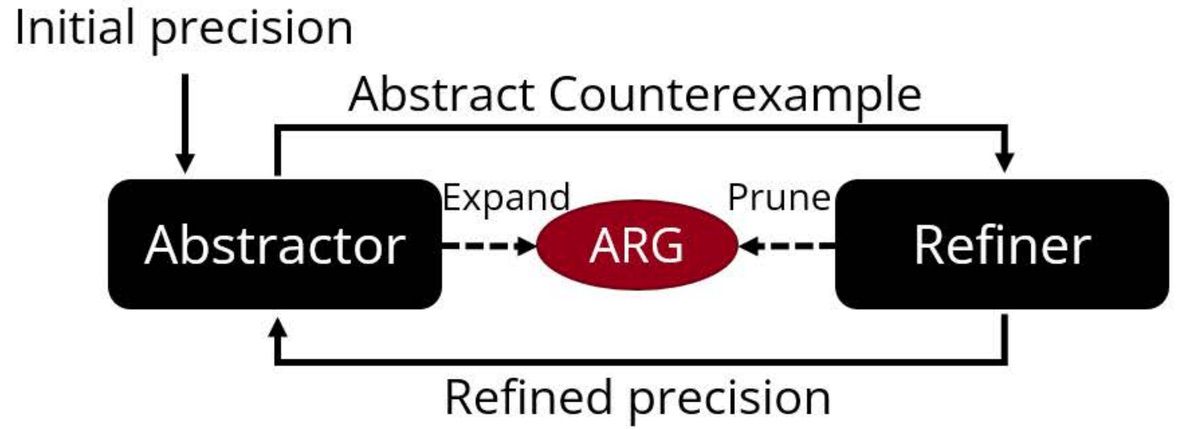
Counterexample-Guided Abstraction Refinement (**CEGAR**)

SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query: $\boldsymbol{\varphi_P}$
  - Encode error property: $\boldsymbol{\varphi_{Err}}$
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

Counterexample-Guided Abstraction Refinement (**CEGAR**)

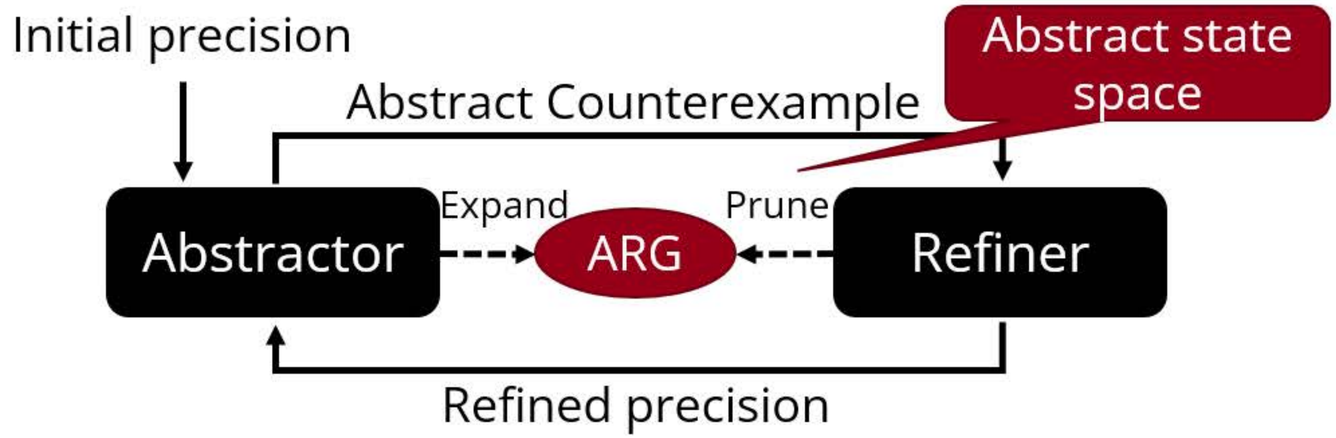


SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query: $\boldsymbol{\varphi_P}$
  - Encode error property: $\boldsymbol{\varphi_{Err}}$
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

Counterexample-Guided Abstraction Refinement (**CEGAR**)

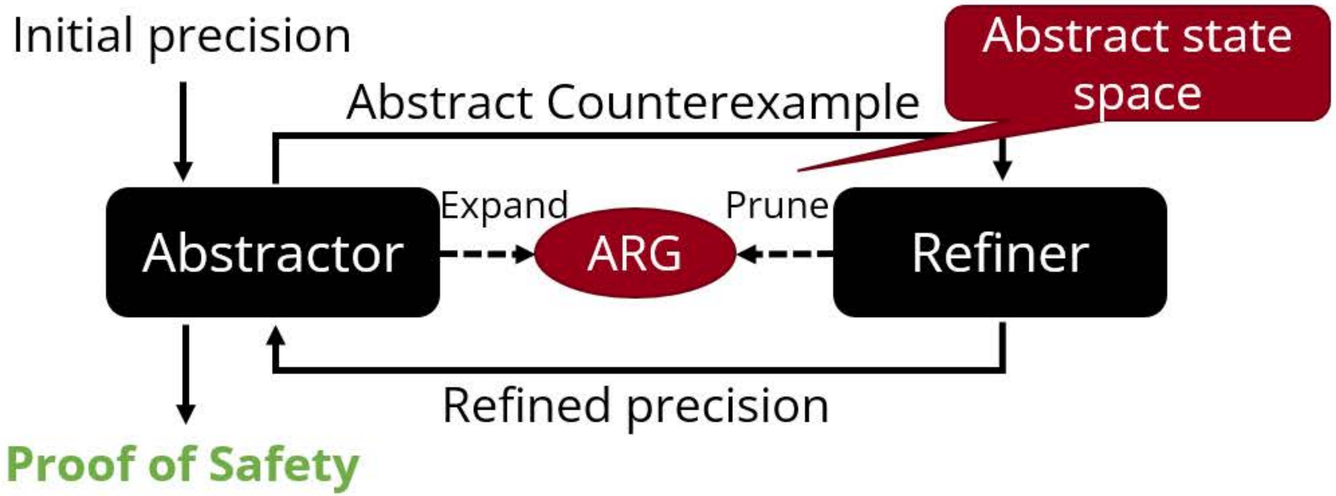


SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query: φ_P
  - Encode error property: φ_{Err}
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

Counterexample-Guided Abstraction Refinement (**CEGAR**)

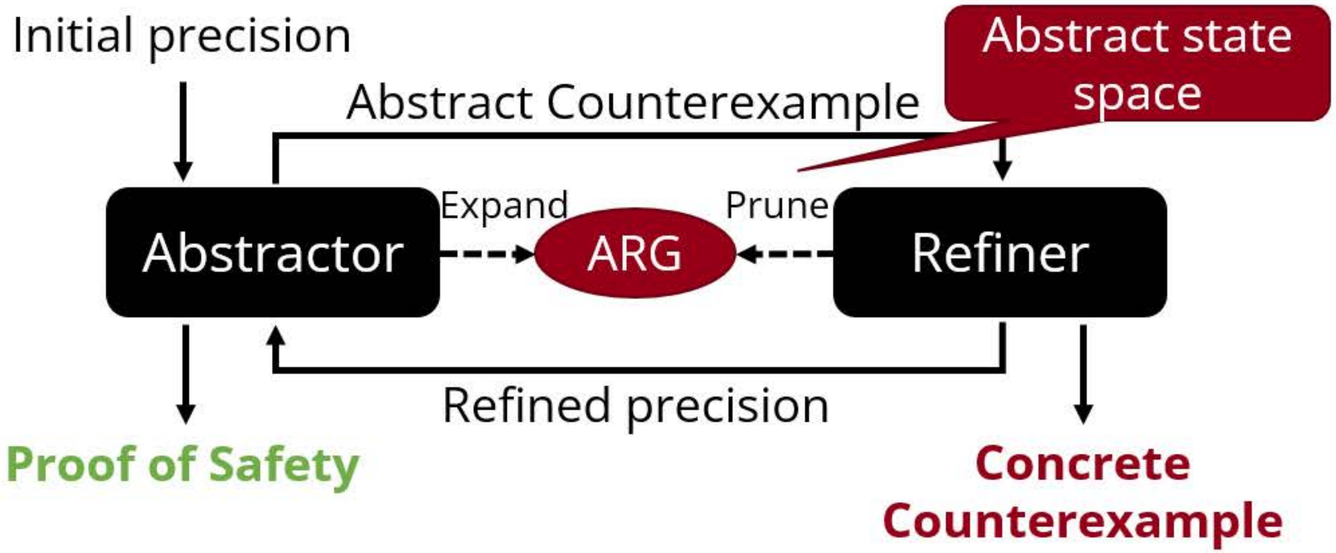


SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query: φ_P
  - Encode error property: φ_{Err}
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

Counterexample-Guided Abstraction Refinement (**CEGAR**)



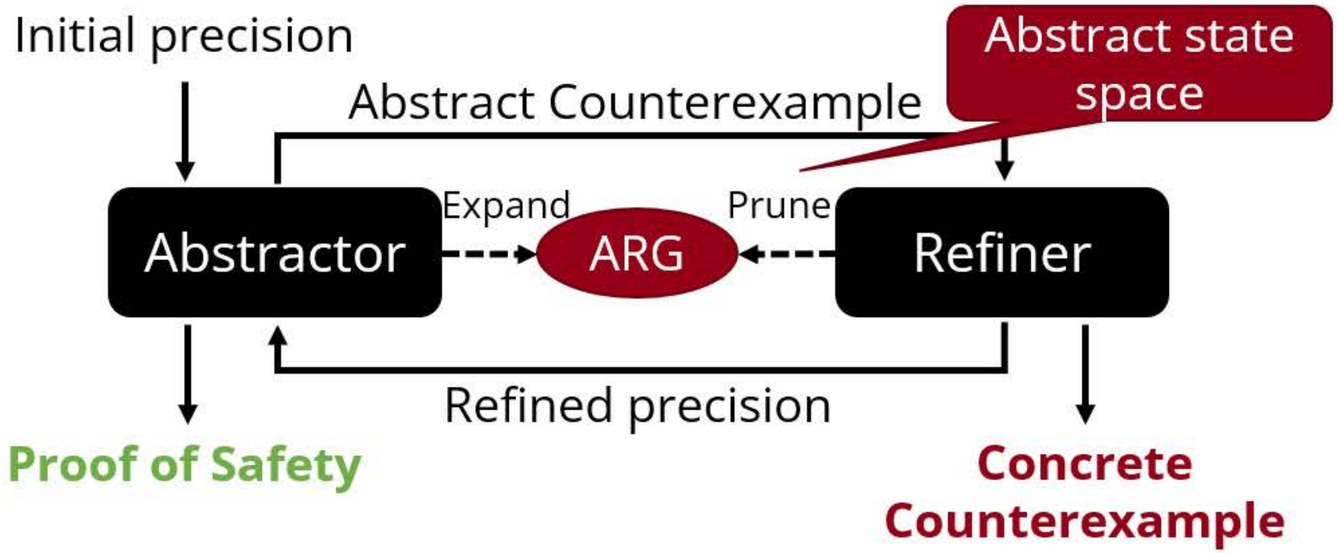
SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query: φ_P
  - Encode error property: φ_{Err}
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

Encodes the program directly

Counterexample-Guided Abstraction Refinement (**CEGAR**)



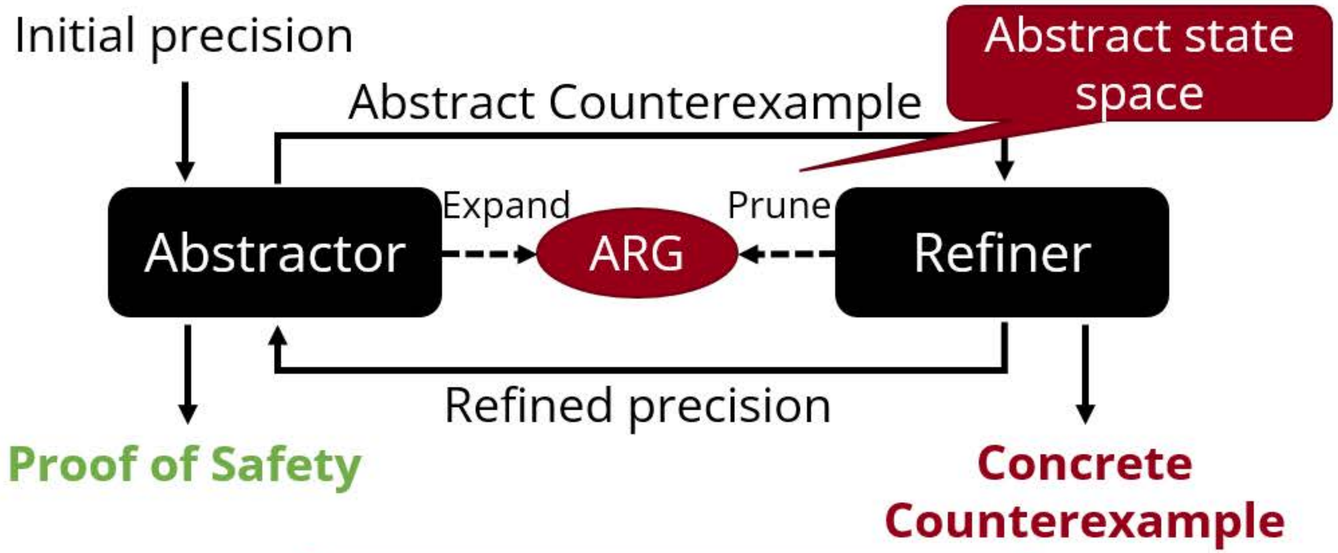
SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query: φ_P
  - Encode error property: φ_{Err}
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

Encodes the program directly

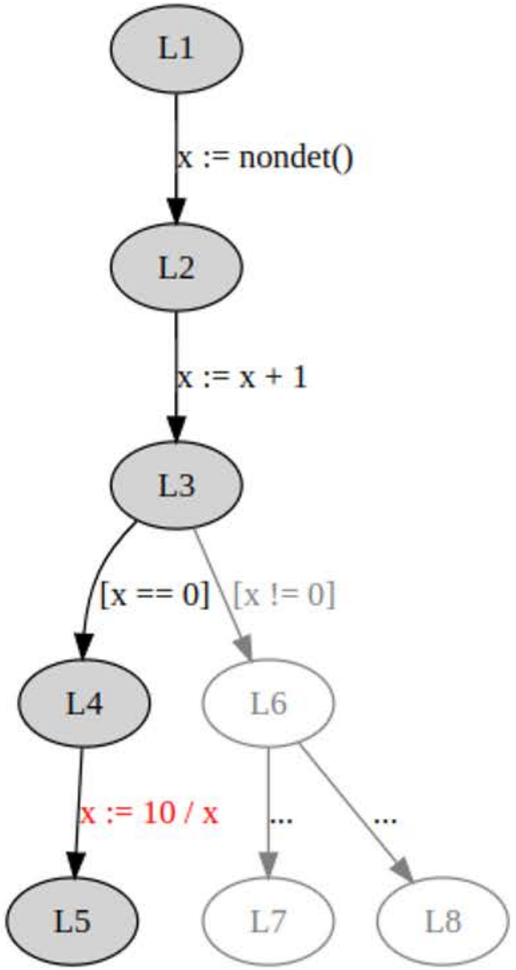
Counterexample-Guided Abstraction Refinement (**CEGAR**)



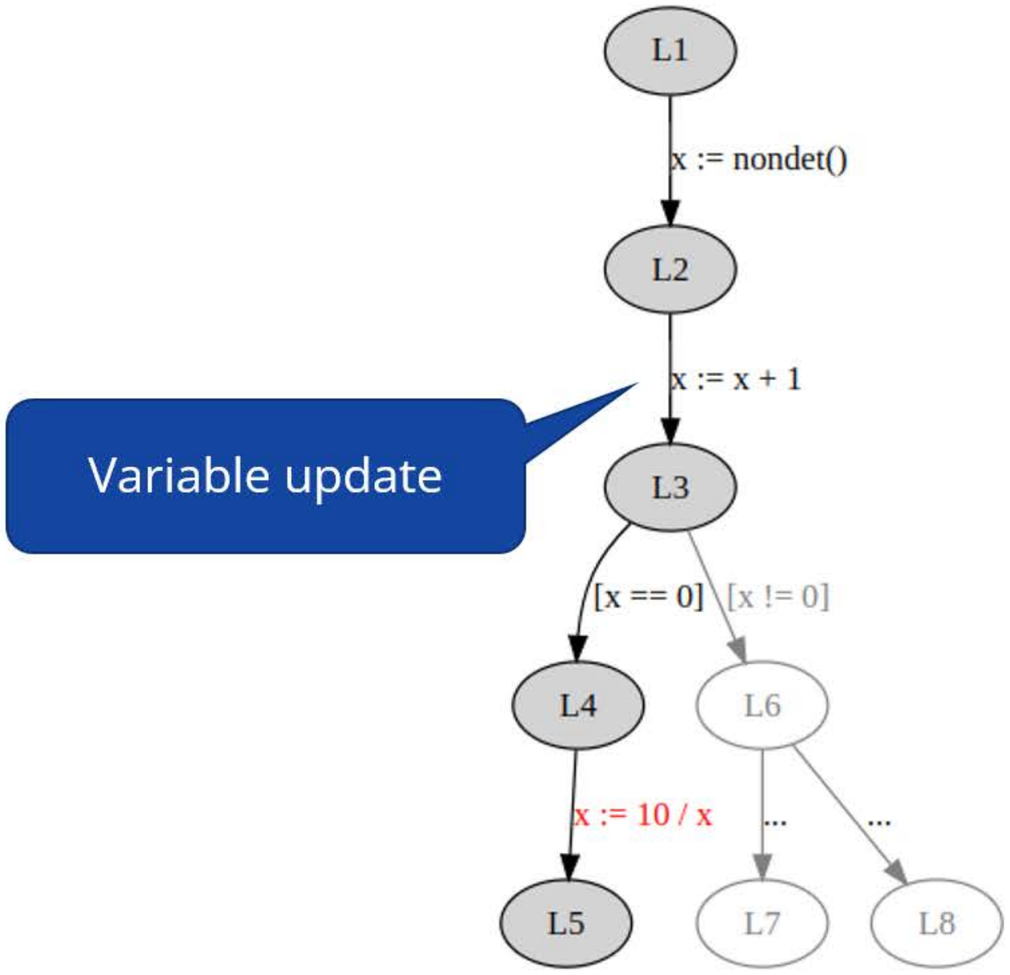
Encodes an abstract trace

Encoding Traces as SMT-formulae

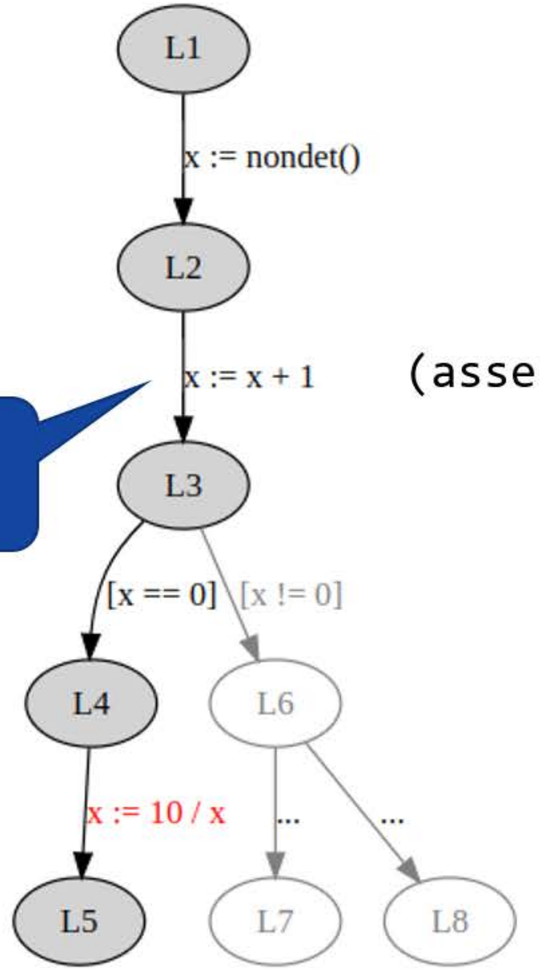
Encoding Traces as SMT-formulae



Encoding Traces as SMT-formulae



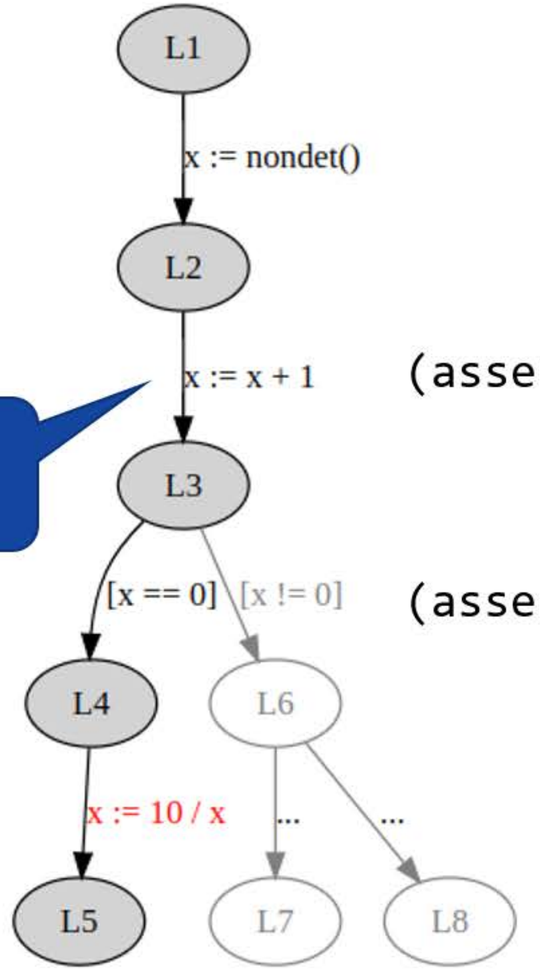
Encoding Traces as SMT-formulae



Variable update

```
(assert (= x1 (+ x0 1)))
```

Encoding Traces as SMT-formulae

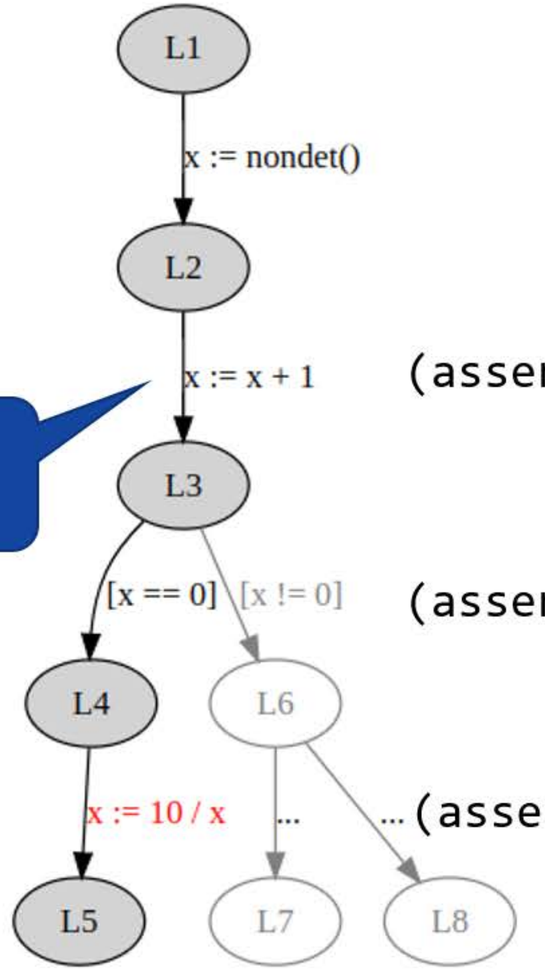


Variable update

```
(assert (= x1 (+ x0 1)))
```

```
(assert (= x1 10))
```

Encoding Traces as SMT-formulae



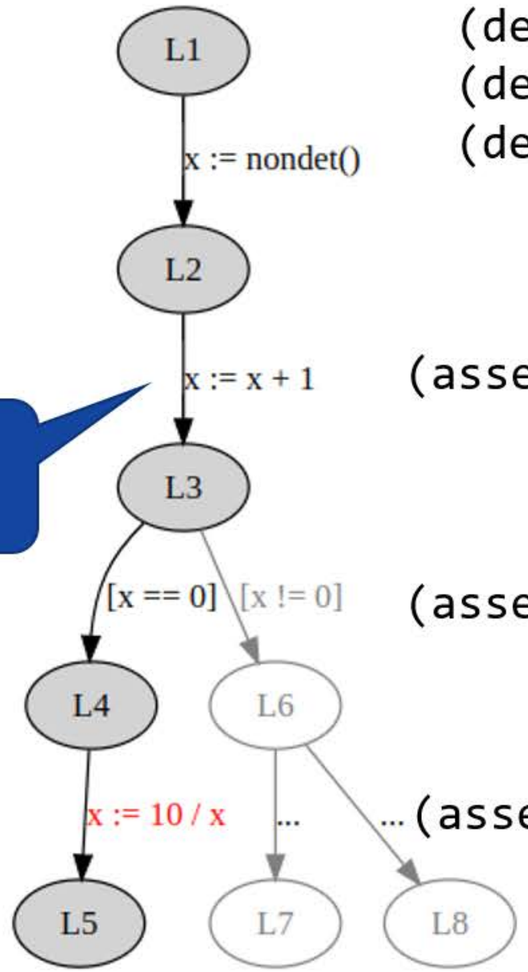
Variable update

```
(assert (= x1 (+ x0 1)))
```

```
(assert (= x1 10))
```

```
(assert (= x2 (/ 10 x1)))
```

Encoding Traces as SMT-formulae



Variable update

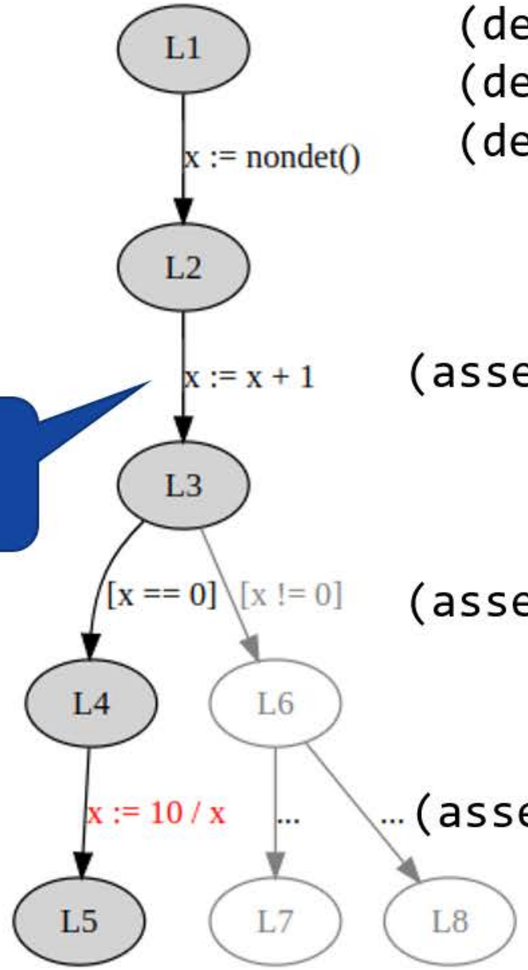
```
; Variable declarations
(declare-fun x0 () Int)
(declare-fun x1 () Int)
(declare-fun x2 () Int)
```

```
(assert (= x1 (+ x0 1)))
```

```
(assert (= x1 10))
```

```
(assert (= x2 (/ 10 x1)))
```

Encoding Traces as SMT-formulae




```
; Variable declarations
(declare-fun x0 () Int)
(declare-fun x1 () Int)
(declare-fun x2 () Int)
```

Multiple constants

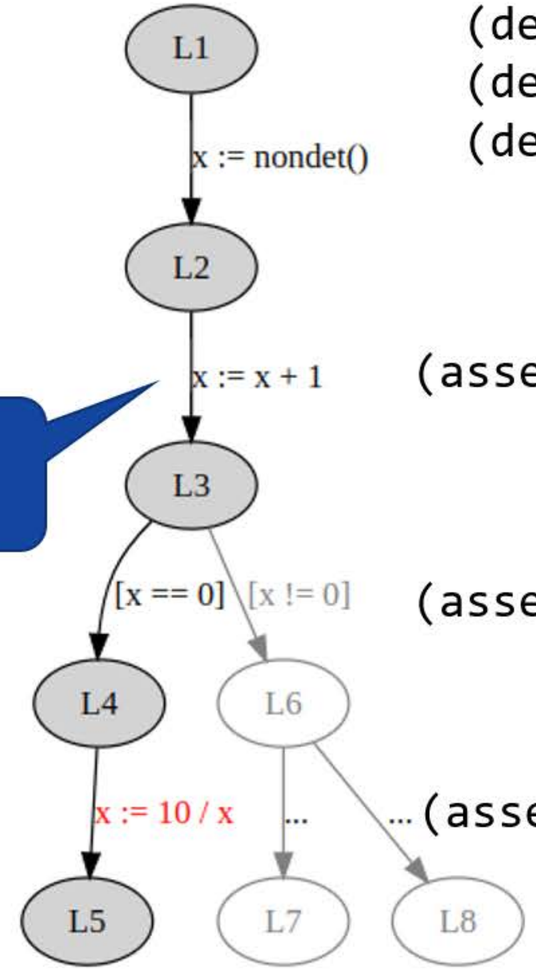
```
(assert (= x1 (+ x0 1)))
```

Variable update

```
(assert (= x1 10))
```

```
(assert (= x2 (/ 10 x1)))
```

Encoding Traces as SMT-formulae




```
; Variable declarations
(declare-fun x0 () Int)
(declare-fun x1 () Int)
(declare-fun x2 () Int)
```

Multiple constants

Variable update

```
(assert (= x1 (+ x0 1)))
```

```
(assert (= x1 10))
```

```
(assert (= x2 (/ 10 x1)))
```

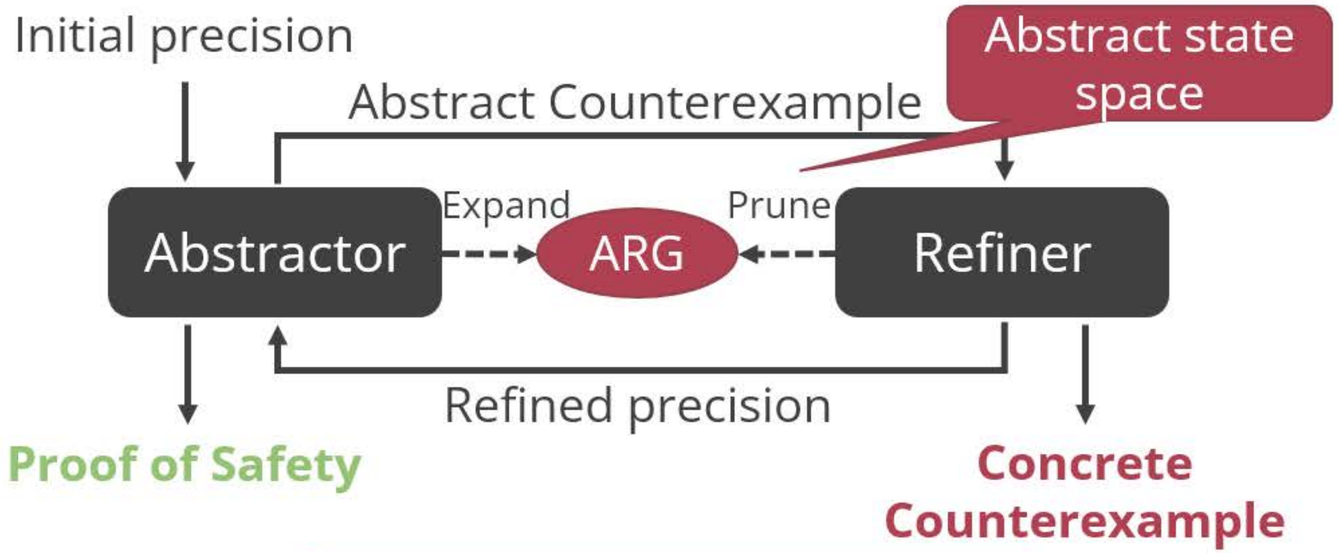
Static Single Assignment (SSA) form

SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query φ_P
  - Encode error property: φ_{Err}
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

Counterexample-Guided Abstraction Refinement (**CEGAR**)



Encodes the program directly

SMT

Encodes an abstract trace

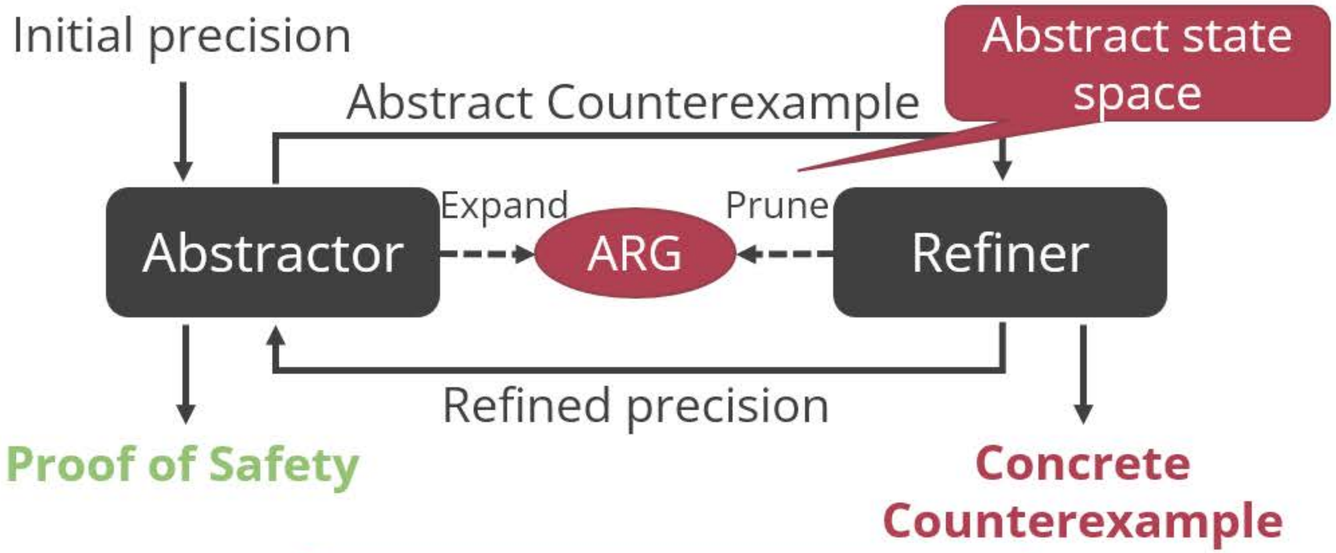
SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query φ_P
  - Encode error property: φ_{Err}
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

SSA

Counterexample-Guided Abstraction Refinement (**CEGAR**)



Encodes the program directly

SMT

Encodes an abstract trace

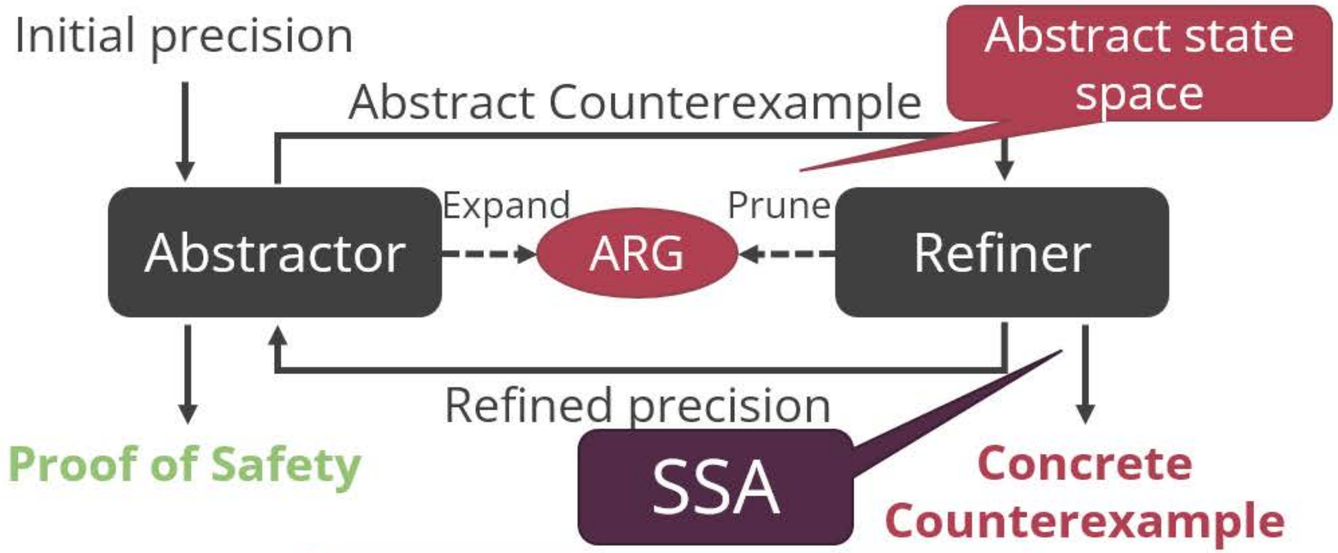
SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query φ_P
  - Encode error property: φ_{Err}
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

SSA

Counterexample-Guided Abstraction Refinement (**CEGAR**)



Encodes the program directly

SMT

Encodes an abstract trace

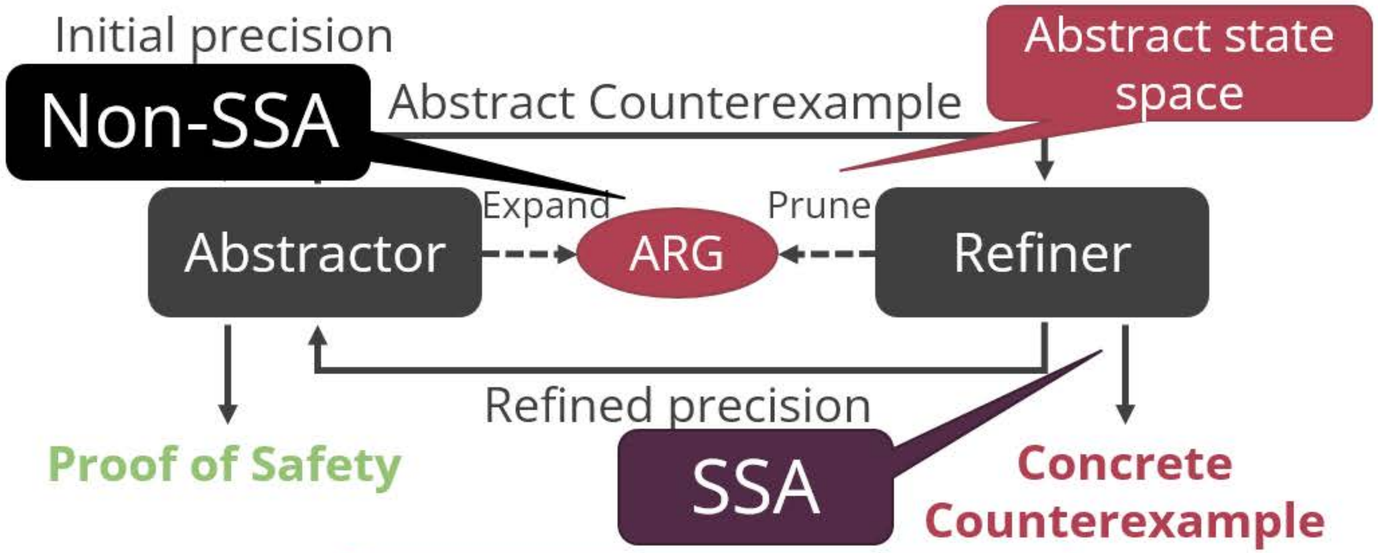
SMT-Based Model Checking

Bounded Model Checking (**BMC**)

- Up to a bound **k**:
  - Traverse all paths
  - Encode into an SMT query φ_P
  - Encode error property: φ_{Err}
  - $SAT(\varphi_P \wedge \varphi_{ERR}) \Rightarrow UNSAFE$

SSA

Counterexample-Guided Abstraction Refinement (**CEGAR**)



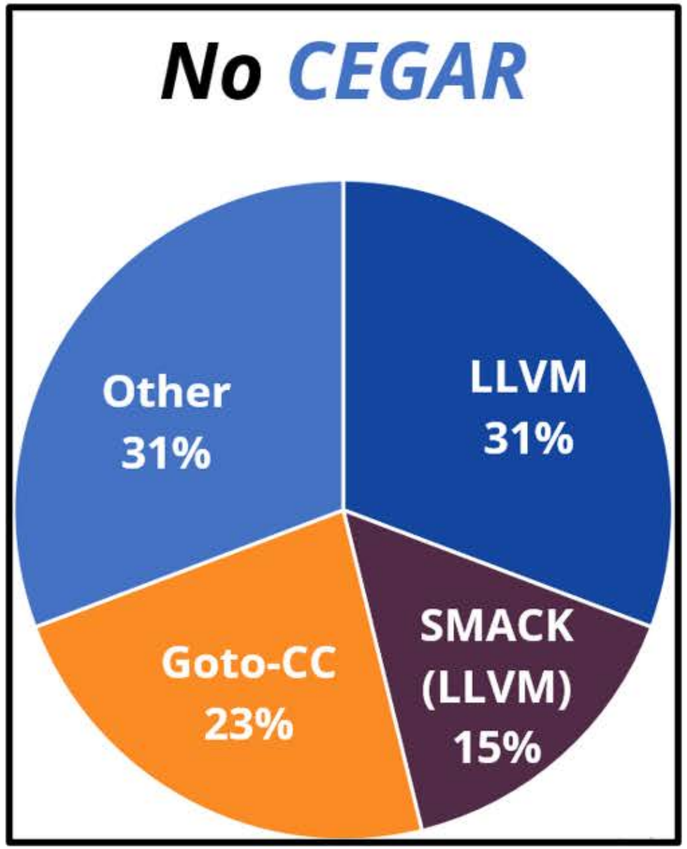
Encodes the program directly

SMT

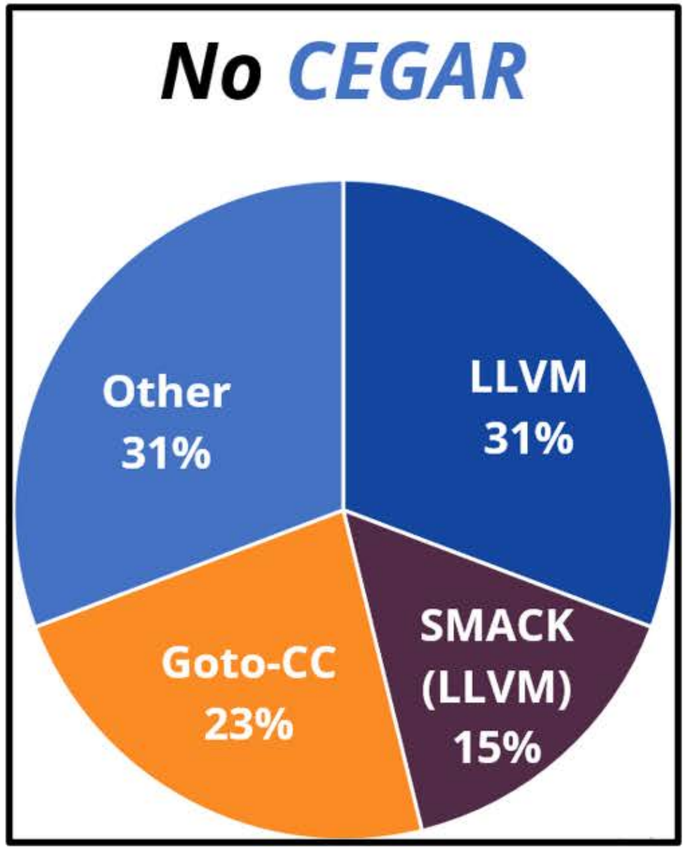
Encodes an abstract trace

LLVM

LLVM



LLVM



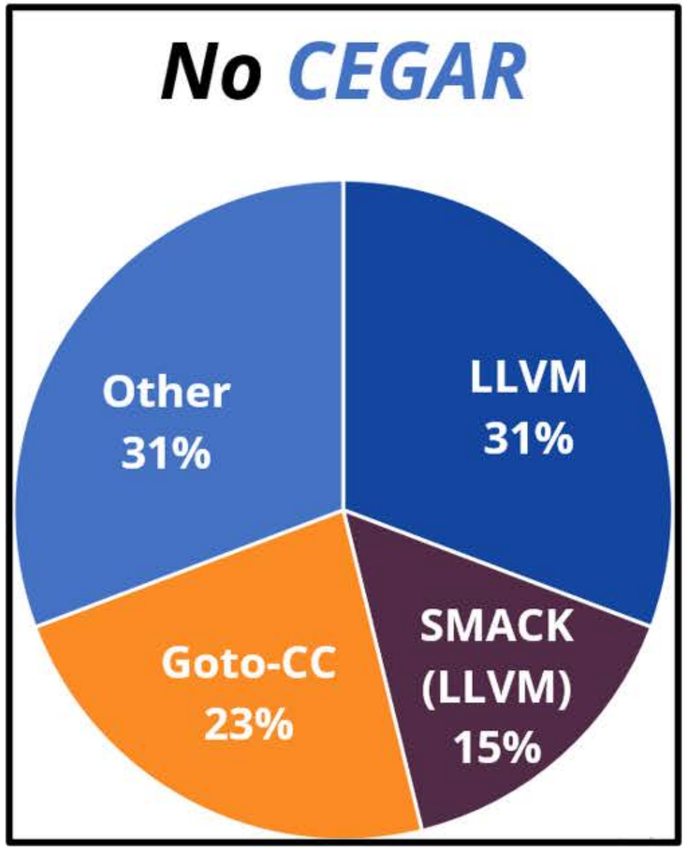
LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

LLVM began as a research project at the University of Illinois, with the goal of providing a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then,

https://llvm.org/

LLVM



LLVM Overview

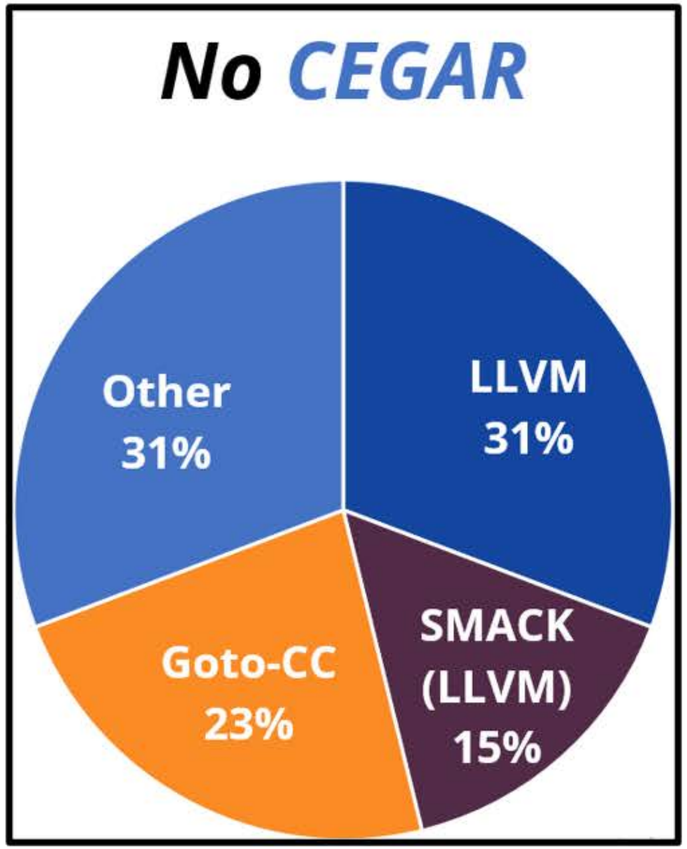
The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The [illegible] "LLVM" itself is not an acronym; it is the full name of the project.

LLVM began as a research project at the University of Illinois, with the goal of providing a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then,

https://llvm.org/

Compiler technologies (e.g. Clang) – **effective optimizations**

LLVM



LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The [illegible] "LLVM" itself is not an acronym; it is the full name of the project.

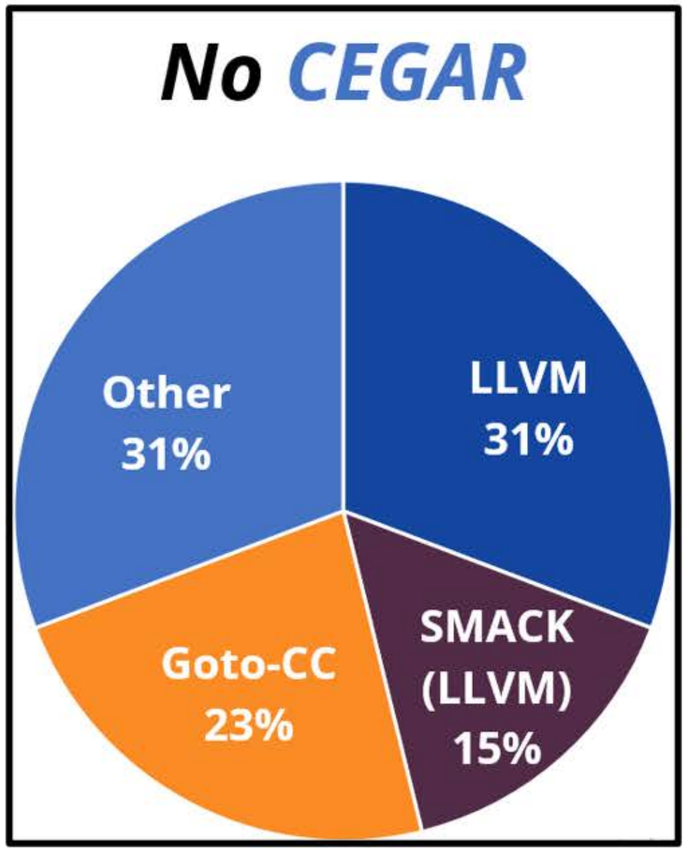
LLVM began as a research project at the University of Illinois, with the goal of providing a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then,

https://llvm.org/

Compiler technologies (e.g. Clang) – **effective optimizations**

SSA-based

LLVM



LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The [illegible] "LLVM" itself is not an acronym; it is the full name of the project.

LLVM began as a research project at the University of Illinois, with the goal of providing a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then,

https://llvm.org/

Compiler technologies (e.g. Clang) – **effective optimizations**

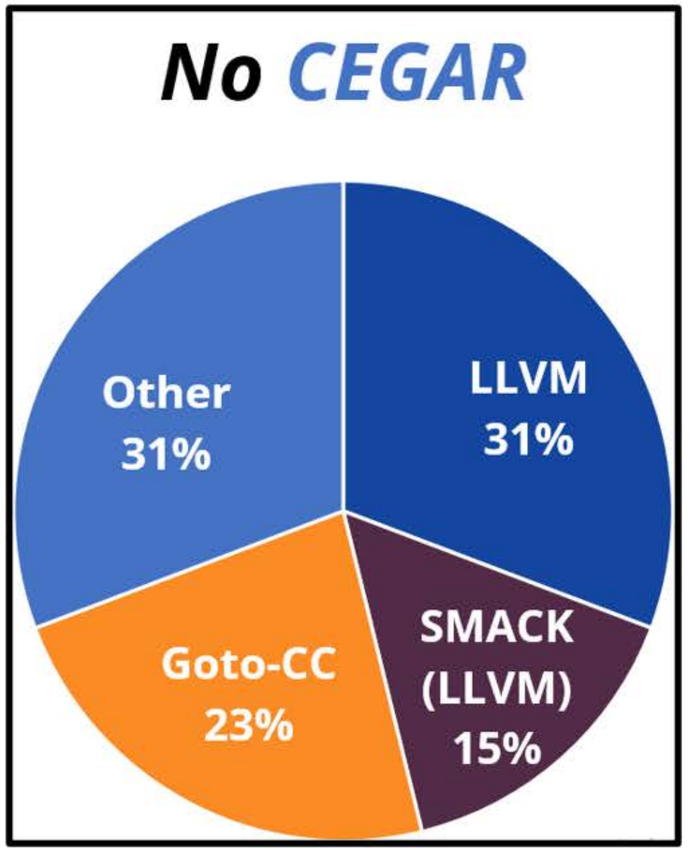
SSA-based

SSA-form

Optimizations

→ Works well for **BMC**

LLVM



LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The [illegible] "LLVM" itself is not an acronym; it is the full name of the project.

LLVM began as a research project at the University of Illinois, with the goal of providing a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then,

https://llvm.org/

Compiler technologies (e.g. Clang) – **effective optimizations**

SSA-based

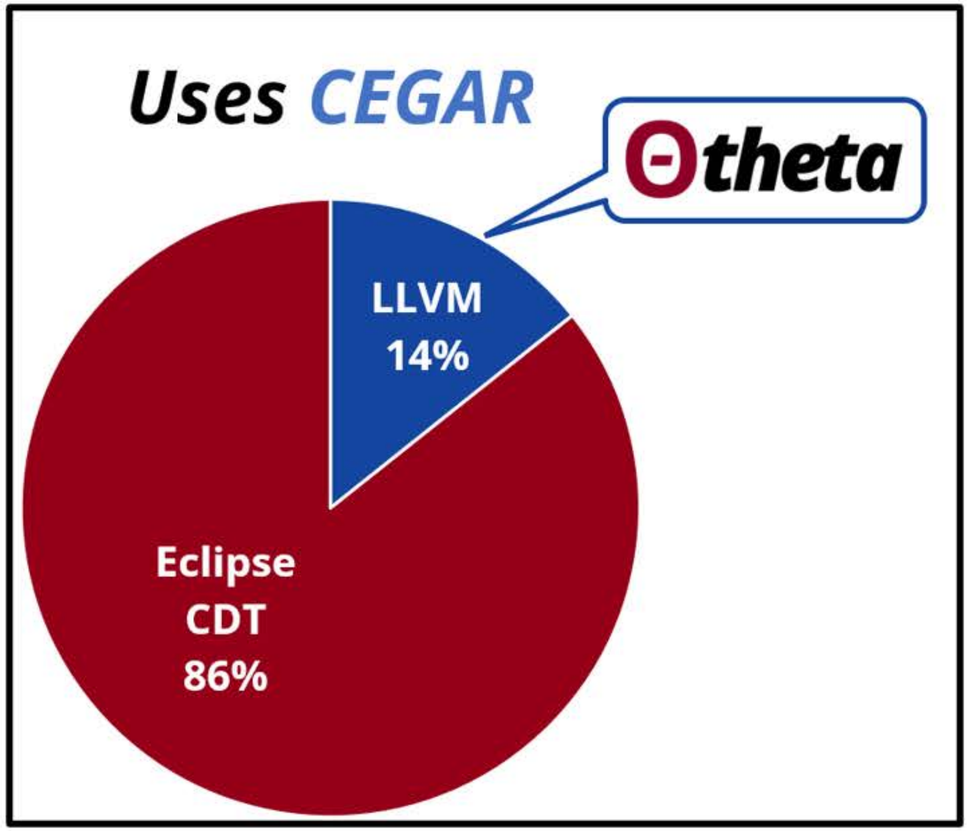
SSA-form

Optimizations

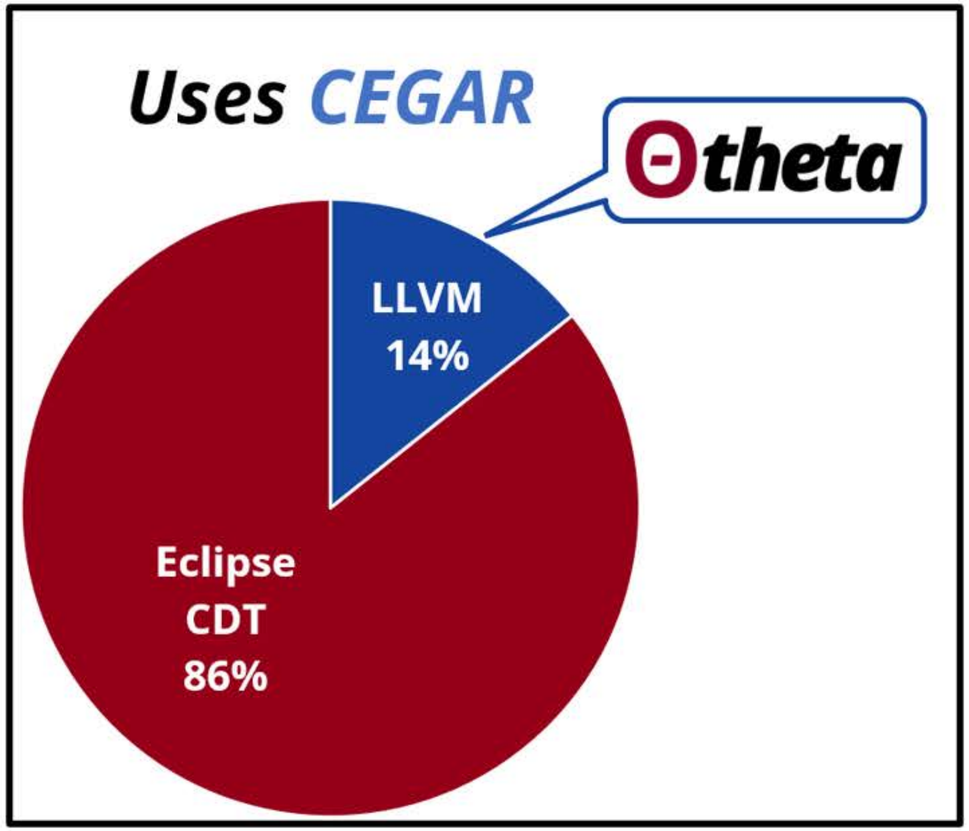
→ Works well for **BMC**

Caveat: Optimizations work **over SSA**

LLVM

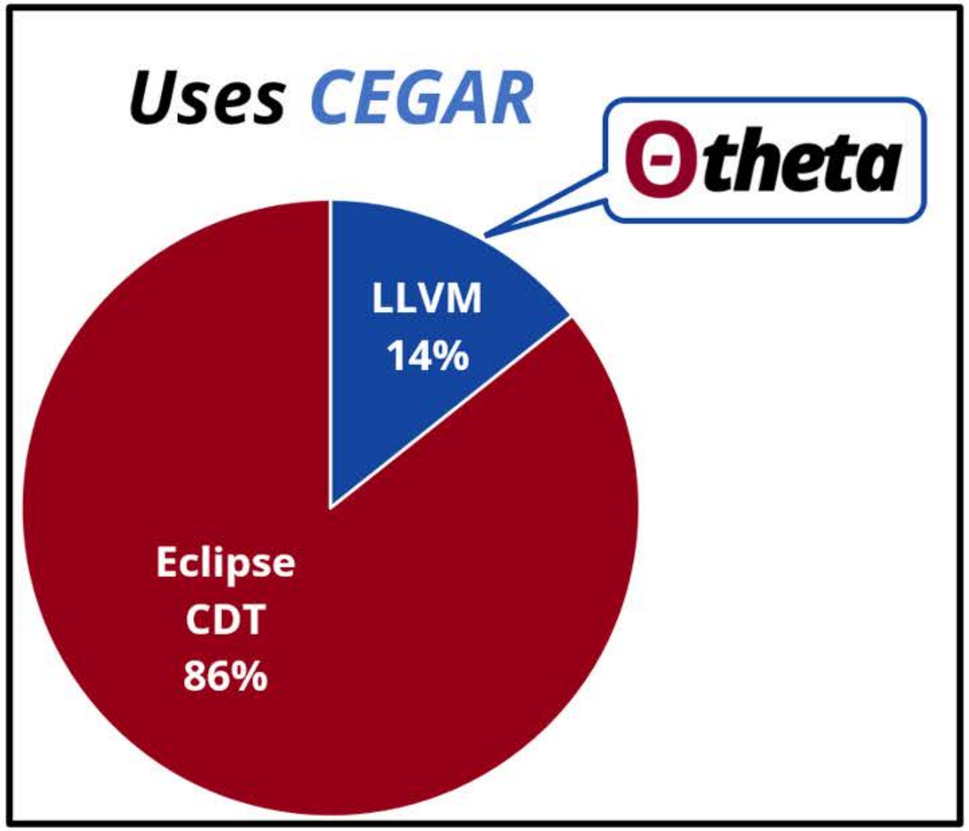


LLVM



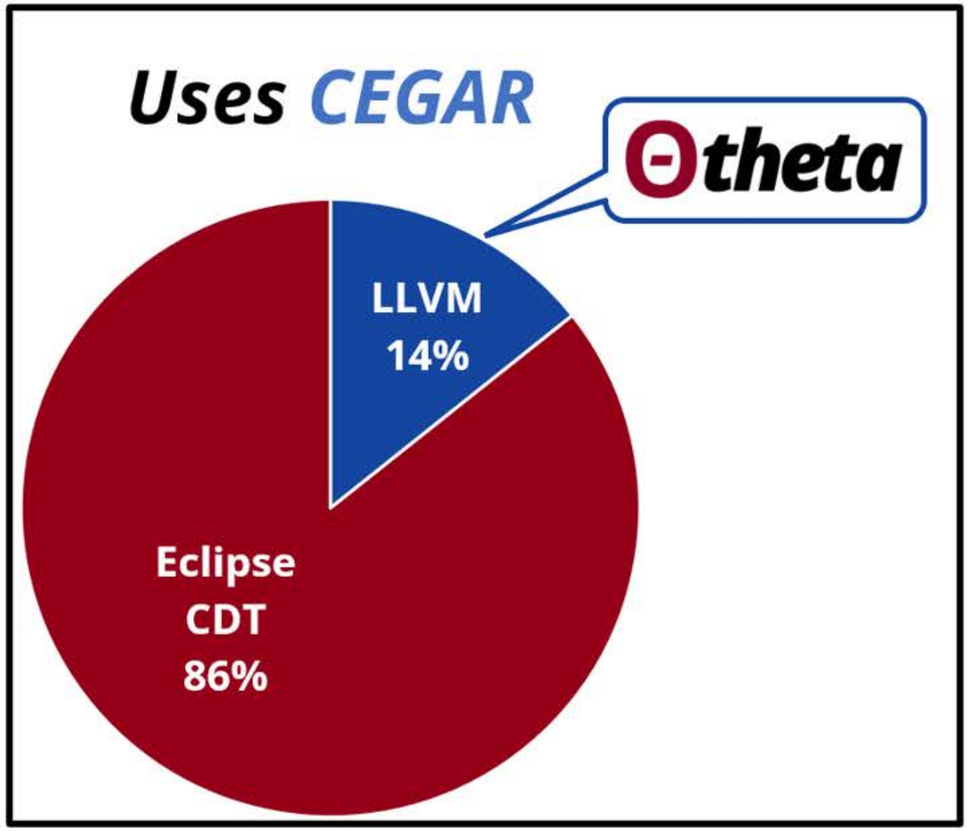
- Works well for BMC: ✓

LLVM

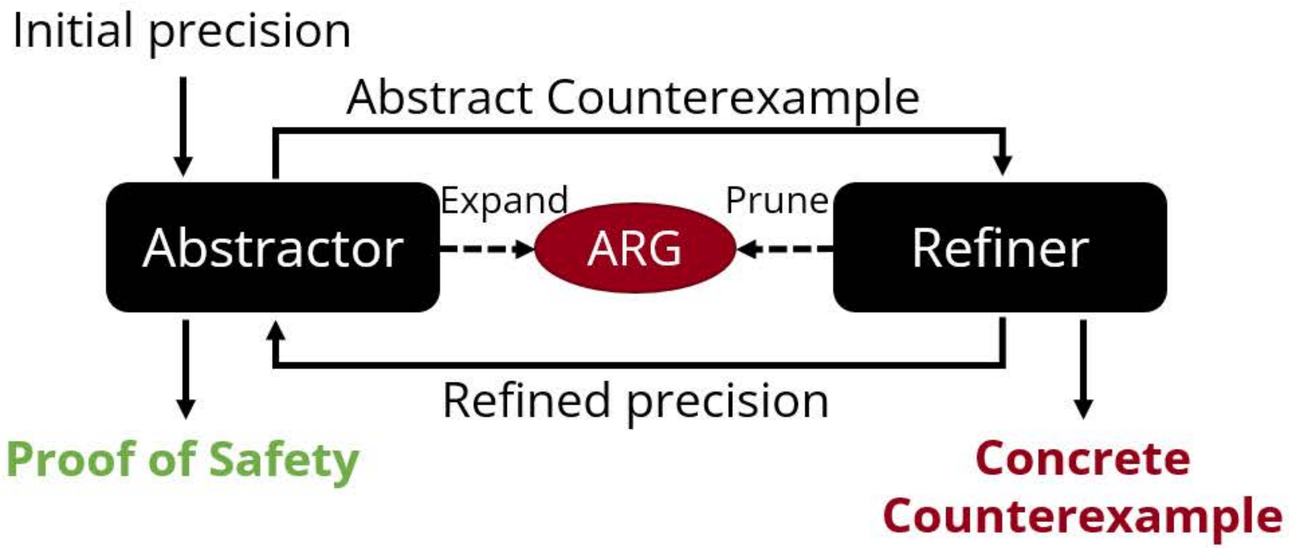


- Works well for BMC: ✓
- Works poorly for CEGAR: ?

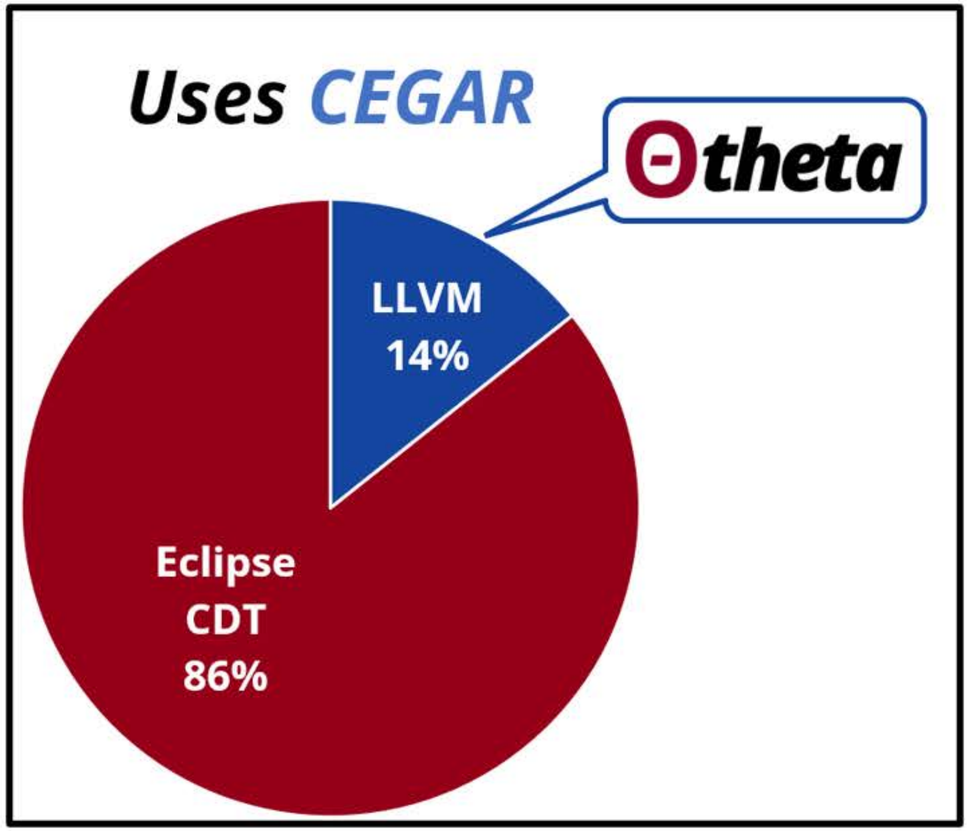
LLVM



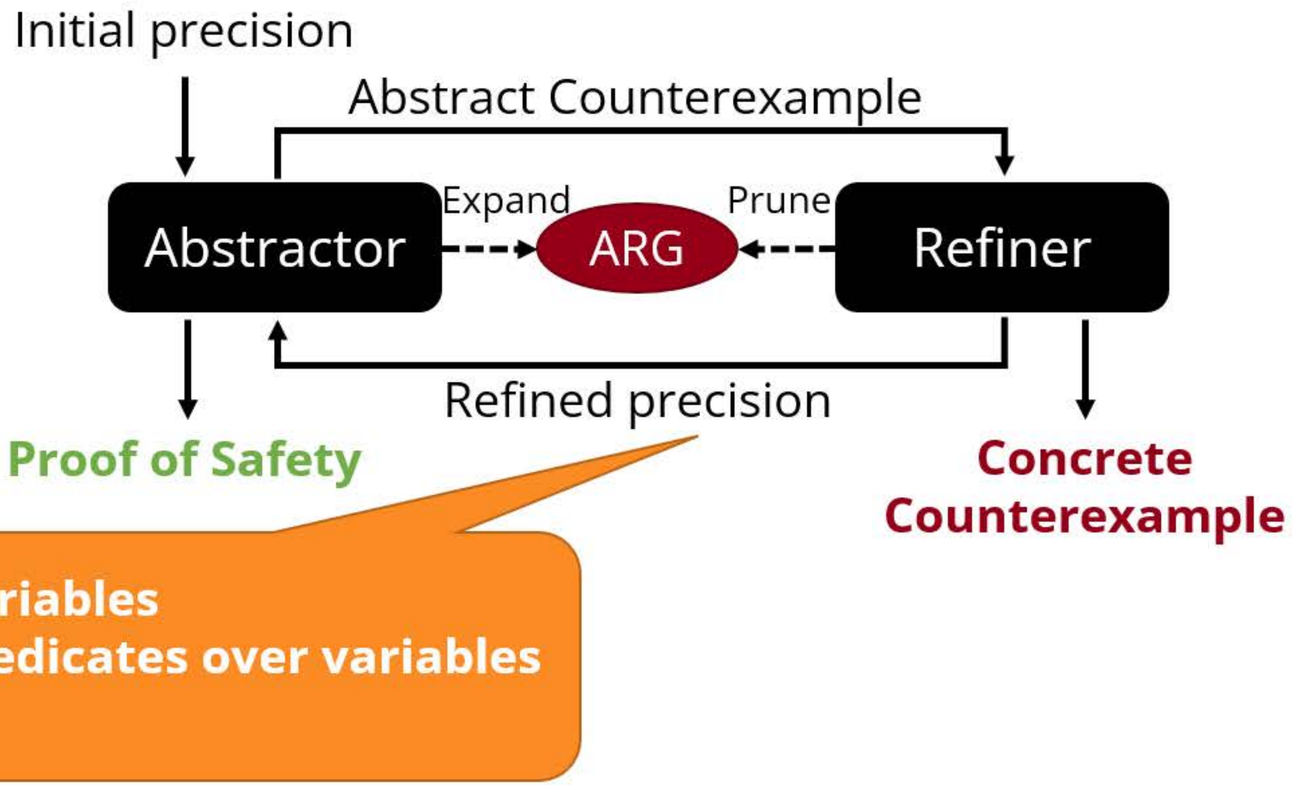
- Works well for BMC: ✓
- Works poorly for CEGAR: ?



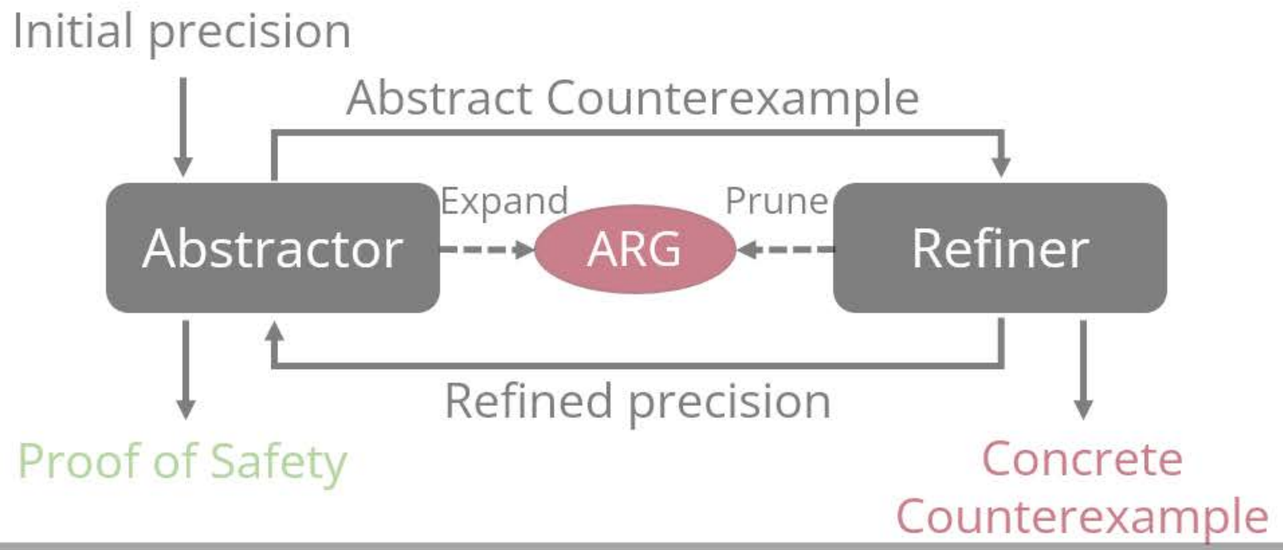
LLVM



- Works well for BMC: ✓
- Works poorly for CEGAR: ?

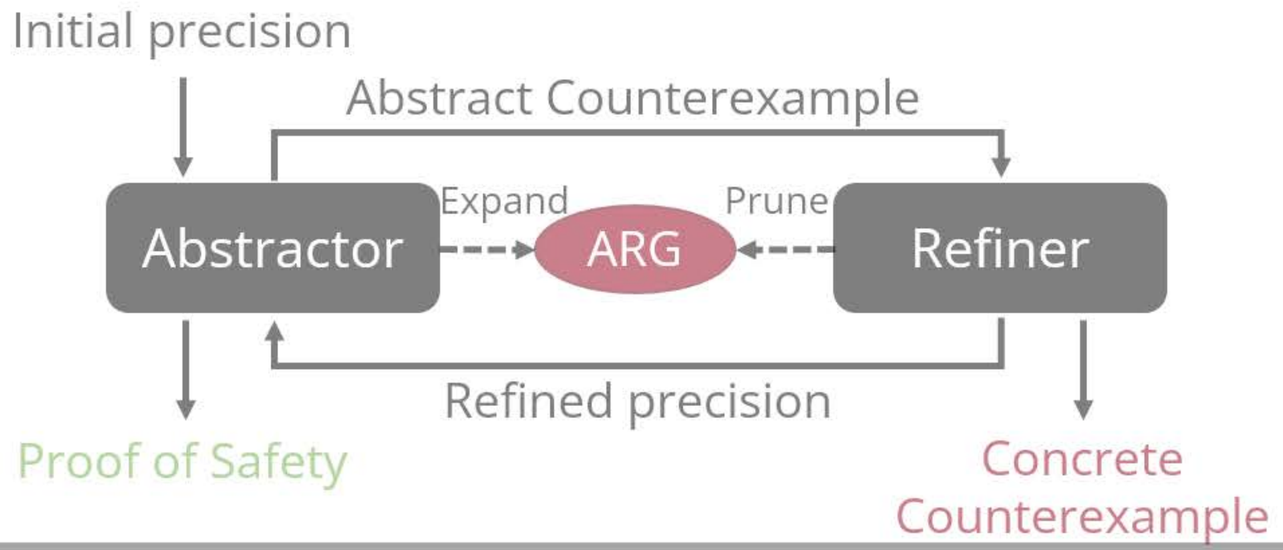


Use of the Precision in CEGAR



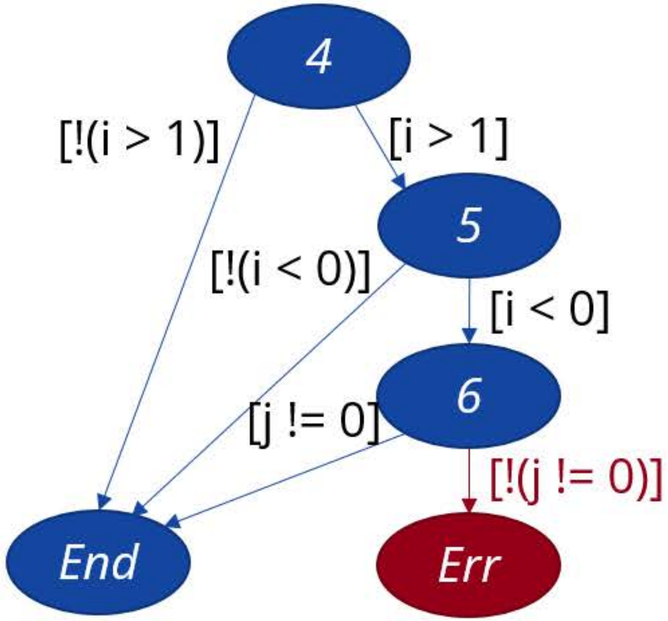
Use of the Precision in CEGAR

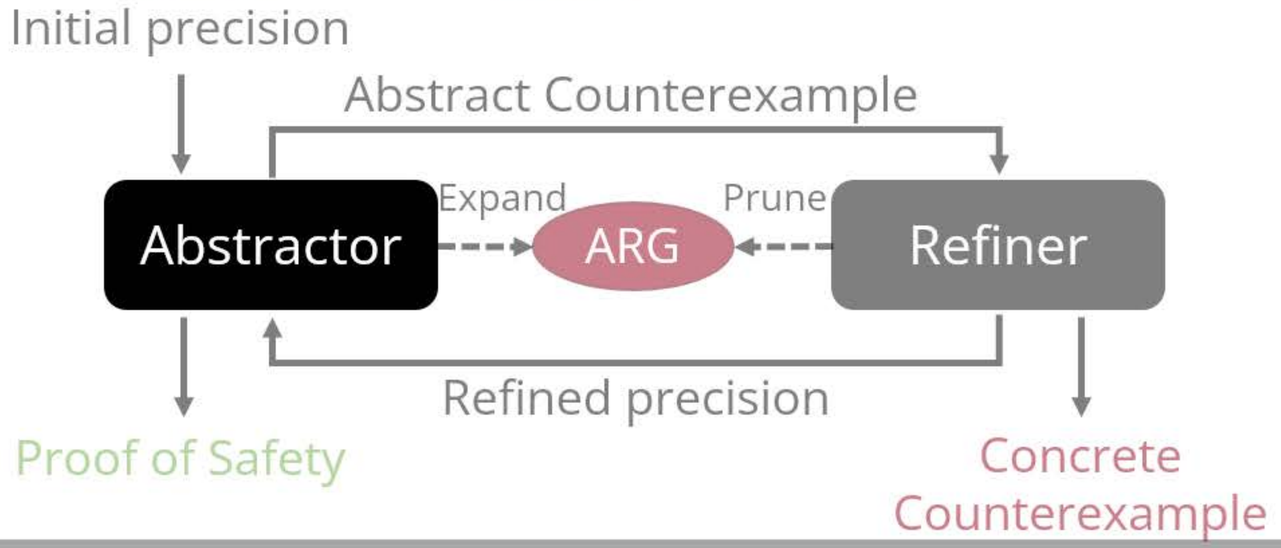
```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



Use of the Precision in CEGAR

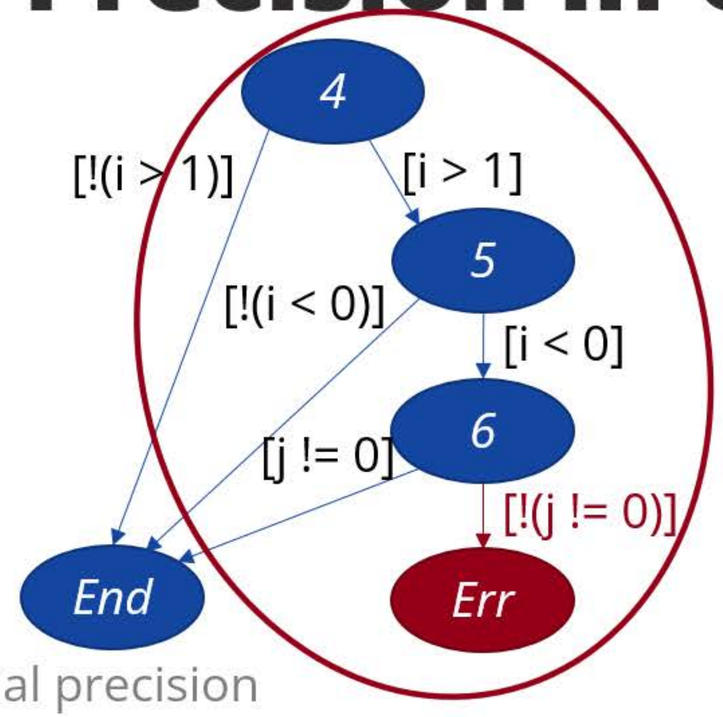
```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```

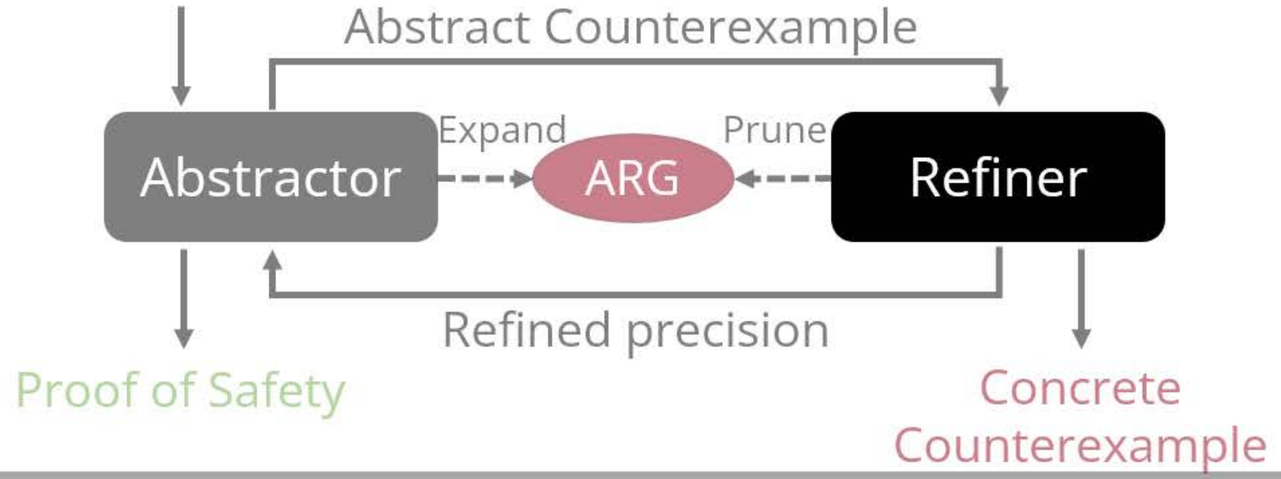




Use of the Precision in CEGAR

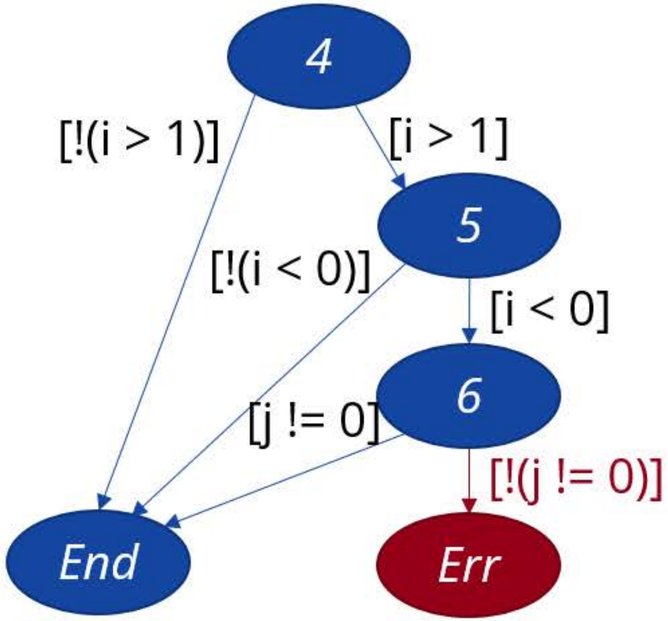
```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



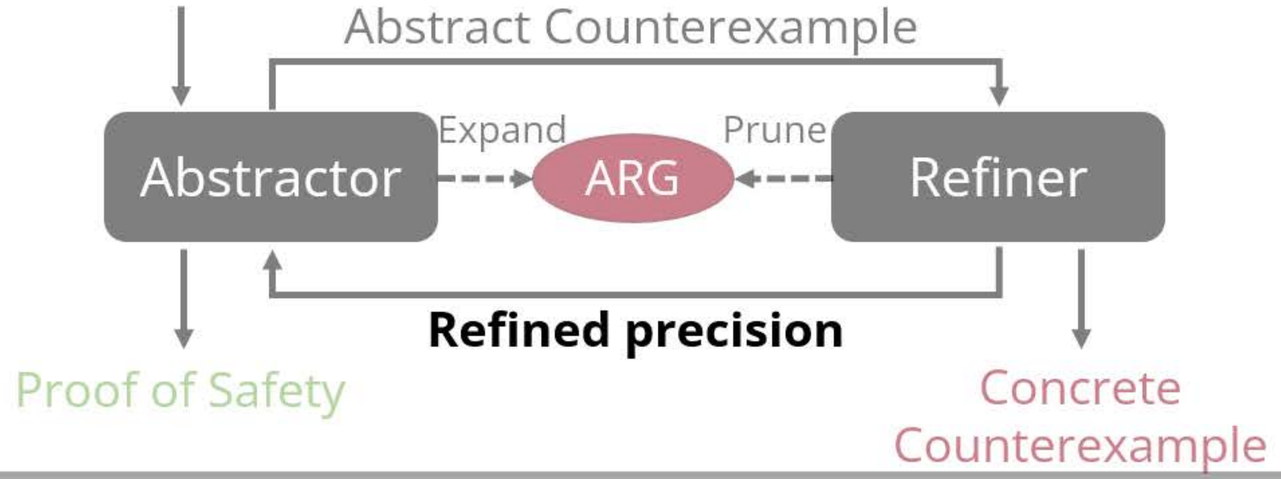


Use of the Precision in CEGAR

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```

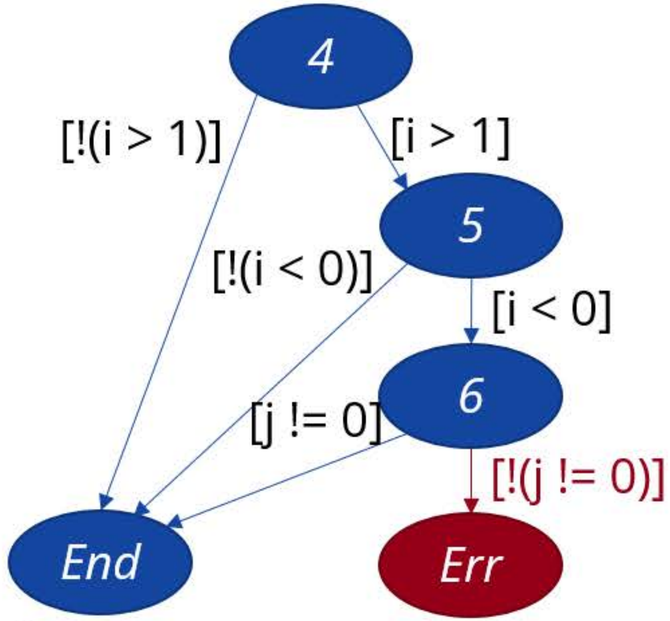


{i > 1}

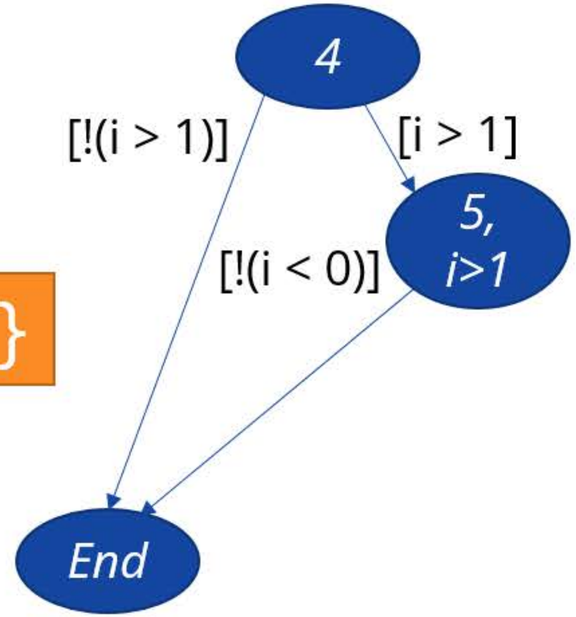


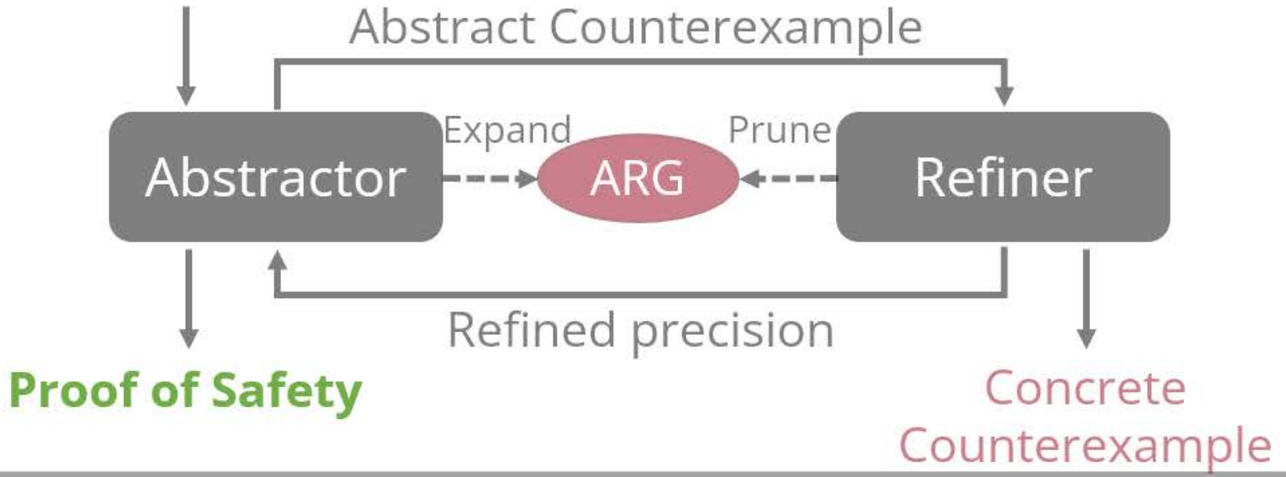
Use of the Precision in CEGAR

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



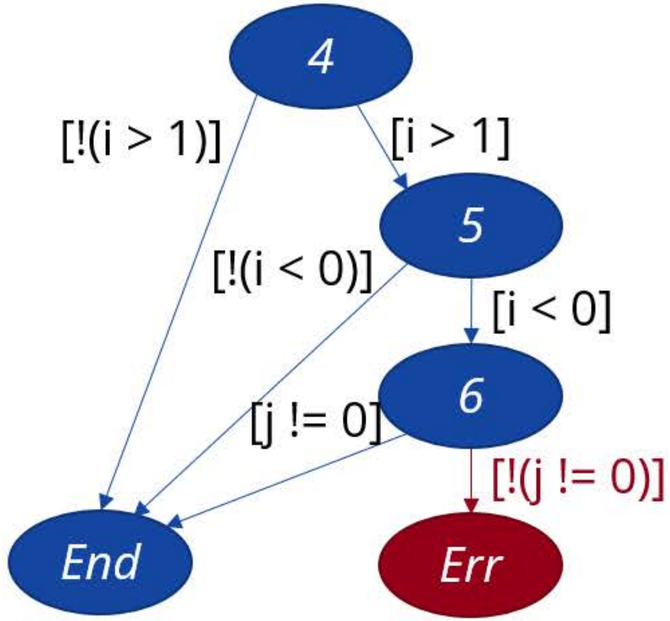
{i > 1}



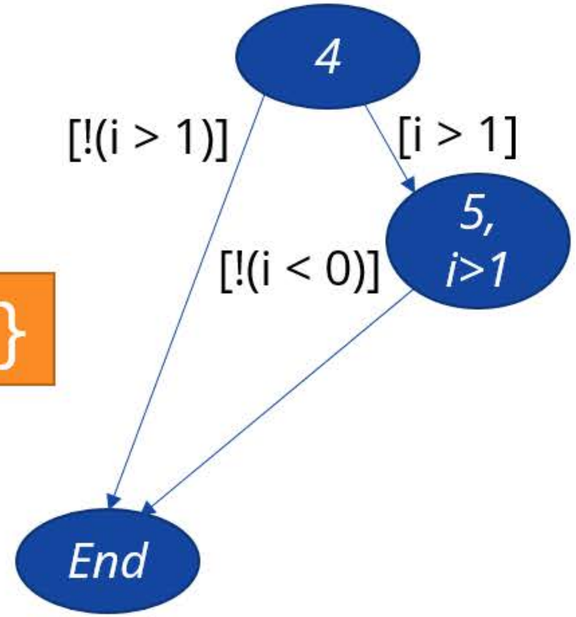


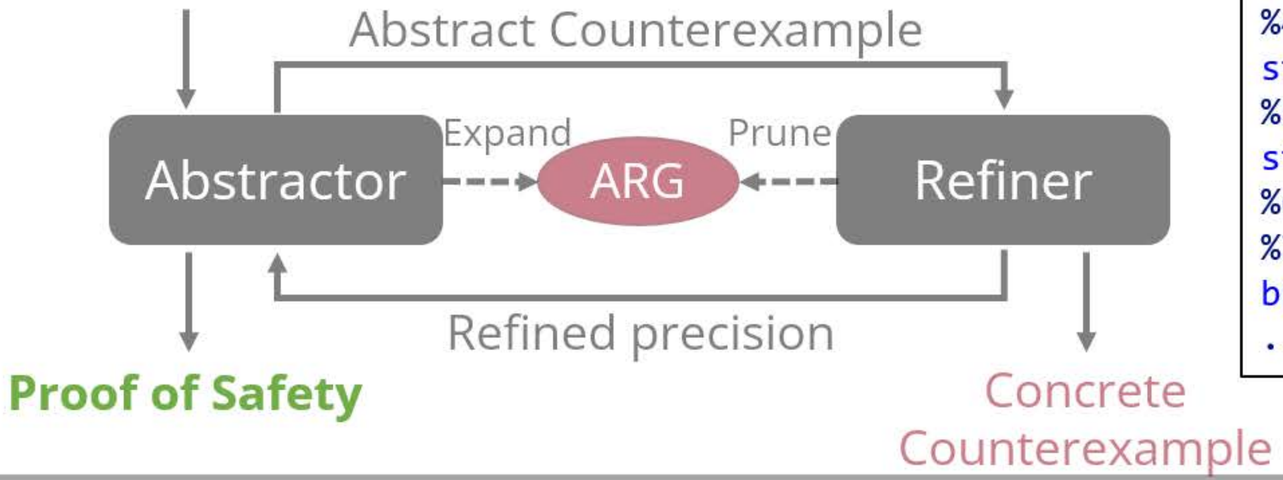
Use of the Precision in CEGAR

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



{i > 1}



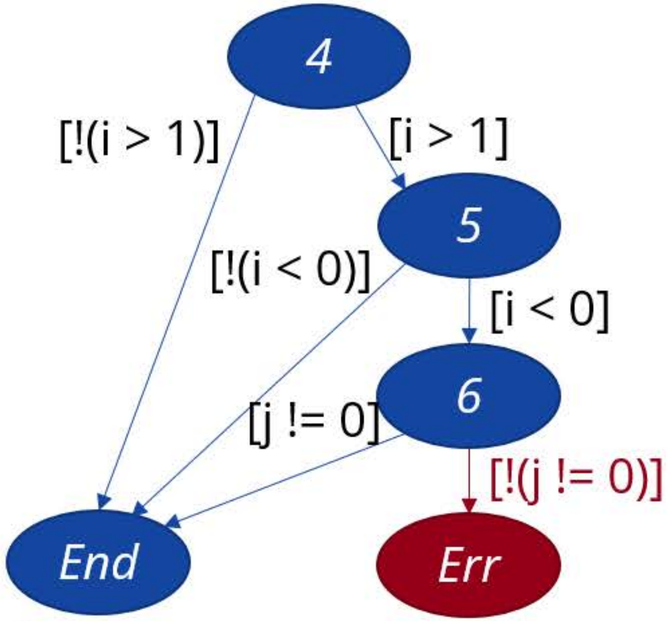



```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

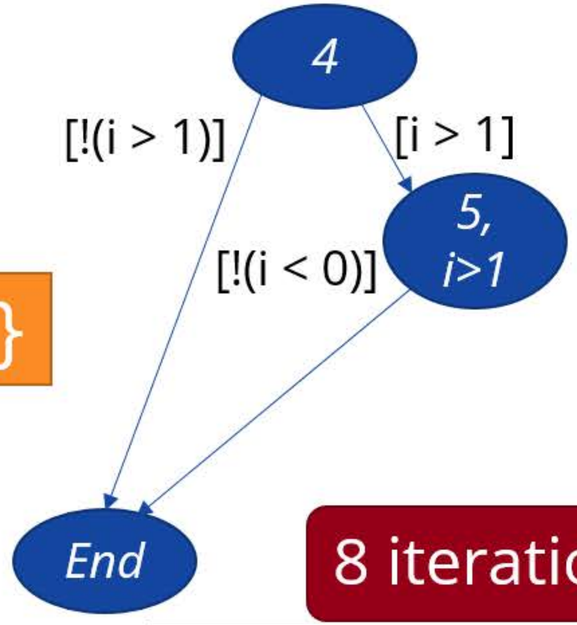
LLVM IR

Use of the Precision in CEGAR

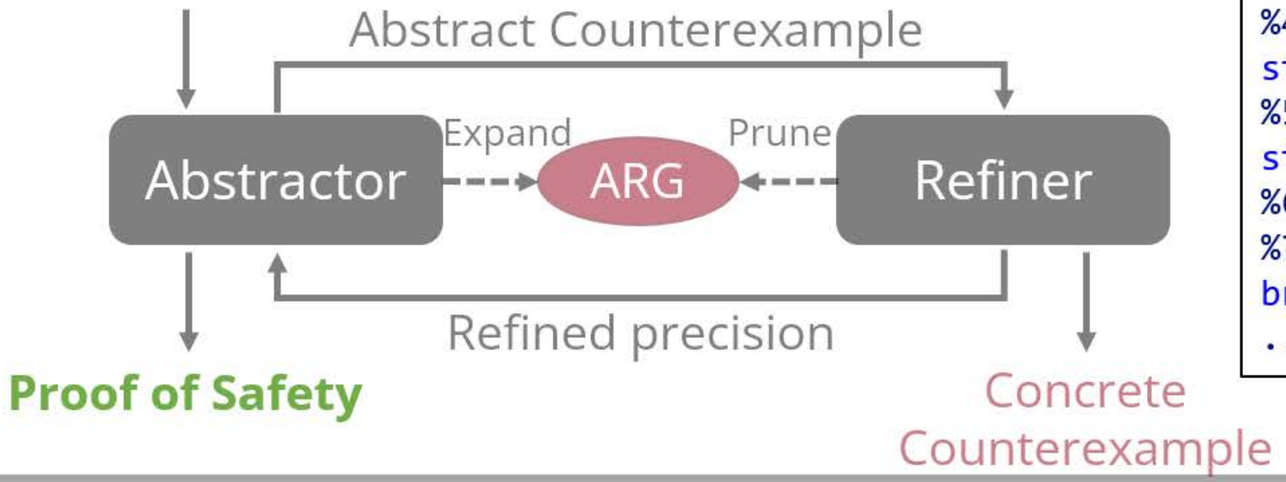
```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



{i > 1}



8 iterations




```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

LLVM IR

LLVM

C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```

LLVM IR

```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

LLVM

C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```

- 2 variables

LLVM IR

```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

LLVM

C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```

- 2 variables

LLVM IR

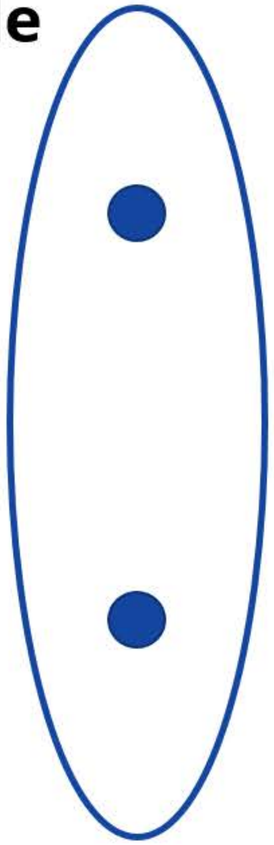
```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

- 18 “variables”

LLVM

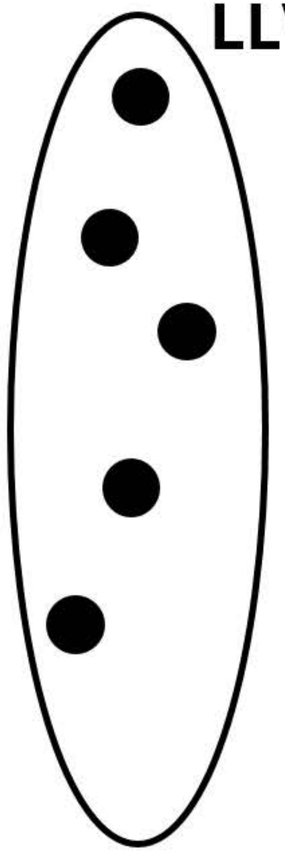
C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



- 2 variables

LLVM IR



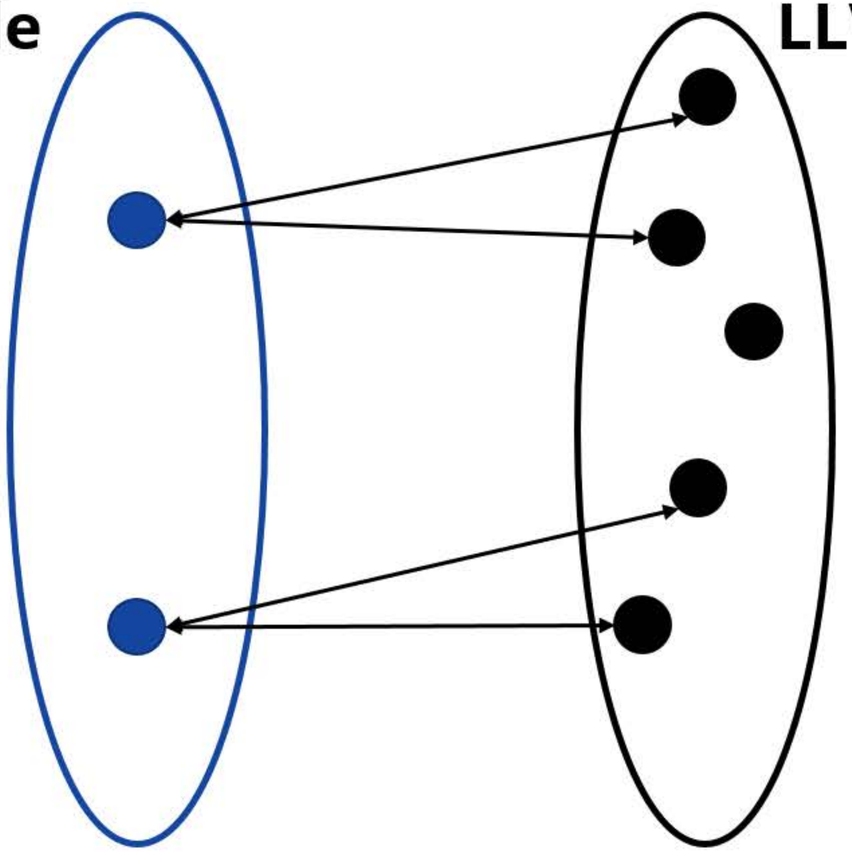
```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

- 18 “variables”

LLVM

C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



LLVM IR

```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

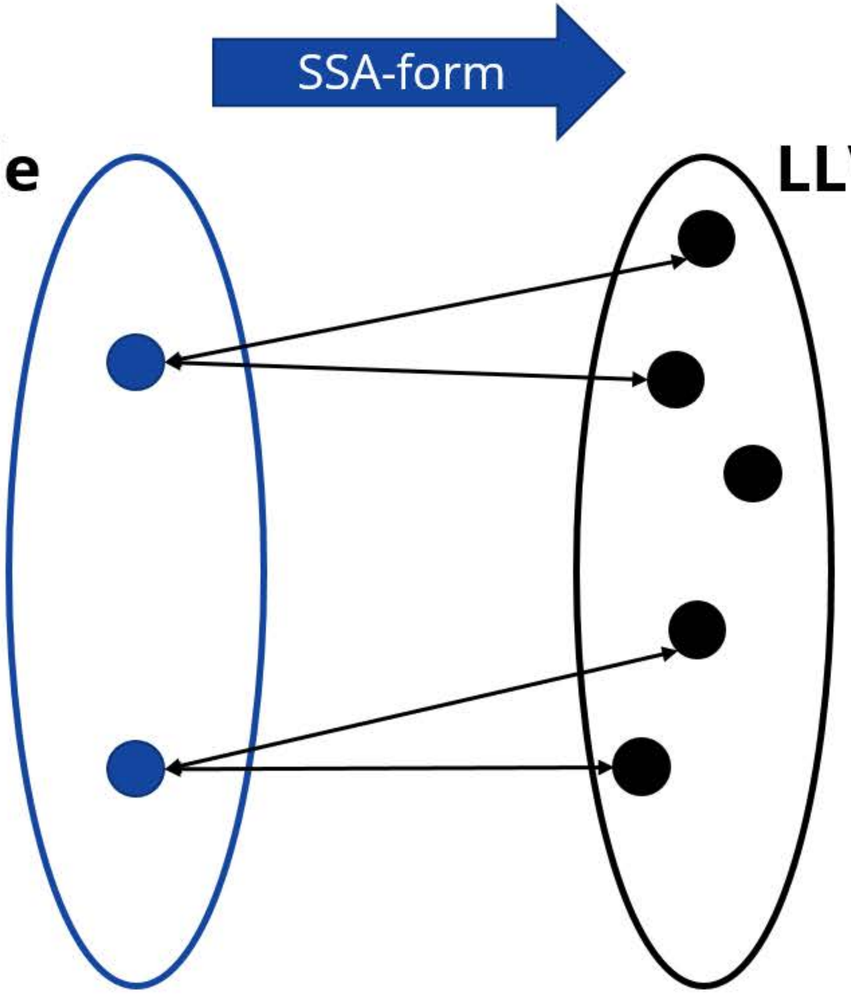
- 2 variables

- 18 “variables”

LLVM

C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



LLVM IR

```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

- 2 variables

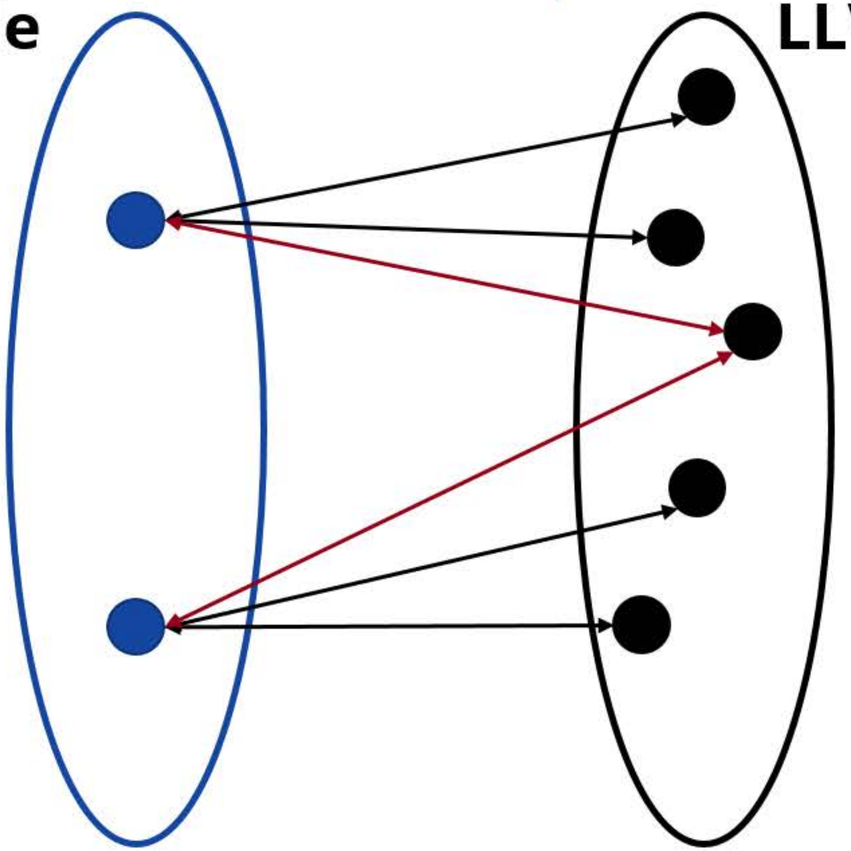
- 18 “variables”

LLVM

SSA-form

C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



LLVM IR

```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

- 2 variables

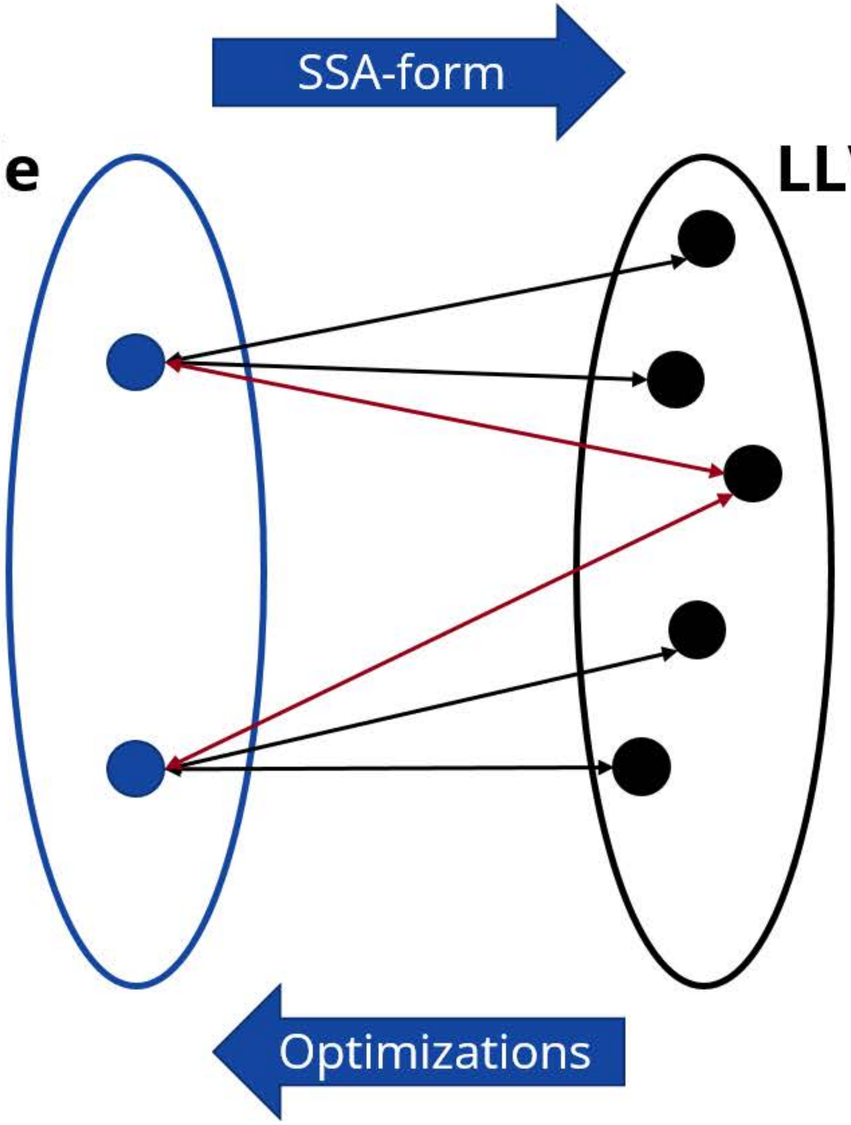
- 18 “variables”

ftsrg

LLVM

C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



LLVM IR

```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

- 2 variables

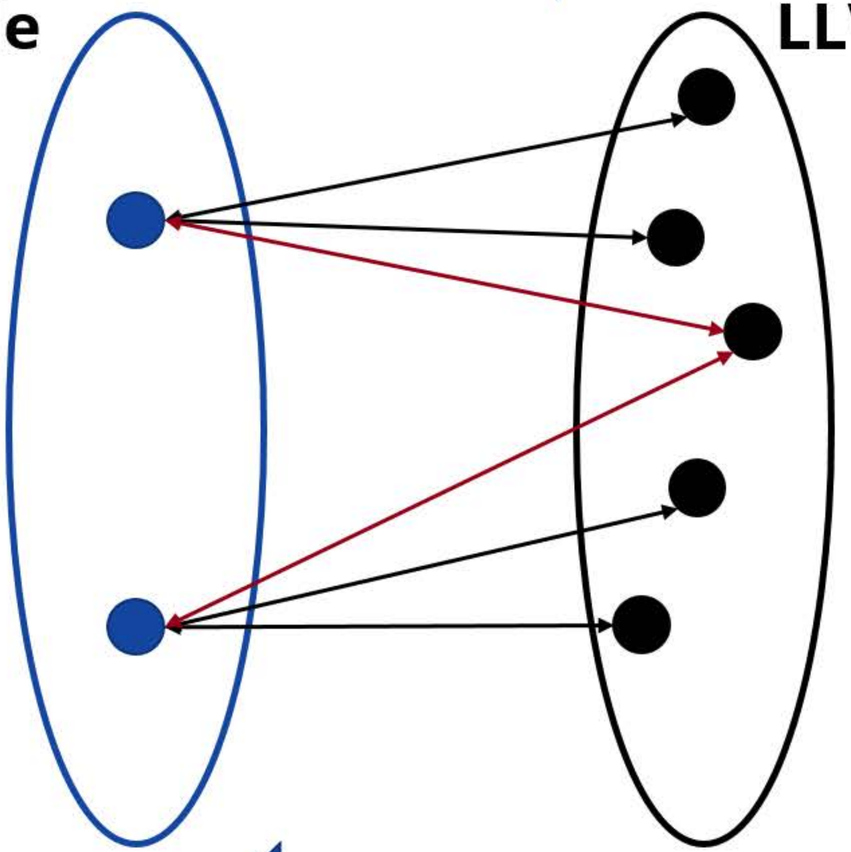
- 18 “variables”

LLVM

SSA-form

C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



LLVM IR

```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

Optimizations

- 2 variables
- Well-defined roles

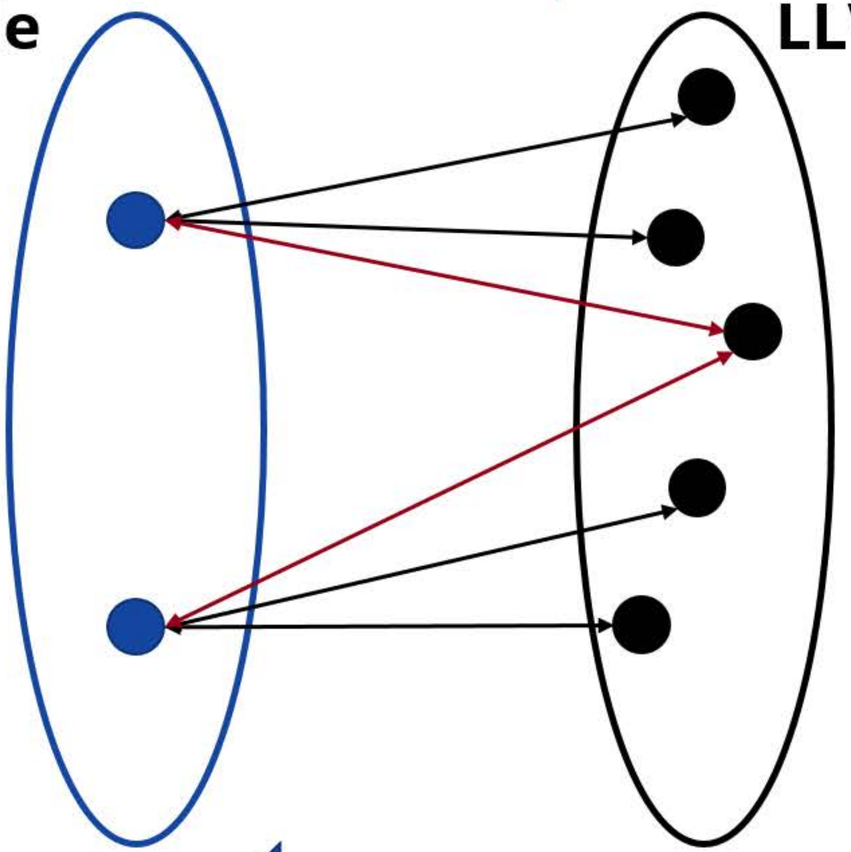
- 18 “variables”

LLVM

SSA-form

C code

```
1: int main() {
2:   int i = nondet();
3:   int j = nondet();
4:   if(i > 1) {
5:     if (i < 0) {
6:       assert(j != 0);
7:     }
8:   }
9: }
```



LLVM IR

```
...
%4 = call i32 (...) @nondet()
store i32 %4, i32* %2, align 4
%5 = call i32 (...) @nondet()
store i32 %5, i32* %3, align 4
%6 = load i32, i32* %2, align 4
%7 = icmp eq i32 %6, 1
br i1 %7, label %8, label %17
...
```

Optimizations

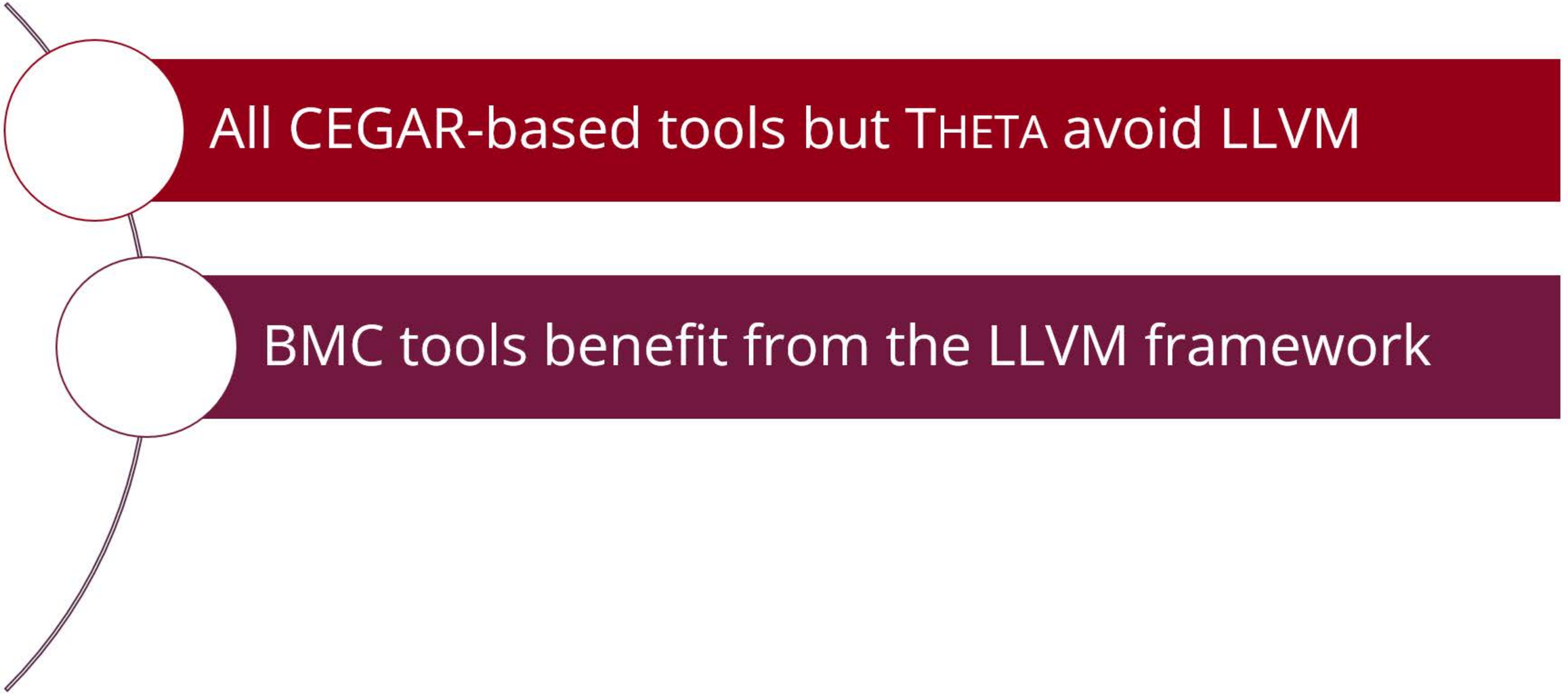
- 2 variables
- Well-defined roles

- 18 “variables”
- N:N mapping

Summary of the Survey

Summary of the Survey

Summary of the Survey



Summary of the Survey

All CEGAR-based tools but THETA avoid LLVM

BMC tools benefit from the LLVM framework

LLVM's SSA form hurts the performance of CEGAR

Verification-first frontend

Proposal

Requirements

Requirements

R1 *Formal*	The resulting model shall be **mathematically precise** and algorithmically **easy to interpret**

Requirements

R1 *Formal*	The resulting model shall be **mathematically precise** and algorithmically **easy to interpret**
R2 *Configurable*	The transformation process shall be **configurable** in terms of handling **undefined elements** of the source language

Requirements

R1 *Formal*	The resulting model shall be **mathematically precise** and algorithmically **easy to interpret**
R2 *Configurable*	The transformation process shall be **configurable** in terms of handling **undefined elements** of the source language
R3 *Direct Access*	The analyses must have **direct access to variables** in the C code

Requirements

R1 *Formal*	The resulting model shall be **mathematically precise** and algorithmically **easy to interpret**
R2 *Configurable*	The transformation process shall be **configurable** in terms of handling **undefined elements** of the source language
R3 *Direct Access*	The analyses must have **direct access to variables** in the C code
R4 *Verifier-Centric*	The resulting model should be **optimized for verification**, not for executable generation

Requirements

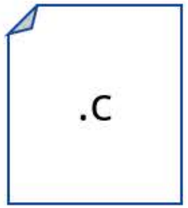
R1 *Formal*	The resulting model shall be **mathematically precise** and algorithmically **easy to interpret**
R2 *Configurable*	The transformation process shall be **configurable** in terms of handling **undefined elements** of the source language
R3 *Direct Access*	The analyses must have **direct access to variables** in the C code
R4 *Verifier-Centric*	The resulting model should be **optimized for verification**, not for executable generation
R5 *Metadata Access*	The verification tools must have access to **metadata** from the source file

Requirements

R1 *Formal*	The resulting model shall be **mathematically precise** and algorithmically **easy to interpret**
R2 *Configurable*	The transformation process shall be **configurable** in terms of handling **undefined elements** of the source language
R3 *Direct Access*	The analyses must have **direct access to variables** in the C code
R4 *Verifier-Centric*	The resulting model should be **optimized for verification**, not for executable generation
R5 *Metadata Access*	The verification tools must have access to **metadata** from the source file
R6 *Unhandled Patterns*	The transformation must **raise an exception** if an unhandled pattern is found, lest an erroneous model be created

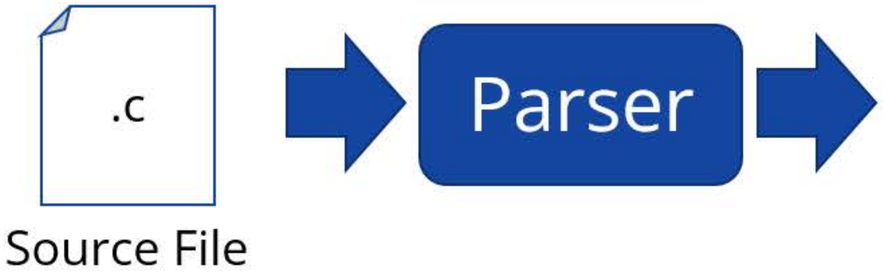
Proposed Workflow - „The LLVM of verification”

Proposed Workflow - „The LLVM of verification”



Source File

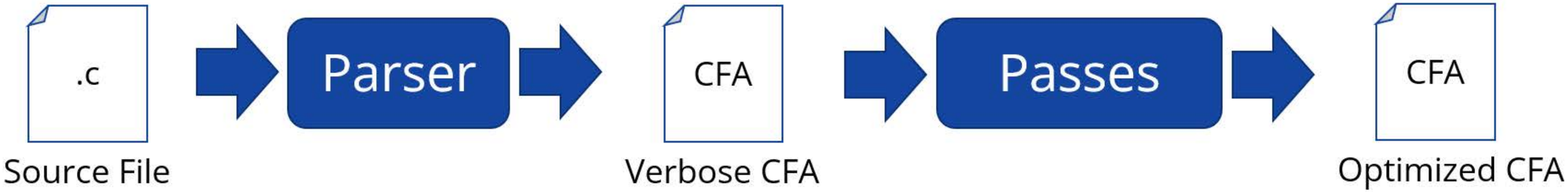
Proposed Workflow - „The LLVM of verification”



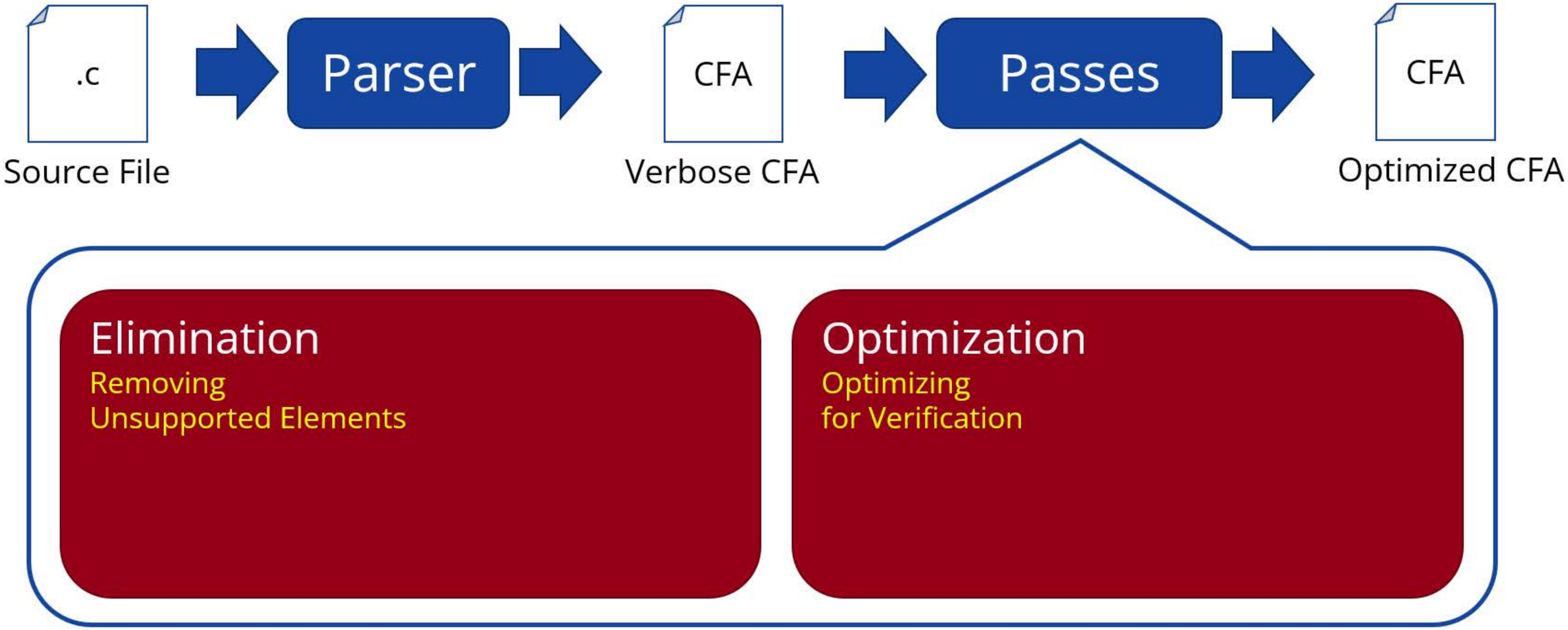
Proposed Workflow - „The LLVM of verification”

Proposed Workflow - „The LLVM of verification”

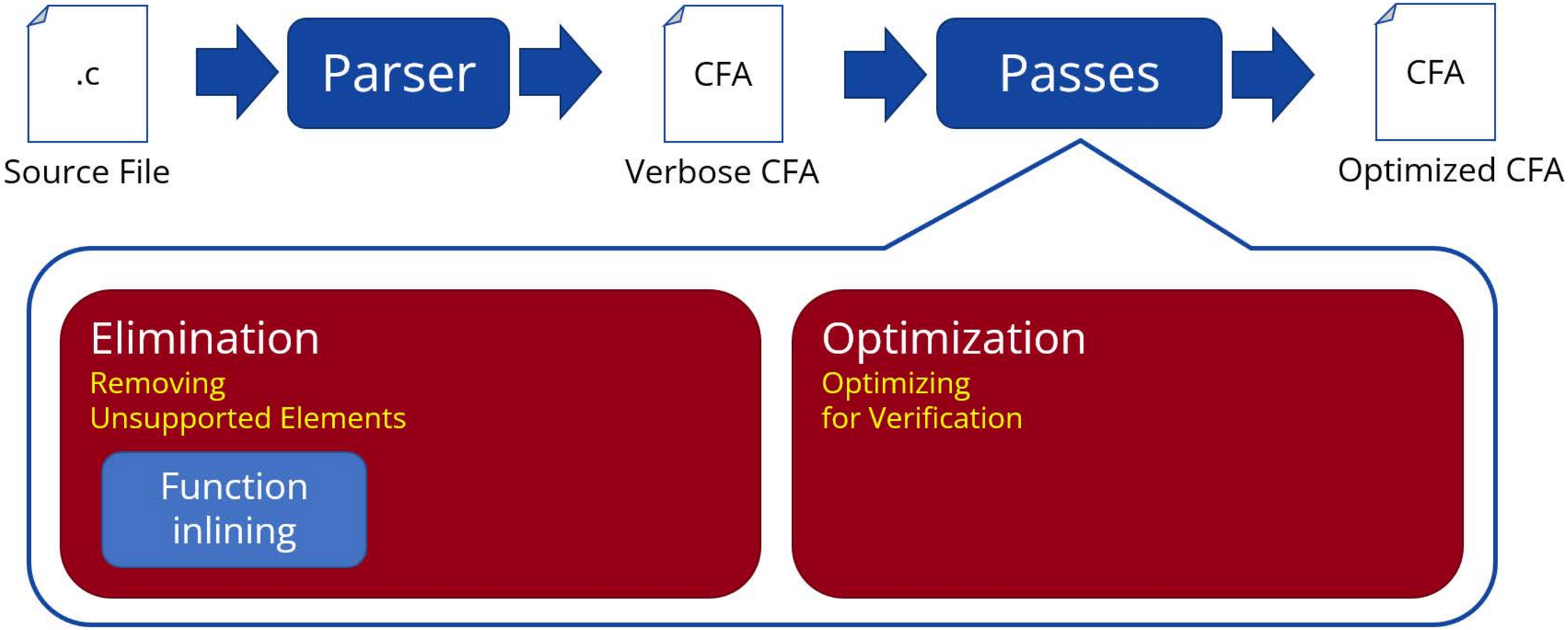
Proposed Workflow - „The LLVM of verification”



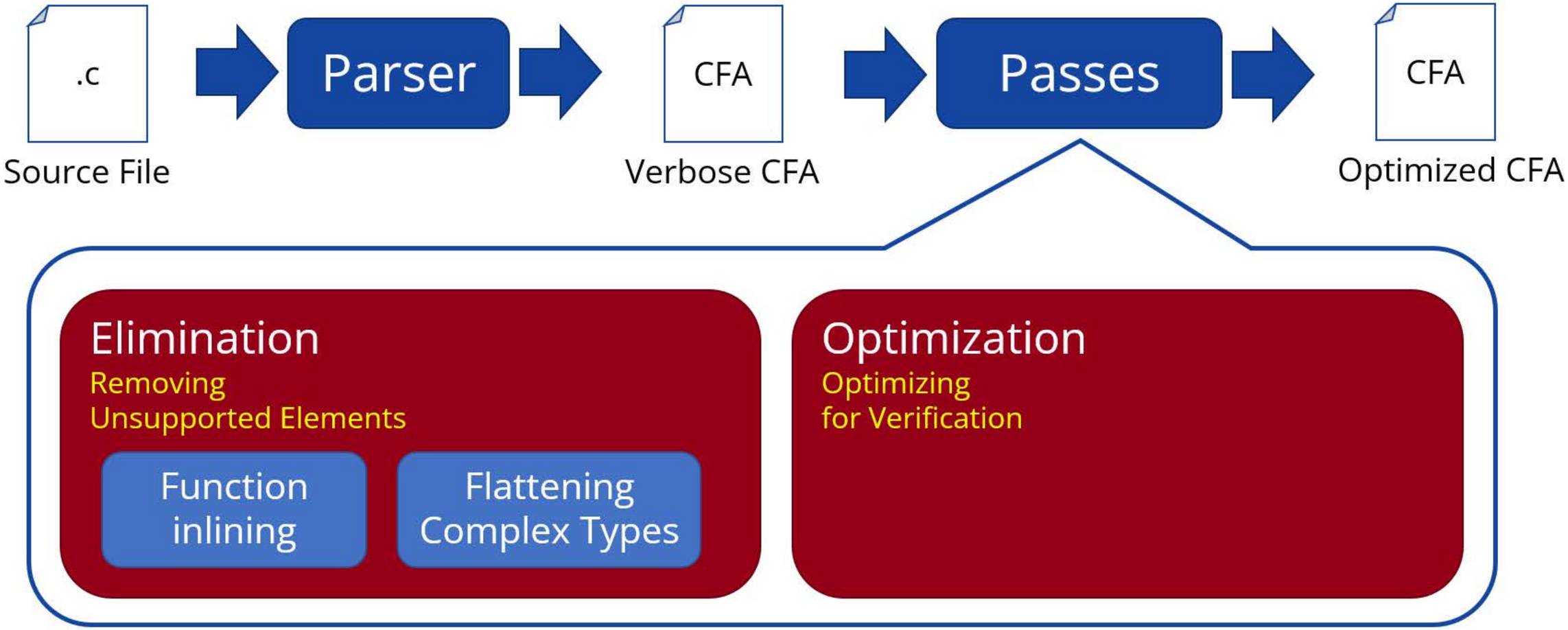
Proposed Workflow - „The LLVM of verification”



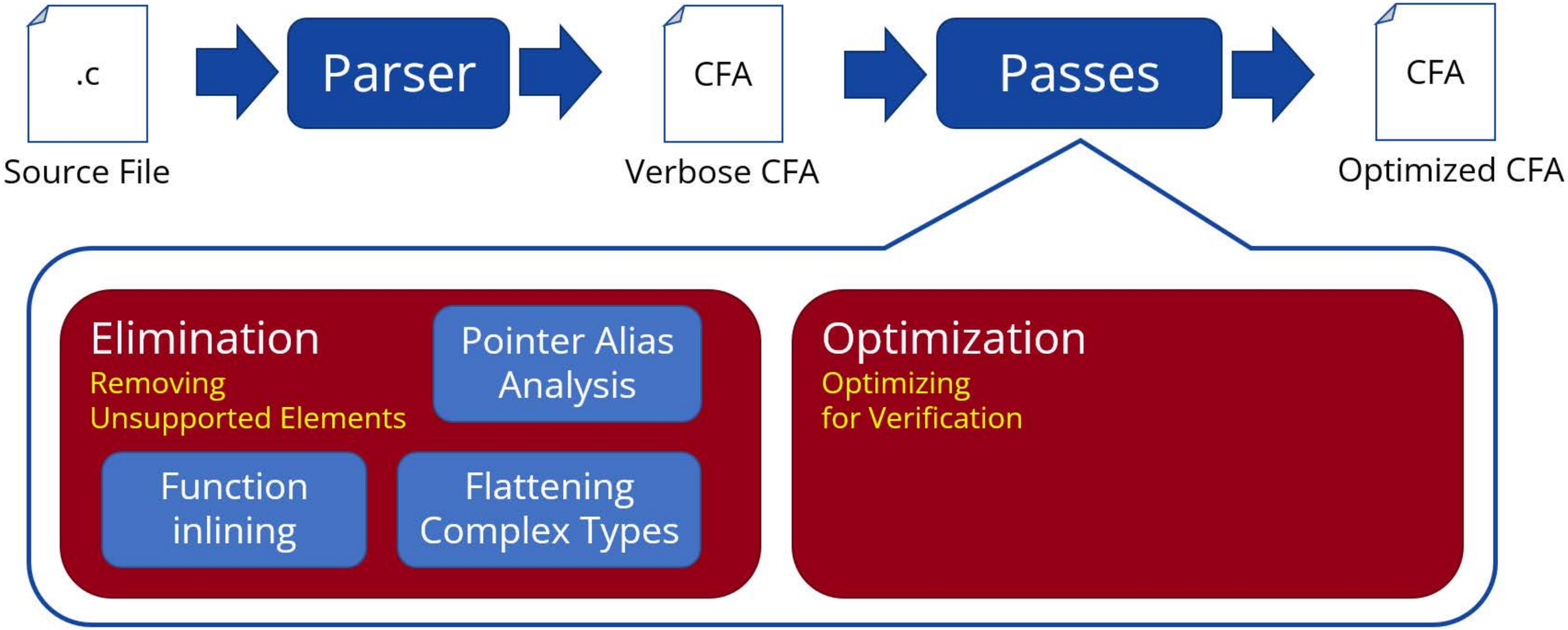
Proposed Workflow - „The LLVM of verification”



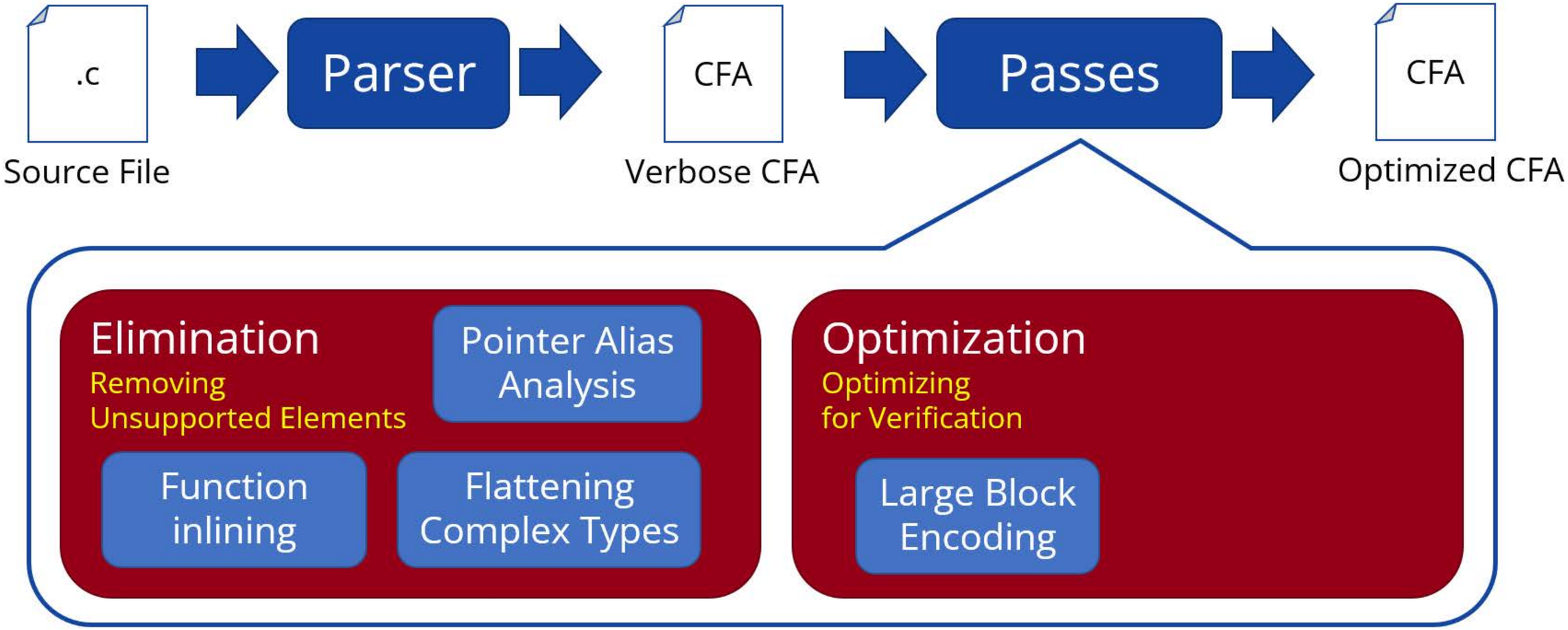
Proposed Workflow - „The LLVM of verification”



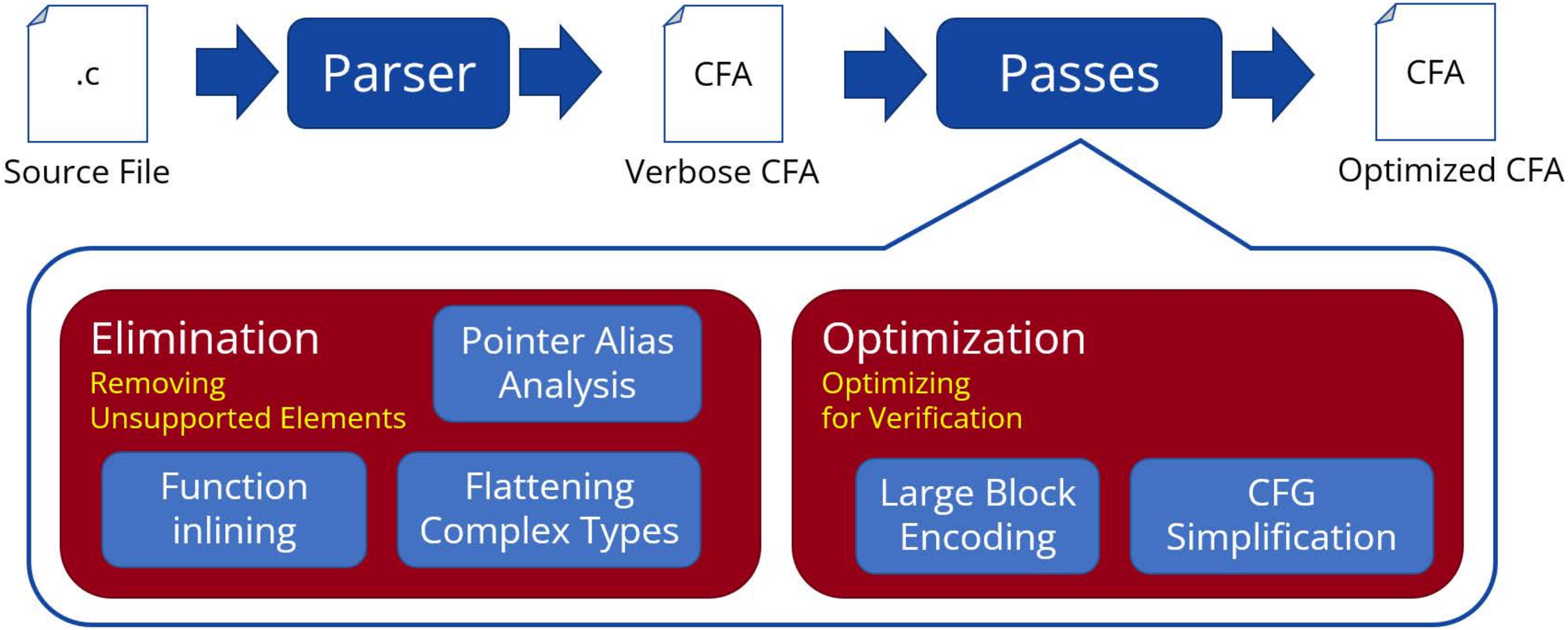
Proposed Workflow - „The LLVM of verification”



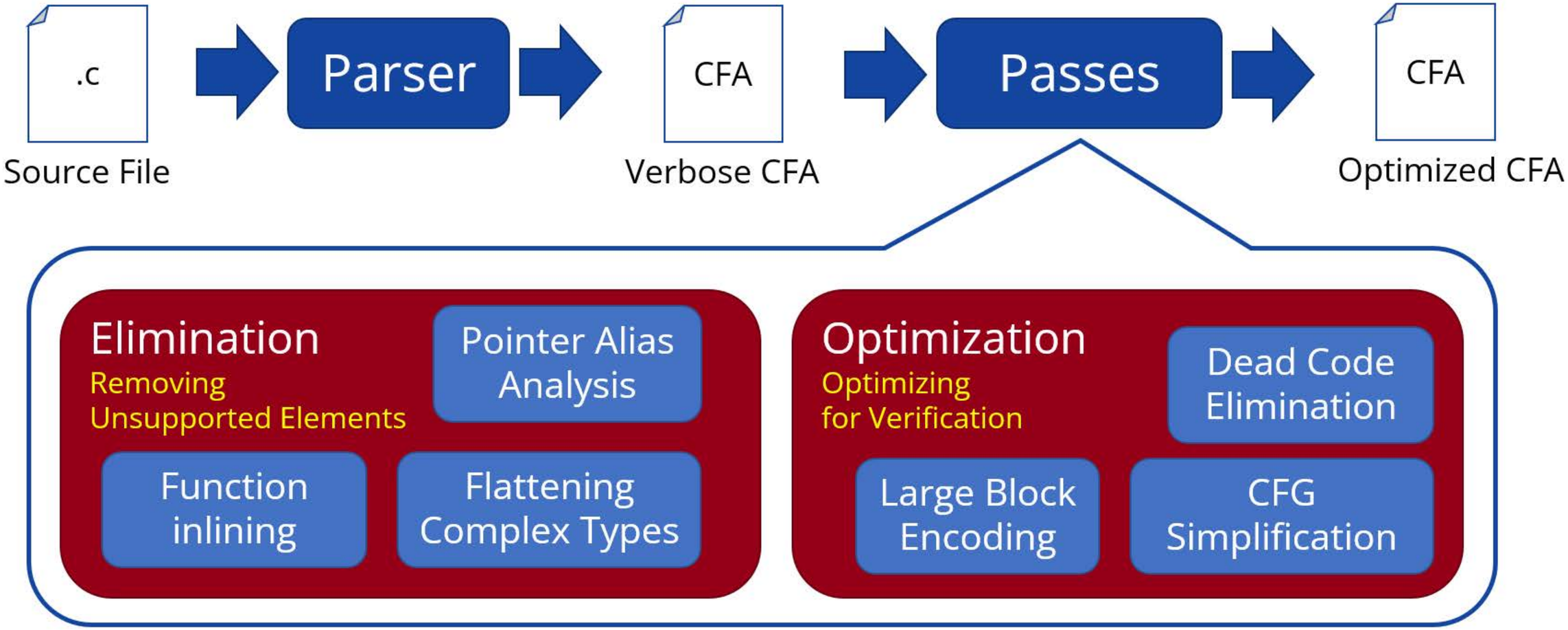
Proposed Workflow - „The LLVM of verification”



Proposed Workflow - „The LLVM of verification”

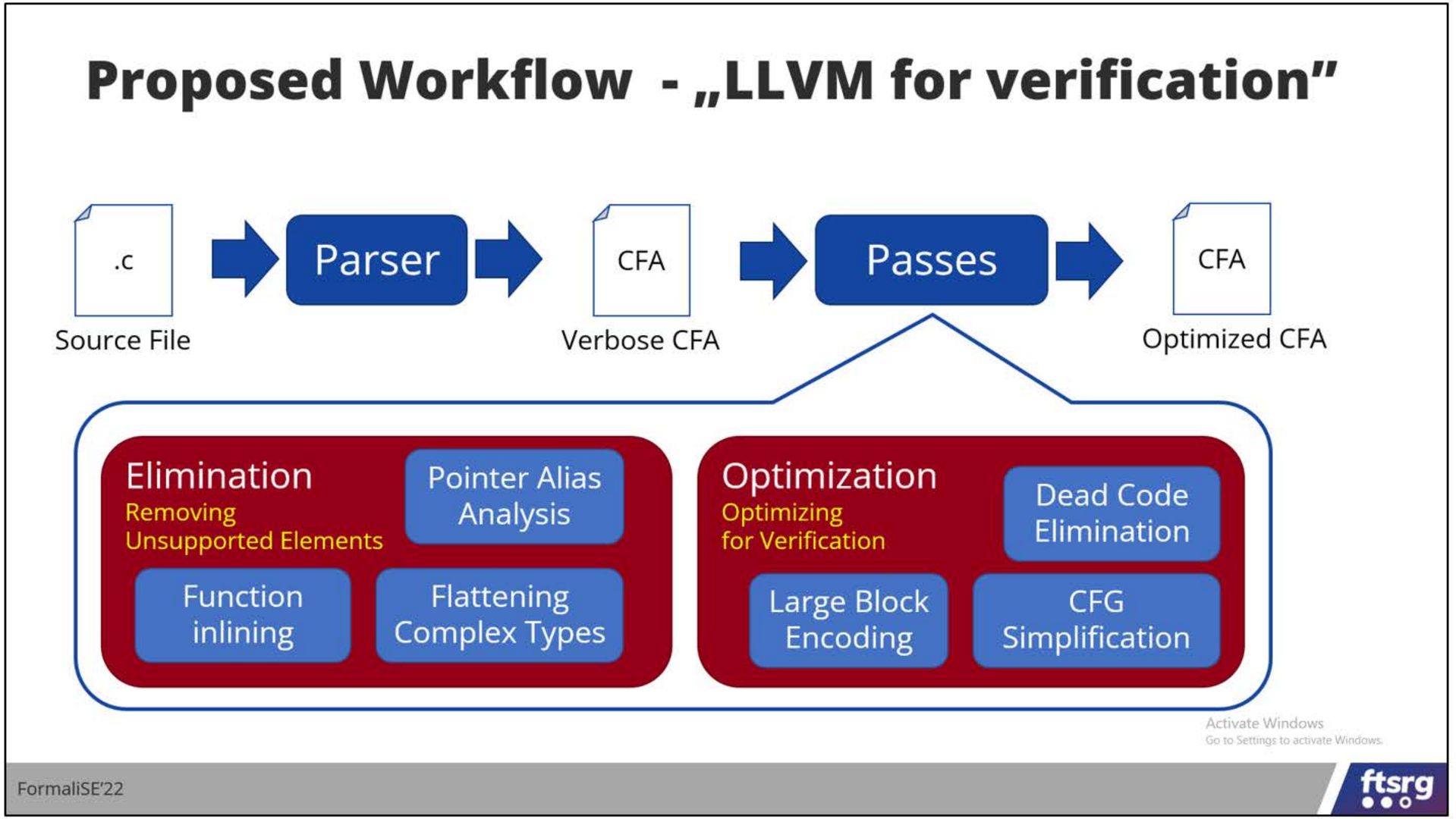


Proposed Workflow - „The LLVM of verification”



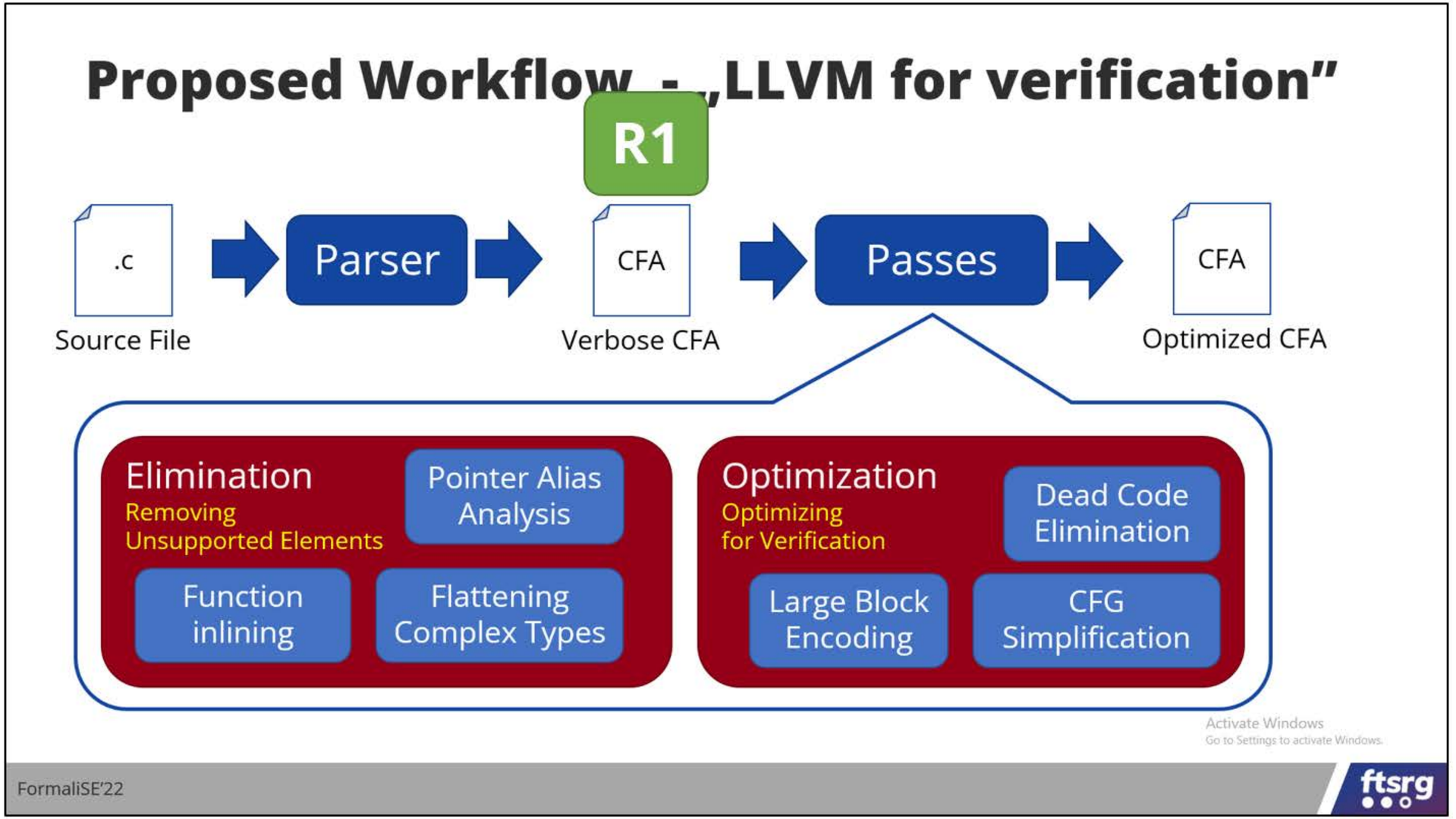
Requirements

- R1 *Formal*
- R2 *Configurable*
- R3 *Direct Access*
- R4 *Verifier-Centric*
- R5 *Metadata Access*
- R6 *Unhandled Patterns*



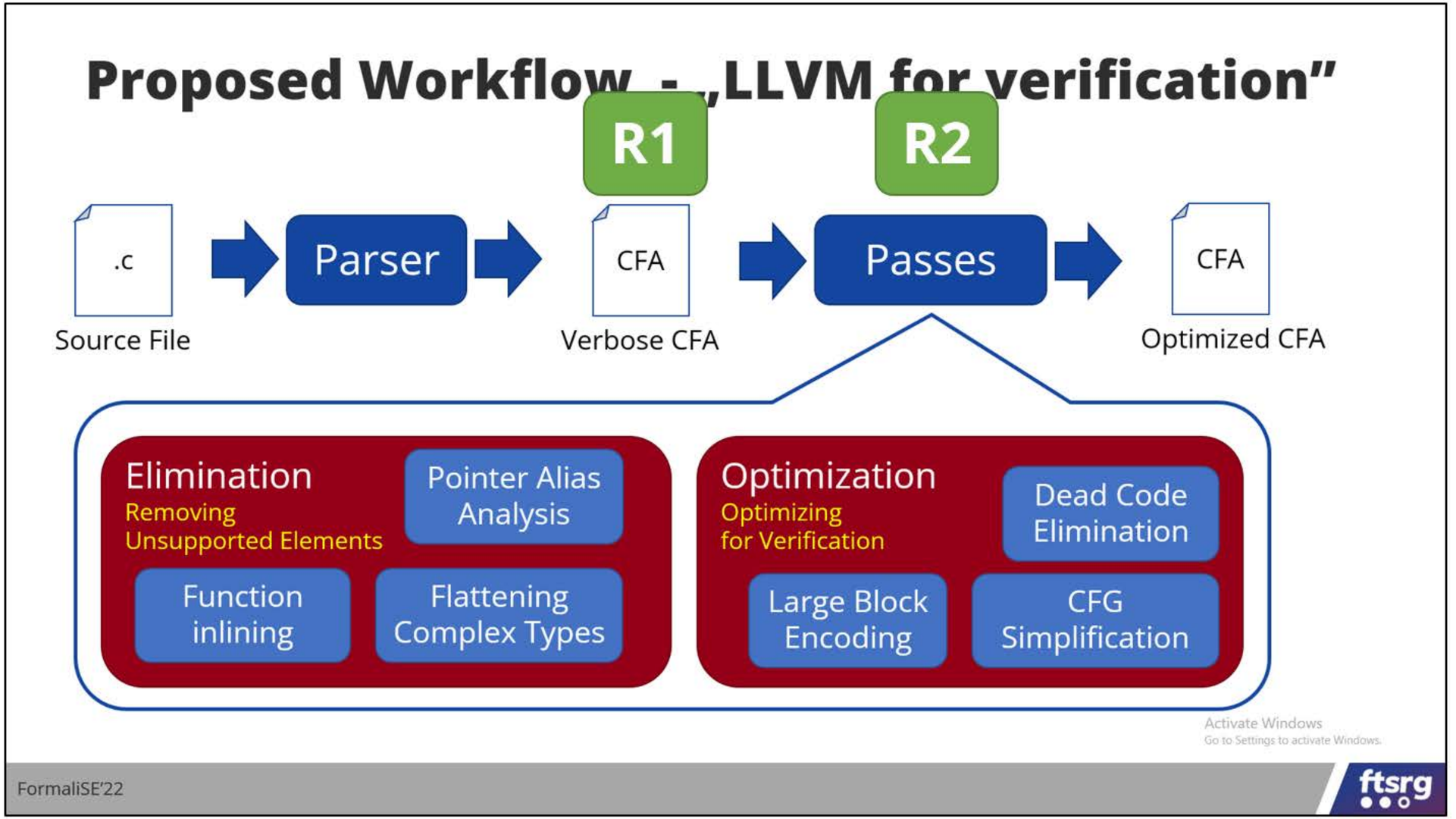
Requirements

- R1 *Formal*
- R2 *Configurable*
- R3 *Direct Access*
- R4 *Verifier-Centric*
- R5 *Metadata Access*
- R6 *Unhandled Patterns*



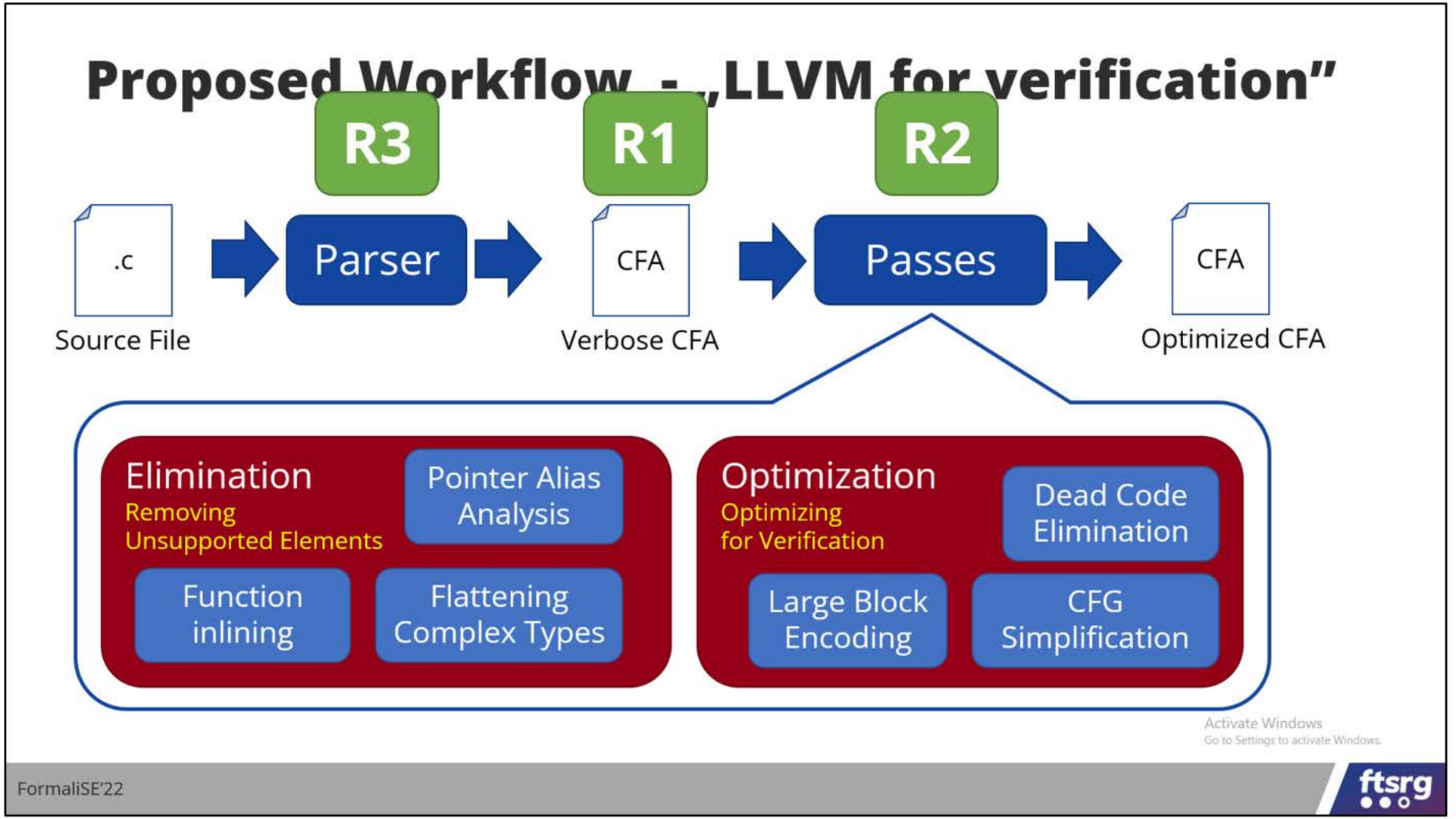
Requirements

- R1 *Formal*
- R2 *Configurable*
- R3 *Direct Access*
- R4 *Verifier-Centric*
- R5 *Metadata Access*
- R6 *Unhandled Patterns*



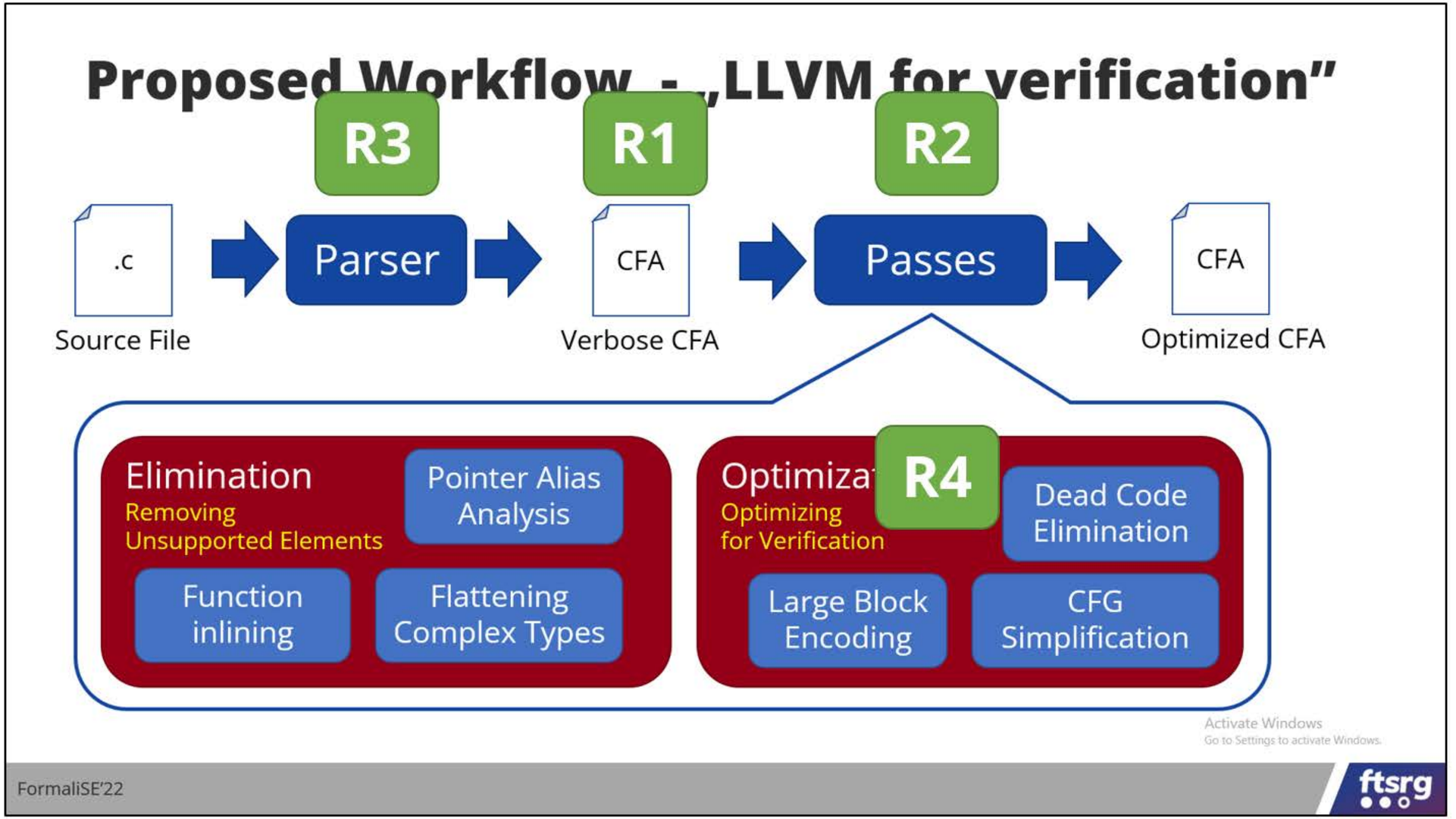
Requirements

- R1 *Formal*
- R2 *Configurable*
- R3 *Direct Access*
- R4 *Verifier-Centric*
- R5 *Metadata Access*
- R6 *Unhandled Patterns*



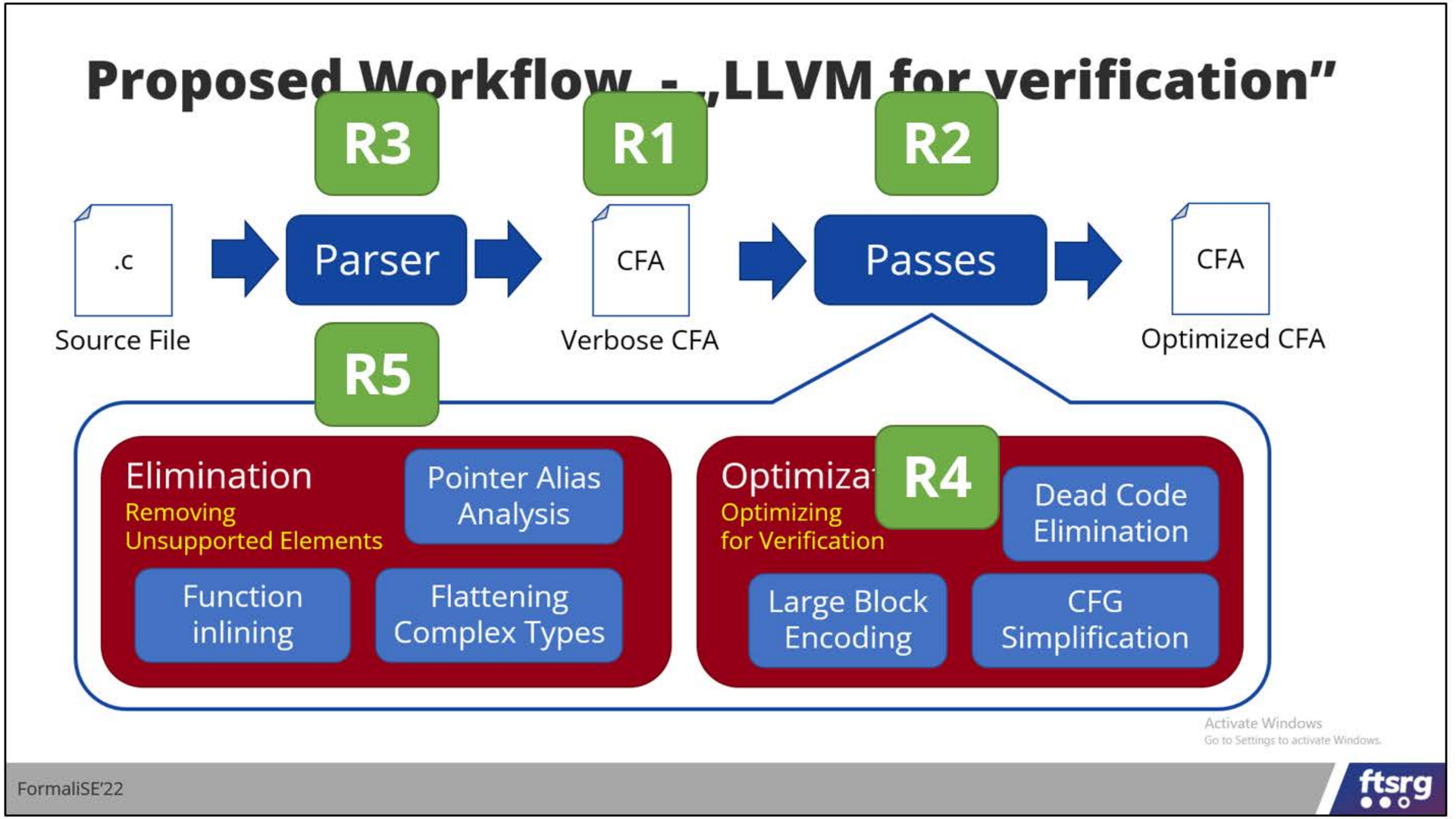
Requirements

- R1 *Formal*
- R2 *Configurable*
- R3 *Direct Access*
- R4 *Verifier-Centric*
- R5 *Metadata Access*
- R6 *Unhandled Patterns*



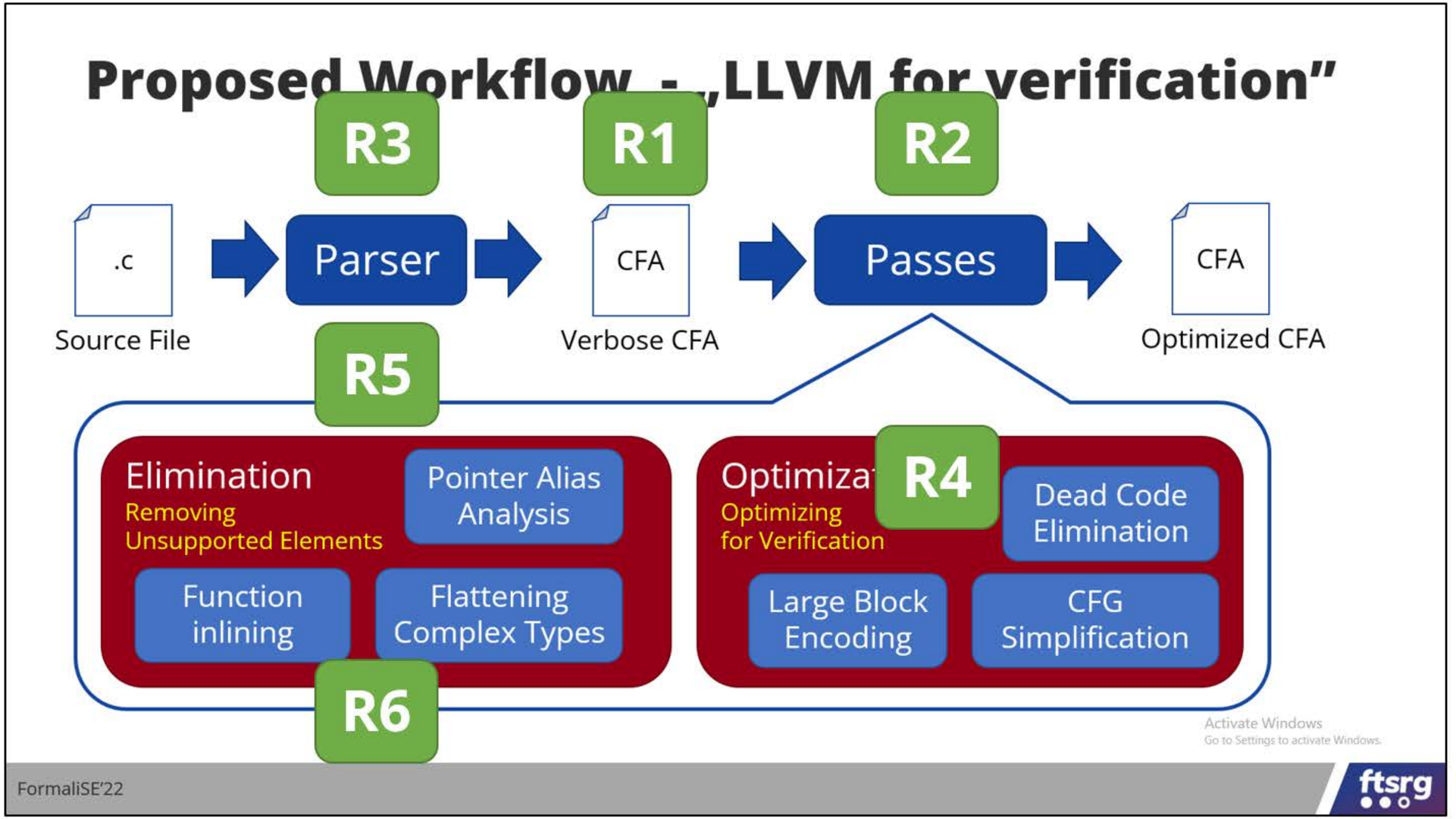
Requirements

- R1 *Formal*
- R2 *Configurable*
- R3 *Direct Access*
- R4 *Verifier-Centric*
- R5 *Metadata Access*
- R6 *Unhandled Patterns*



Requirements

- R1 *Formal*
- R2 *Configurable*
- R3 *Direct Access*
- R4 *Verifier-Centric*
- R5 *Metadata Access*
- R6 *Unhandled Patterns*



Requirements

R1
Formal

R2
Configurable

R3
Direct Access

R4
Verifier-Centric

R5
Metadata Access

R6
Unhandled Patterns

Requirements

	LLVM
R1 *Formal*	✓
R2 *Configurable*	✓
R3 *Direct Access*	✗
R4 *Verifier-Centric*	✗
R5 *Metadata Access*	✓/?
R6 *Unhandled Patterns*	✗

Requirements

	LLVM
R1 *Formal*	✓
R2 *Configurable*	✓
R3 *Direct Access*	✗
R4 *Verifier-Centric*	✗
R5 *Metadata Access*	✓/?
R6 *Unhandled Patterns*	✗

SSA

Requirements

	LLVM
R1 *Formal*	✓
R2 *Configurable*	✓
R3 *Direct Access*	✗
R4 *Verifier-Centric*	✗
R5 *Metadata Access*	✓/?
R6 *Unhandled Patterns*	✗

SSA

Best Effort

Requirements

	LLVM	Grammar-Based (Eclipse CDT, etc)
R1 *Formal*	✓	✗
R2 *Configurable*	✓	✗
R3 *Direct Access*	✗	✓
R4 *Verifier-Centric*	✗	✗
R5 *Metadata Access*	✓/?	✓
R6 *Unhandled Patterns*	✗	✗

SSA

Best Effort

Requirements

	LLVM	Grammar-Based (Eclipse CDT, etc)
R1 *Formal*	✗	✗
R2 *Configurable*	✓	✗
R3 *Direct Access*	✗	✓
R4 *Verifier-Centric*	✗	✗
R5 *Metadata Access*	✓/?	✓
R6 *Unhandled Patterns*	✗	✗

Implementation dependent

SSA

Best Effort

Requirements

	LLVM	Grammar-Based (Eclipse CDT, etc)	Goto-CC
R1 *Formal*	✓/✗	✗	✓
R2 *Configurable*	✓	✗	✗
R3 *Direct Access*	✗	✓	✓
R4 *Verifier-Centric*	✗	✗	✗
R5 *Metadata Access*	✓/?	✓	✓
R6 *Unhandled Patterns*	✗	✗	✓

Implementation dependent

SSA

Best Effort

Requirements

	LLVM	Grammar-Based (Eclipse CDT, etc)	Goto-CC
R1 *Formal*		✗	✓
R2 *Configurable*	✓	✗	✗
R3 *Direct Access*	✗	✓	✓
R4 *Verifier-Centric*	✗	✗	✗
R5 *Metadata Access*	✓/?		✓
R6 *Unhandled Patterns*	✗	✗	✓

Implementation dependent

SSA

Best Effort

No optimization, only verbose CFA

Experimental Evaluation

Is LLVM really that bad?

ANTLR

Goals of the Experiment

Goals of the Experiment

Given a mature frontend project (LLVM),

Goals of the Experiment

Given a mature frontend project (LLVM),

and a proof-of-concept prototype of the proposed workflow…

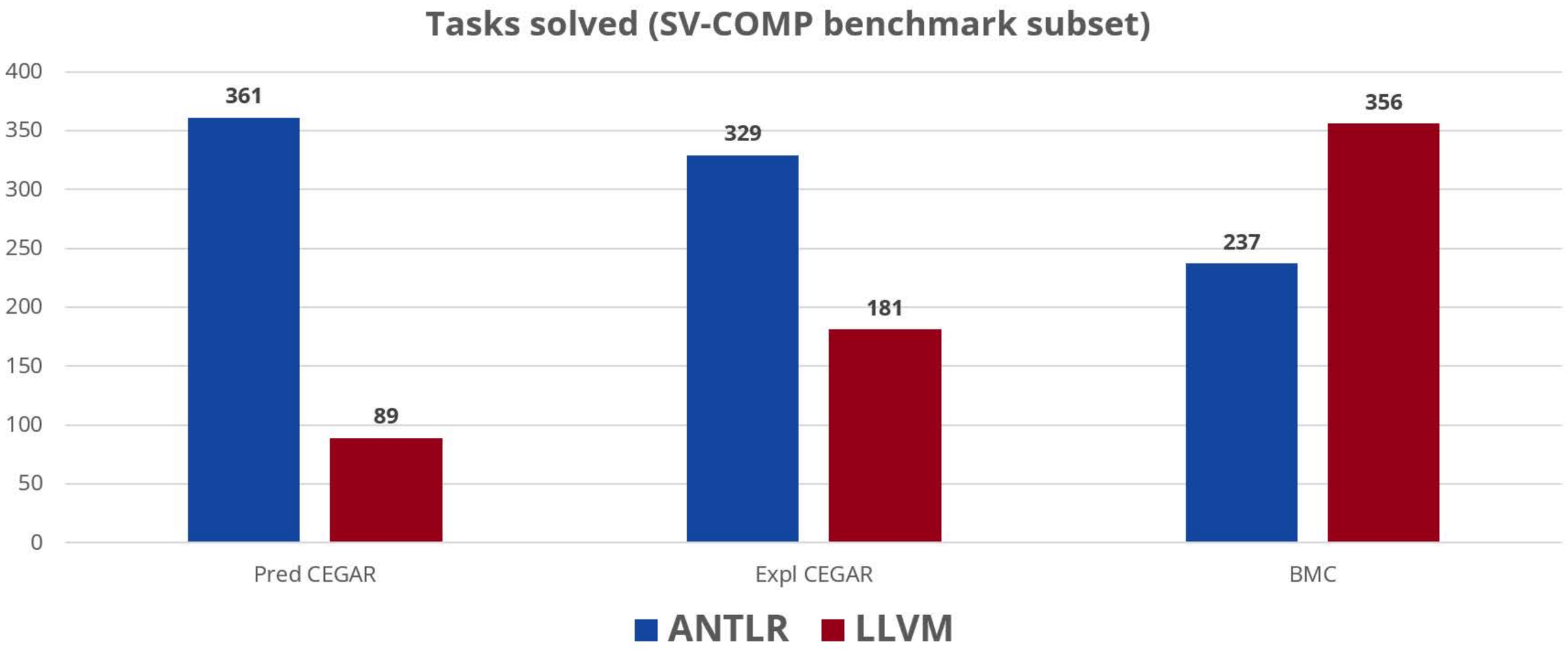
Goals of the Experiment

Given a mature frontend project (LLVM),

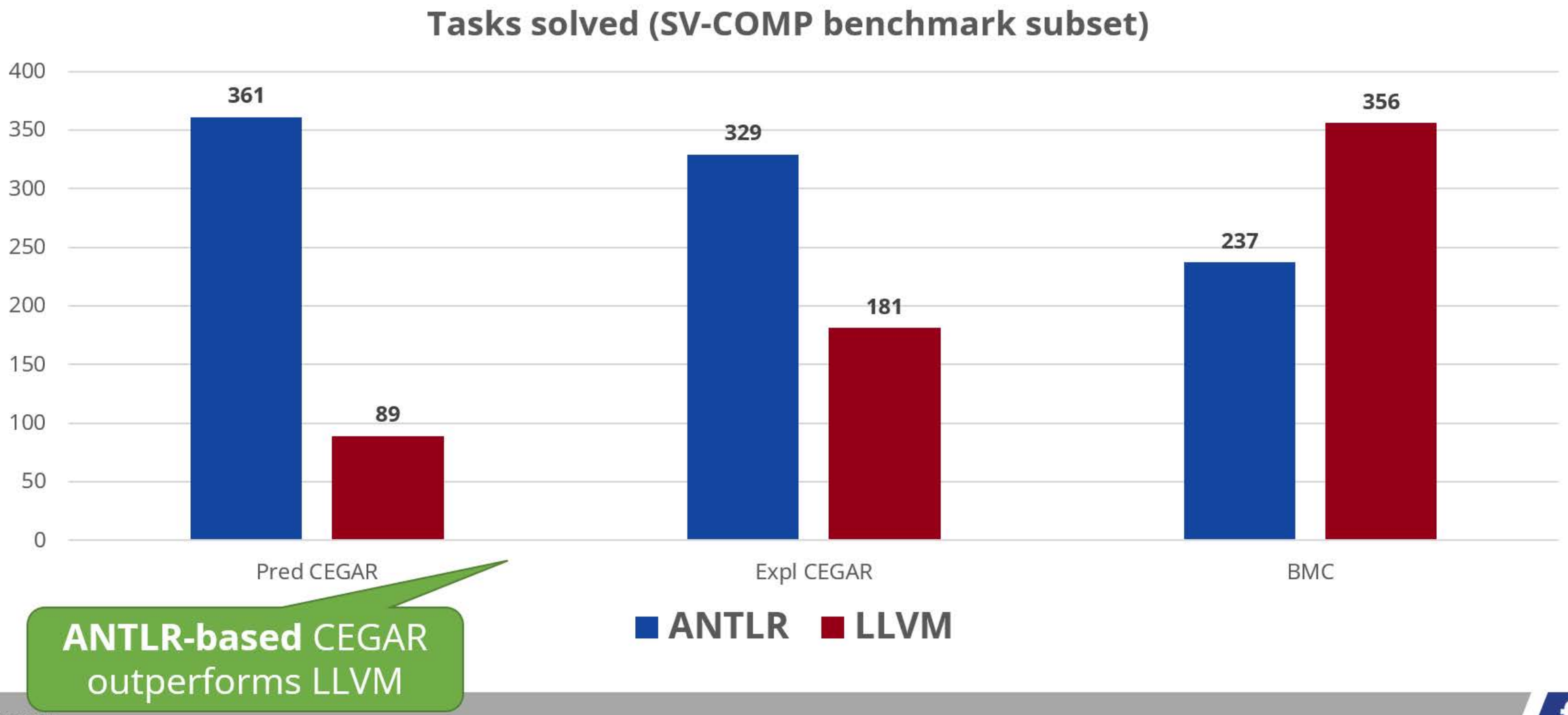
and a proof-of-concept prototype of the proposed workflow…

Can the latter outperform the former over CEGAR?

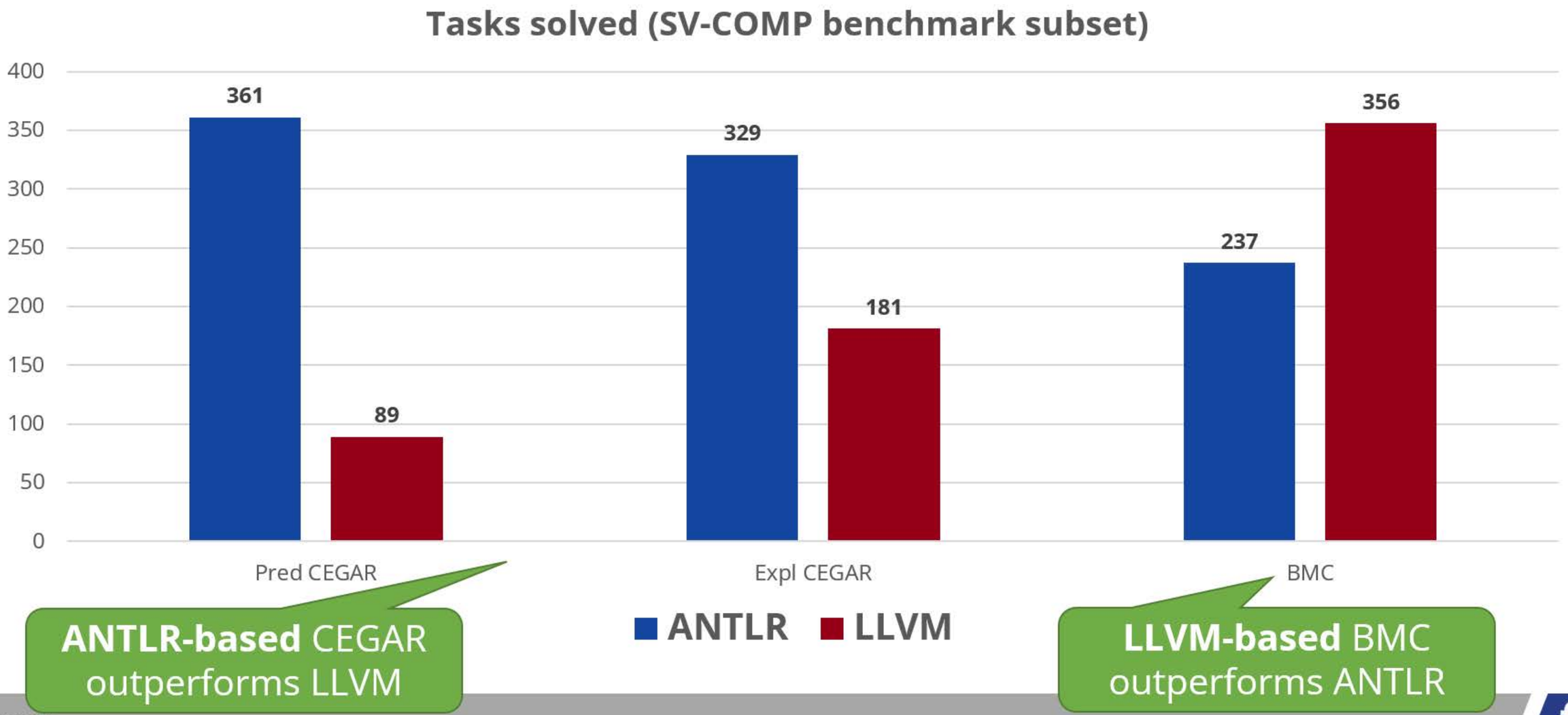
Results


Tasks solved (SV-COMP benchmark subset)
400
350
300
250
200
150
100
50
0
361
89
329
181
237
356
Pred CEGAR
Expl CEGAR
BMC
ANTLR
LLVM

Results


Tasks solved (SV-COMP benchmark subset)
400
350
300
250
200
150
100
50
0
361
89
329
181
237
356
Pred CEGAR
Expl CEGAR
BMC
ANTLR
LLVM
ANTLR-based CEGAR outperforms LLVM

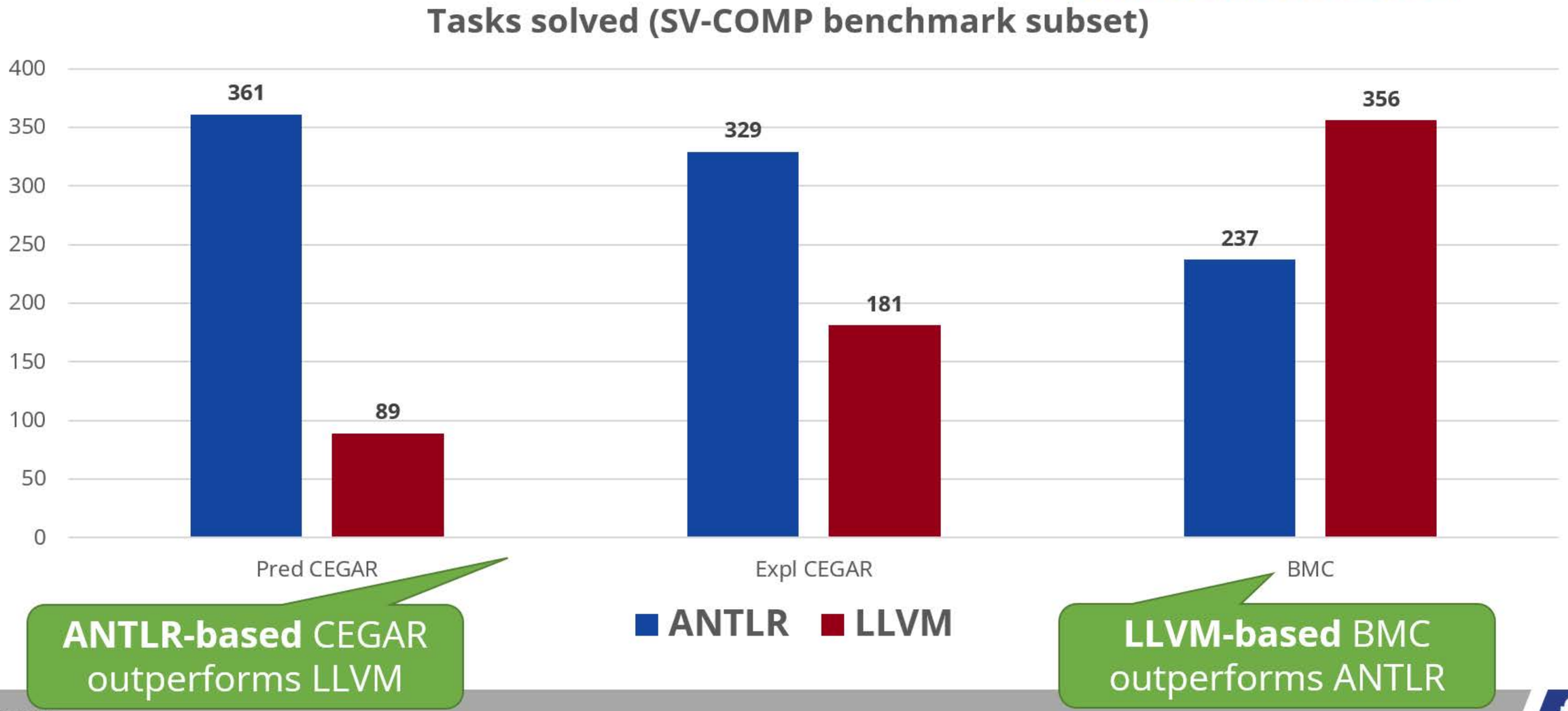
Results


Tasks solved (SV-COMP benchmark subset)
400
350
300
250
200
150
100
50
0
361
89
329
181
237
356
Pred CEGAR
Expl CEGAR
BMC
ANTLR
LLVM
ANTLR-based CEGAR outperforms LLVM
LLVM-based BMC outperforms ANTLR

Results

10.5281/zenodo.5920382

FormaliSE Artifact Evaluation 2022 Accepted



ftsrg