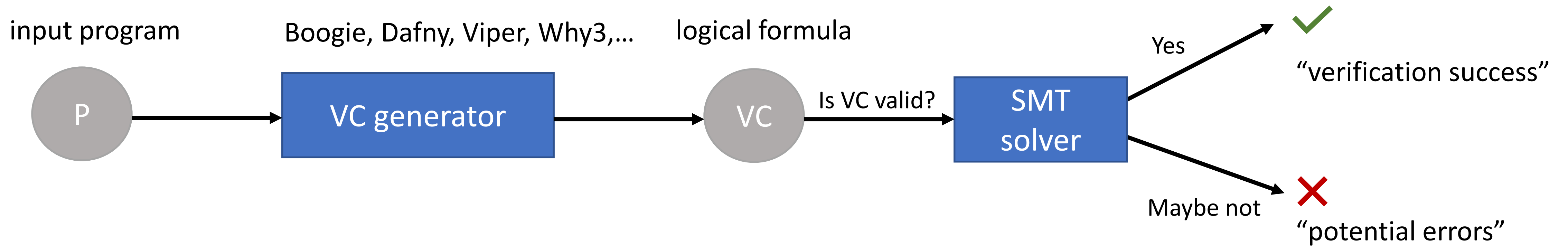
Making the Boogie Verifier Foundational

Gaurav Parthasarathy

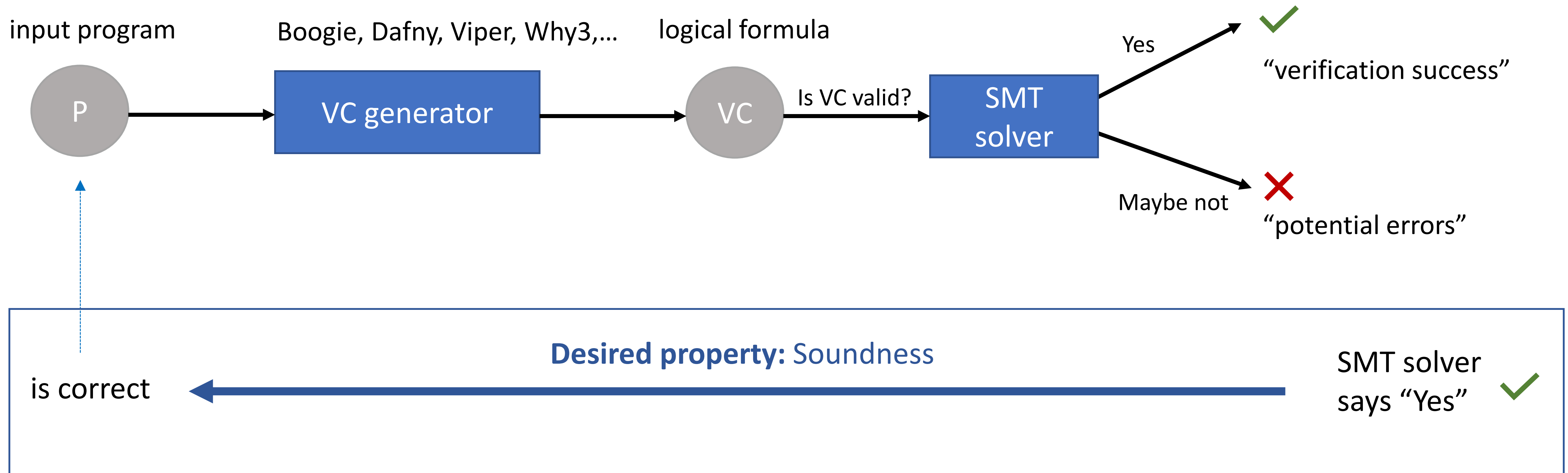
ETH zürich

Joint work with Lukas Himmelreich, Aleksandar Hubanov, Peter Müller and Alex Summers

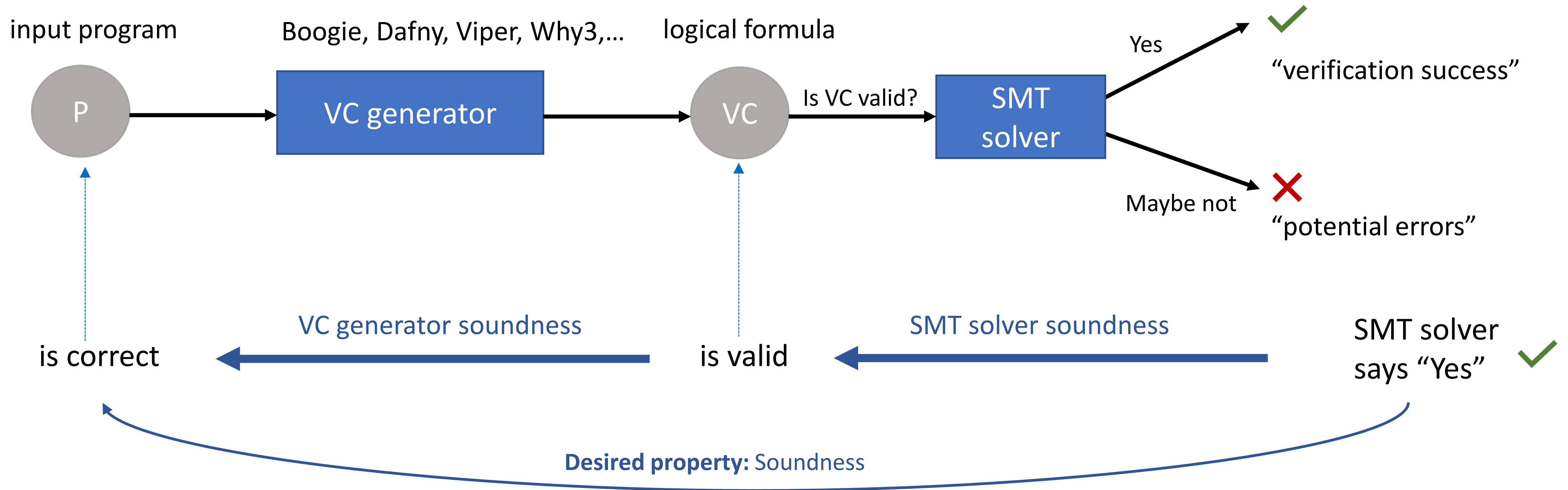
Verification Condition Generator



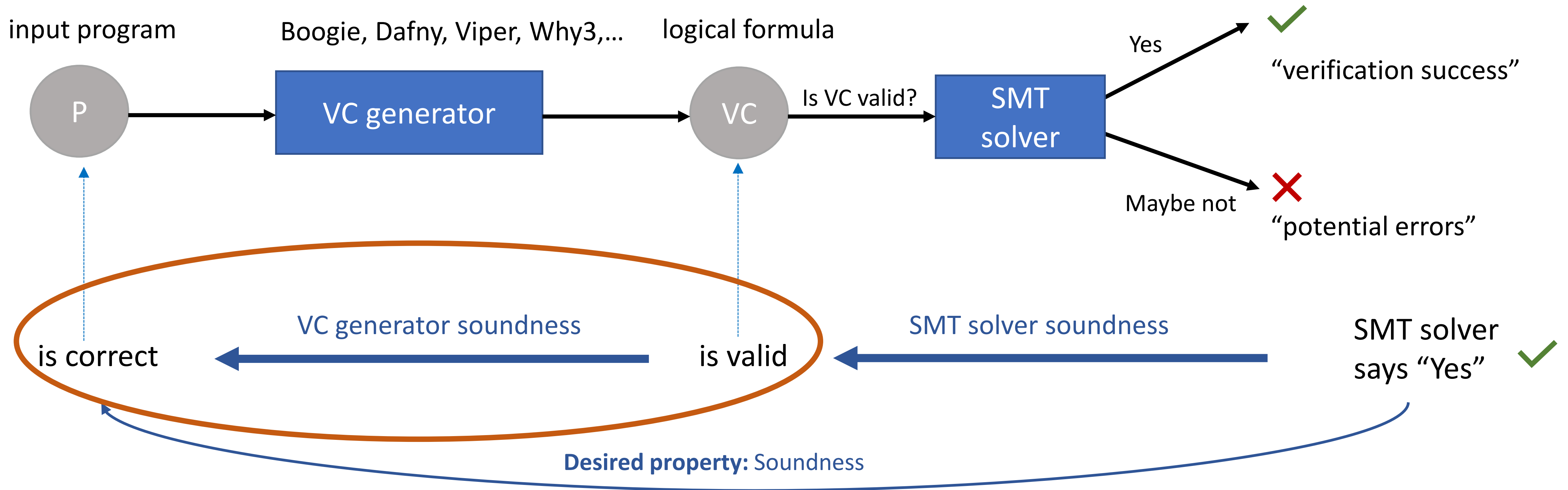
Verification Condition Generator



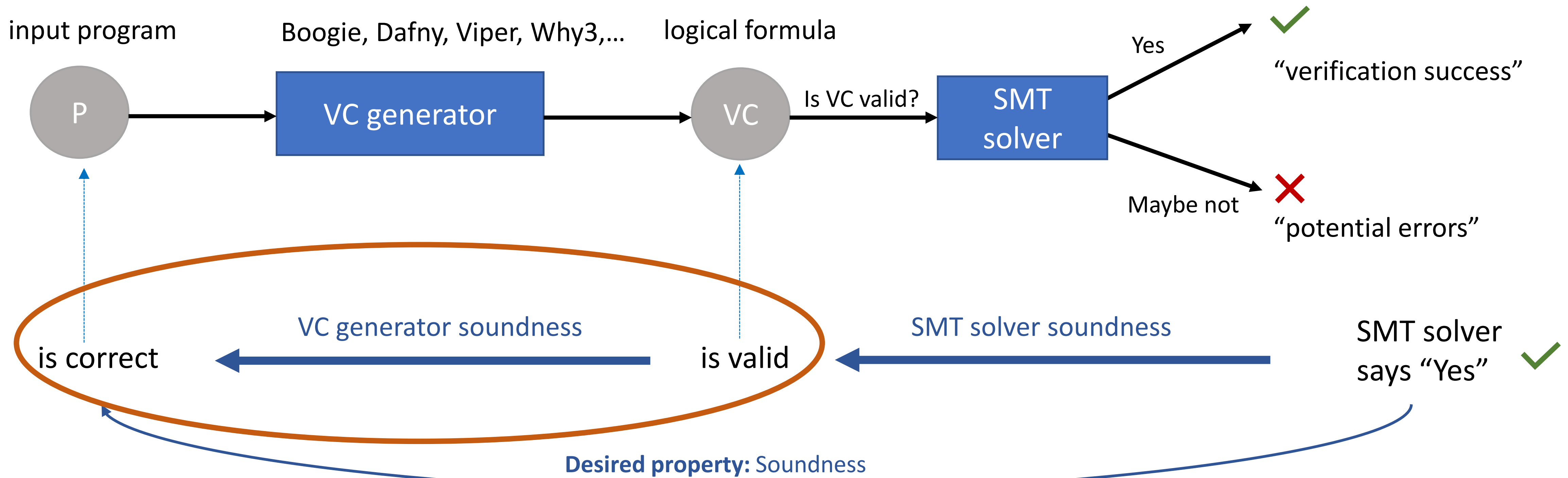
Verification Condition Generator



Verification Condition Generator



Verification Condition Generator



Key problem:
1. No formal guarantees on **implementations used in practice**
2. Nontrivial implementation consisting of many thousands of lines of code

Possible Approach: Prove Once and For All

Possible Approach: Prove Once and For All

Option 1: Prove Existing Implementation Correct

Problem: impractical since implementation languages of existing tools lack formalization

Possible Approach: Prove Once and For All

Option 1: Prove Existing Implementation Correct

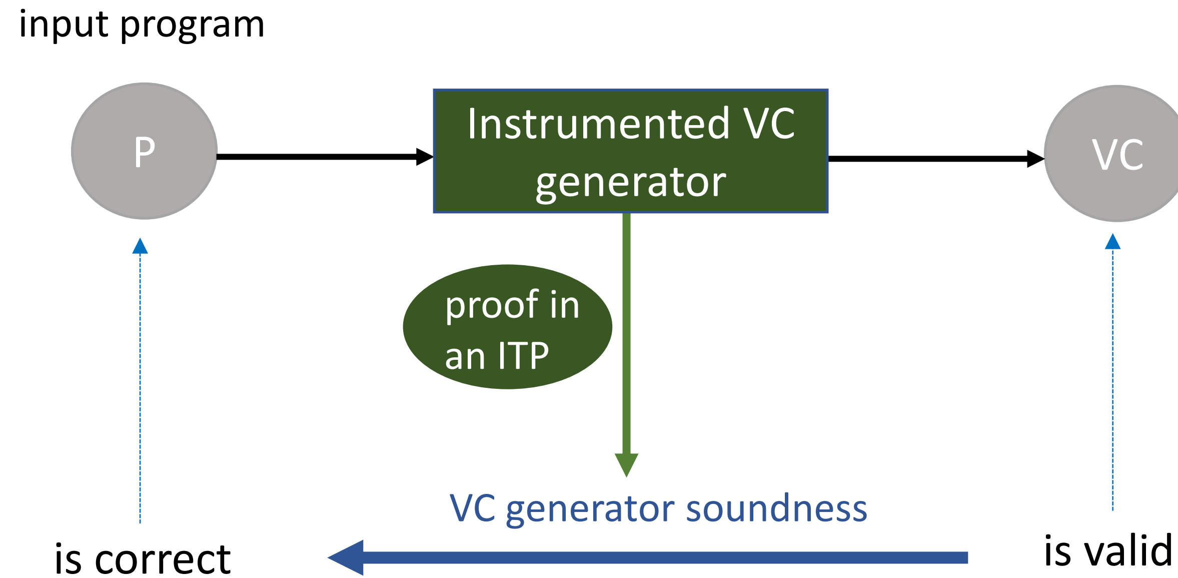
Problem: impractical since implementation languages of existing tools lack formalization

Option 2: Reimplement VC Generator in ITP (e.g., Coq, Isabelle)

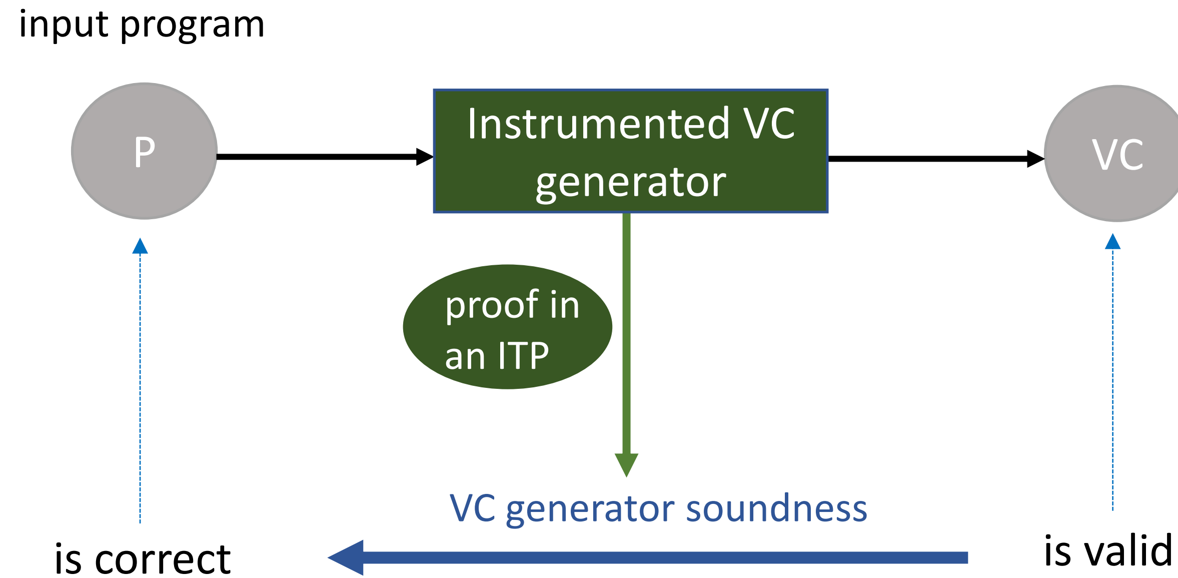
Problems:

- Lose benefits of modern programming languages
- Likely hard to convince developers to move to an ITP

Our Approach: Foundational Per-Run Validation



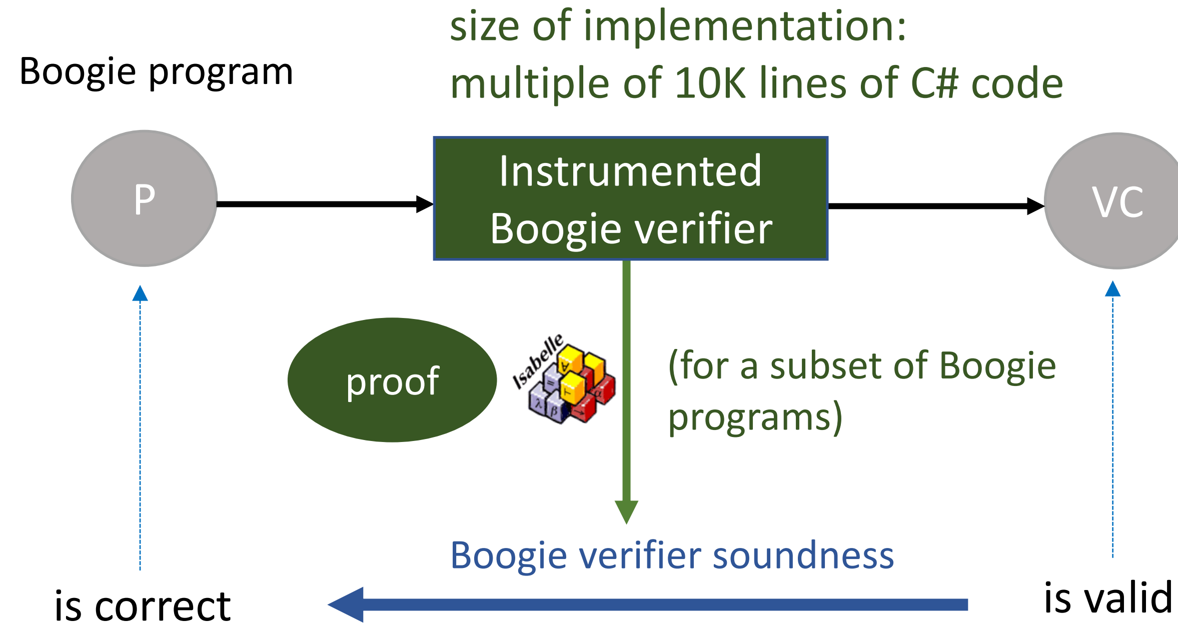
Our Approach: Foundational Per-Run Validation



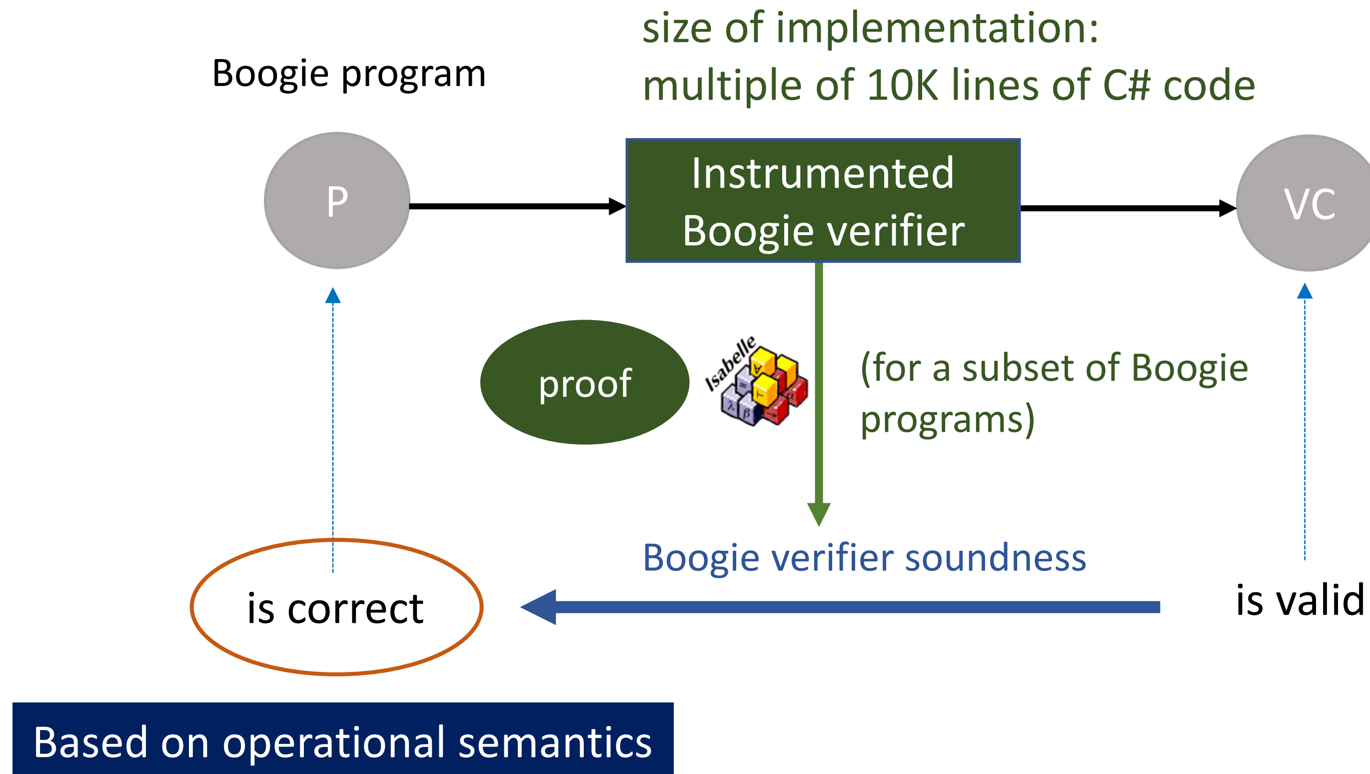
Advantages

1. Can reuse existing implementation
2. Doing a proof for a concrete instance is often easier than doing so for all instances

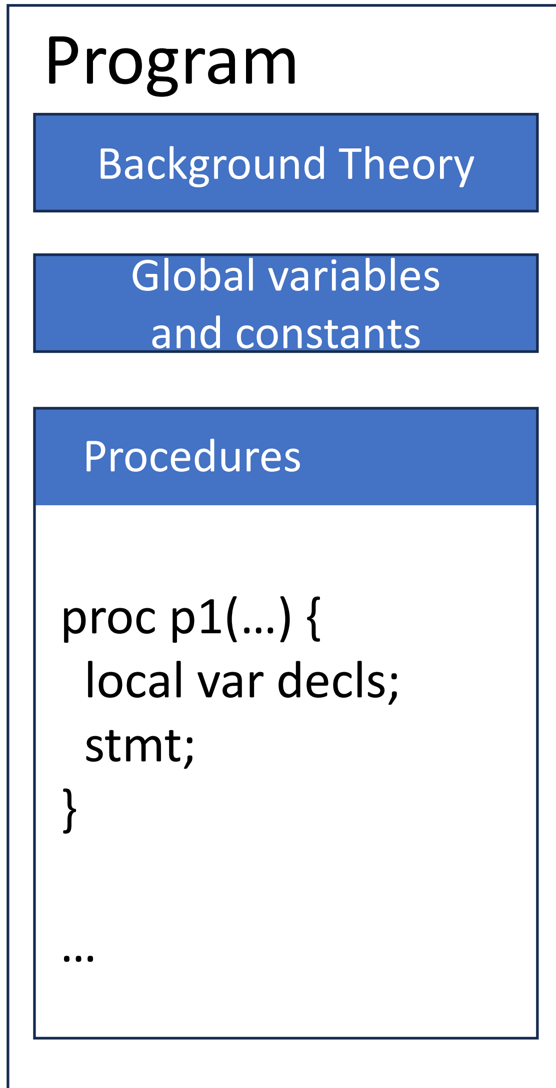
Our Approach: Foundational Per-Run Validation



Our Approach: Foundational Per-Run Validation

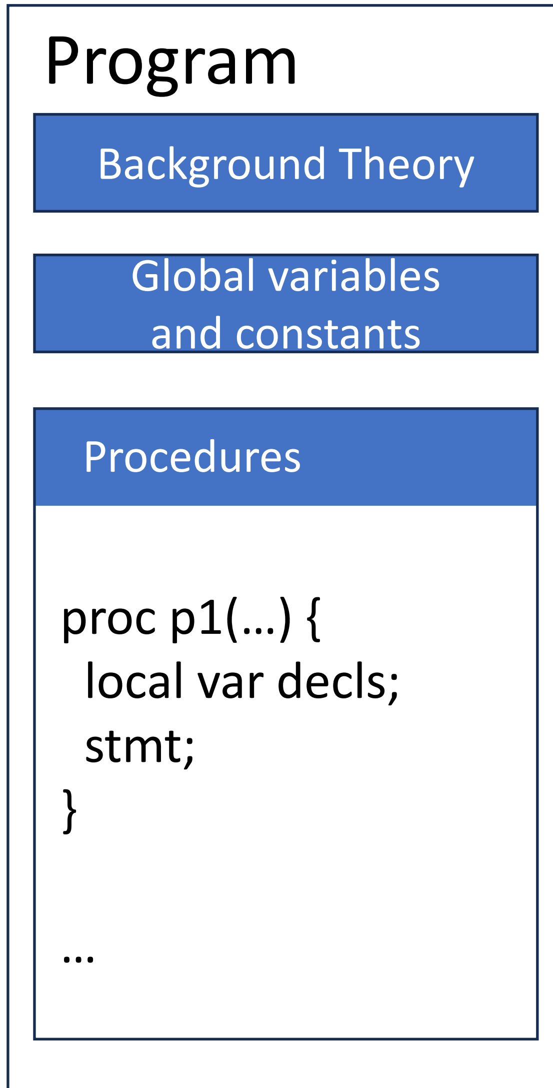


Boogie Program Structure



polymorphic functions, axioms, type constructors

Boogie Program Structure



polymorphic functions, axioms, type constructors

Main primitive statements:

`x := E` **`assert`** `E` **`assume`** `E` **`havoc`** `x`

Control flow:

`stmt; stmt` **`if`**`(*) { stmt }` **`else`** `{ stmt }`

`while``(E)` **`invariant`** `E { stmt }` **`break`** **`goto`**

Expressions:

function calls, value and type quantification, …

Semantics of a Boogie Program

```
procedure p() {
  var i:int;
  var j:int;

  assume i != 0;
  j := 0;

  while (i != 0)
    invariant j >= 0 && (i == 0 ==> j > 0);
  {
    j := j+1;
    i := i-1;
  }

  assert j > 0;
}
```

Semantics of a Boogie Program

```
procedure p() {
  var i:int;
  var j:int;

  assume i != 0;
  j := 0;

  while (i != 0)
    invariant j >= 0 && (i == 0 ==> j > 0);
  {
    j := j+1;
    i := i-1;
  }

  assert j > 0;
}
```

Consider all possible values for i and j

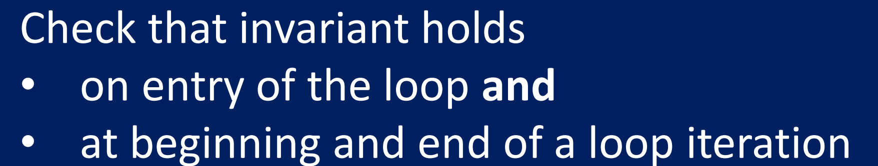
Semantics of a Boogie Program

```
procedure p() {
  var i:int;
  var j:int;

  assume i != 0;
  j := 0;

  while (i != 0)
    invariant j >= 0 && (i == 0 ==> j > 0);
  {
    j := j+1;
    i := i-1;
  }

  assert j > 0;
}
```

Prune executions

Semantics of a Boogie Program

```
procedure p() {
  var i:int;
  var j:int;

  assume i != 0;
  j := 0;

  while (i != 0)
    invariant j >= 0 && (i == 0 ==> j > 0);
  {
    j := j+1;
    i := i-1;
  }

  assert j > 0;
}
```

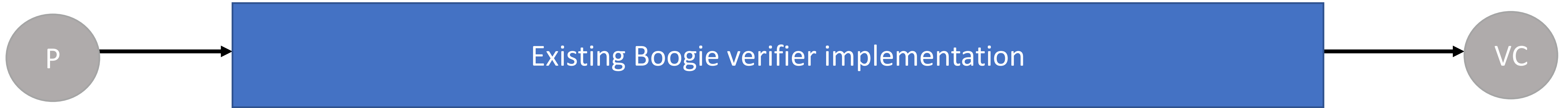


Semantics of a Boogie Program

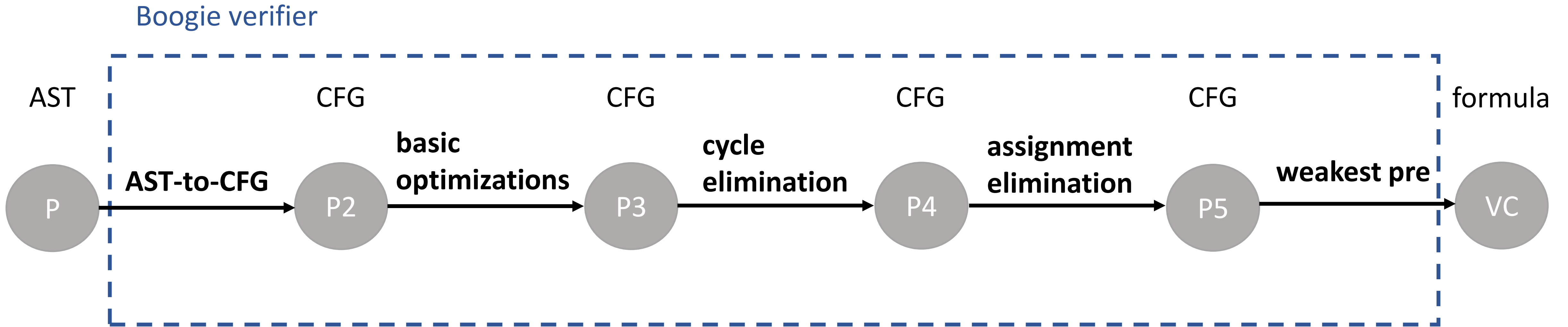
```
procedure p() {
  var i:int;
  var j:int;

  assume i != 0;
  j := 0;

  while (i != 0)
    invariant j >= 0 && (i == 0 ==> j > 0);
  {
    j := j+1;
    i := i-1;
  }

  assert j > 0;
}
```

Check that condition holds

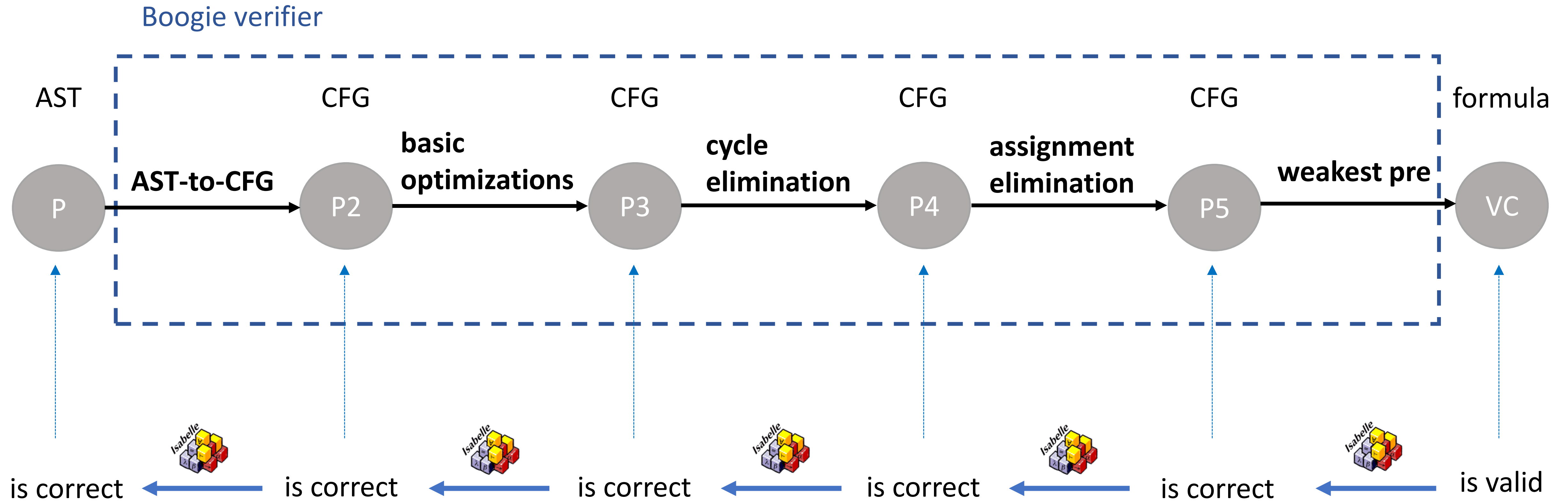
Boogie Verifier Implementation



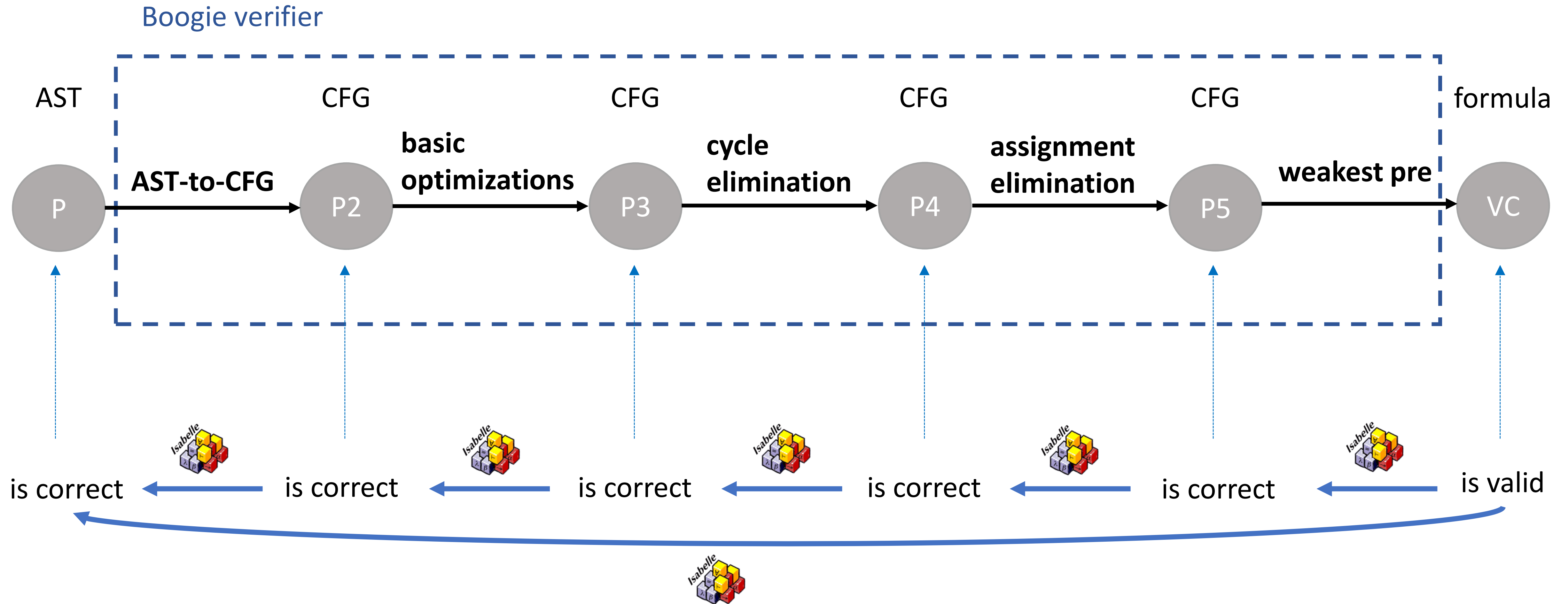
Boogie Verifier Implementation



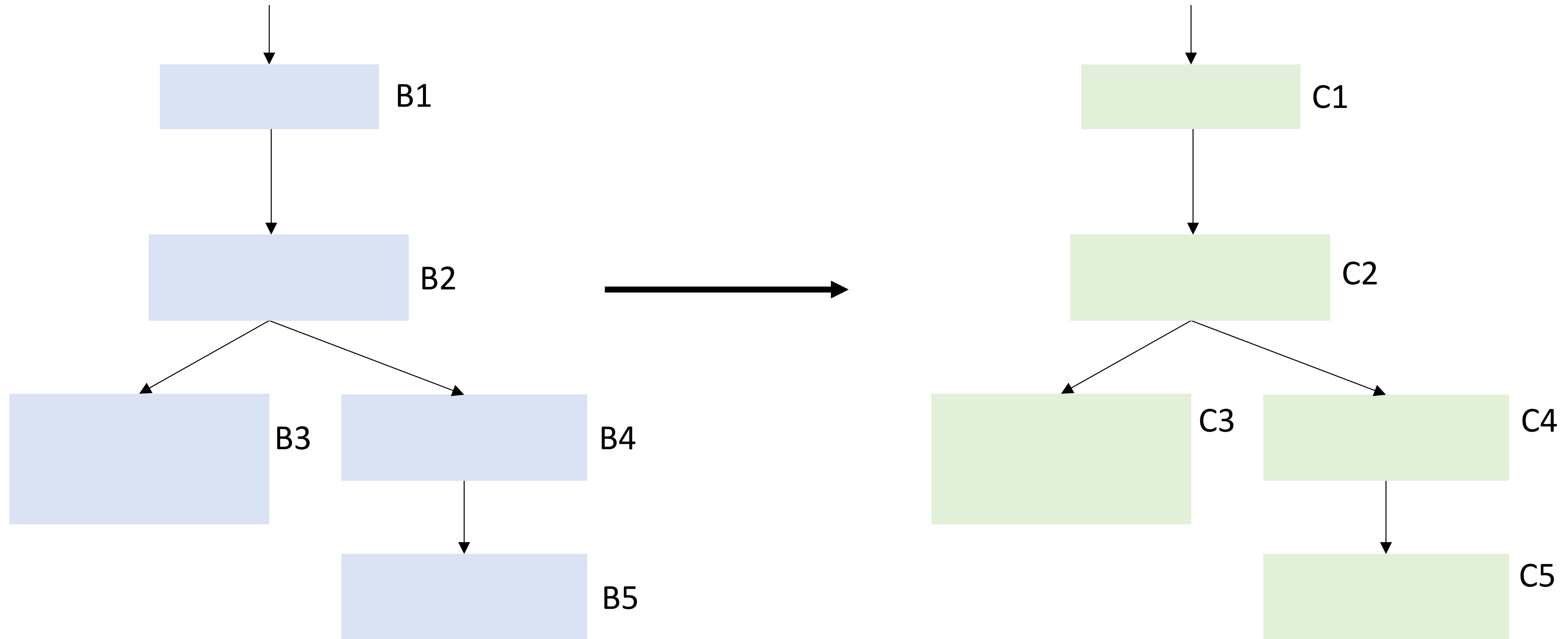
Boogie Verifier Implementation



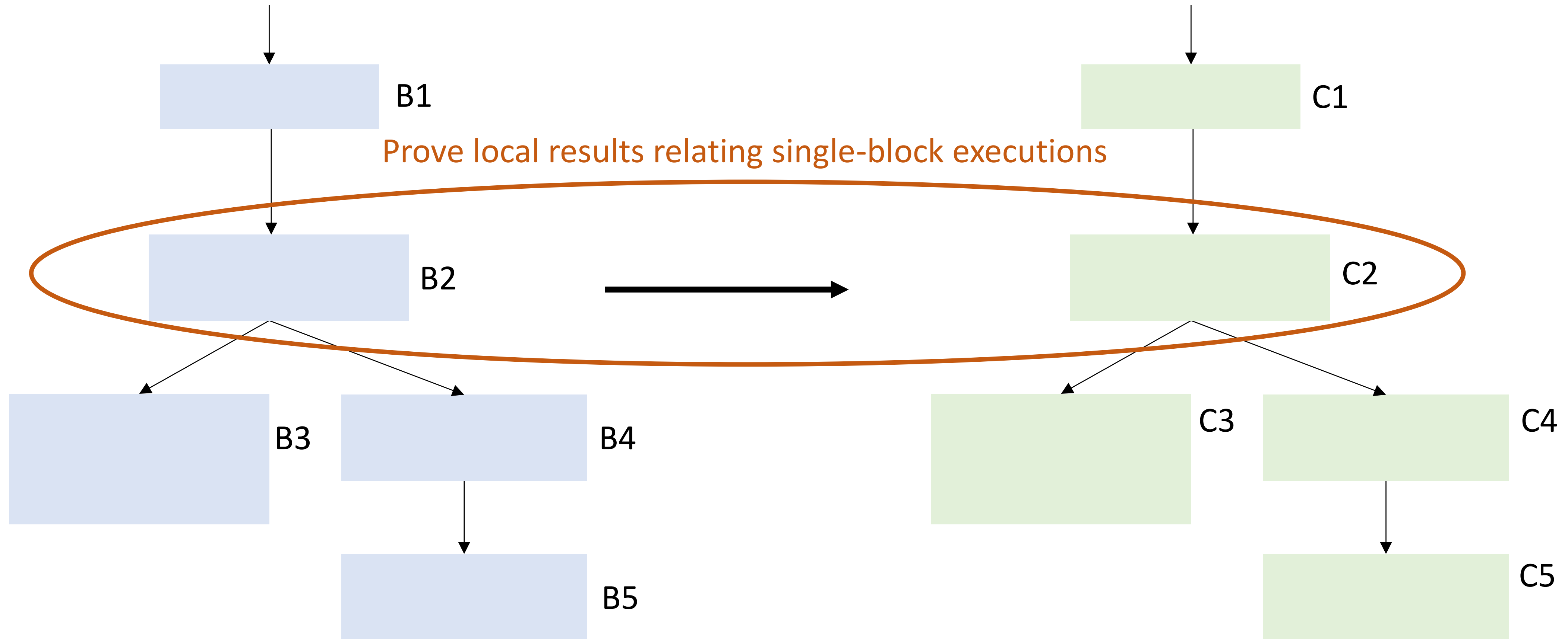
Boogie Verifier Implementation



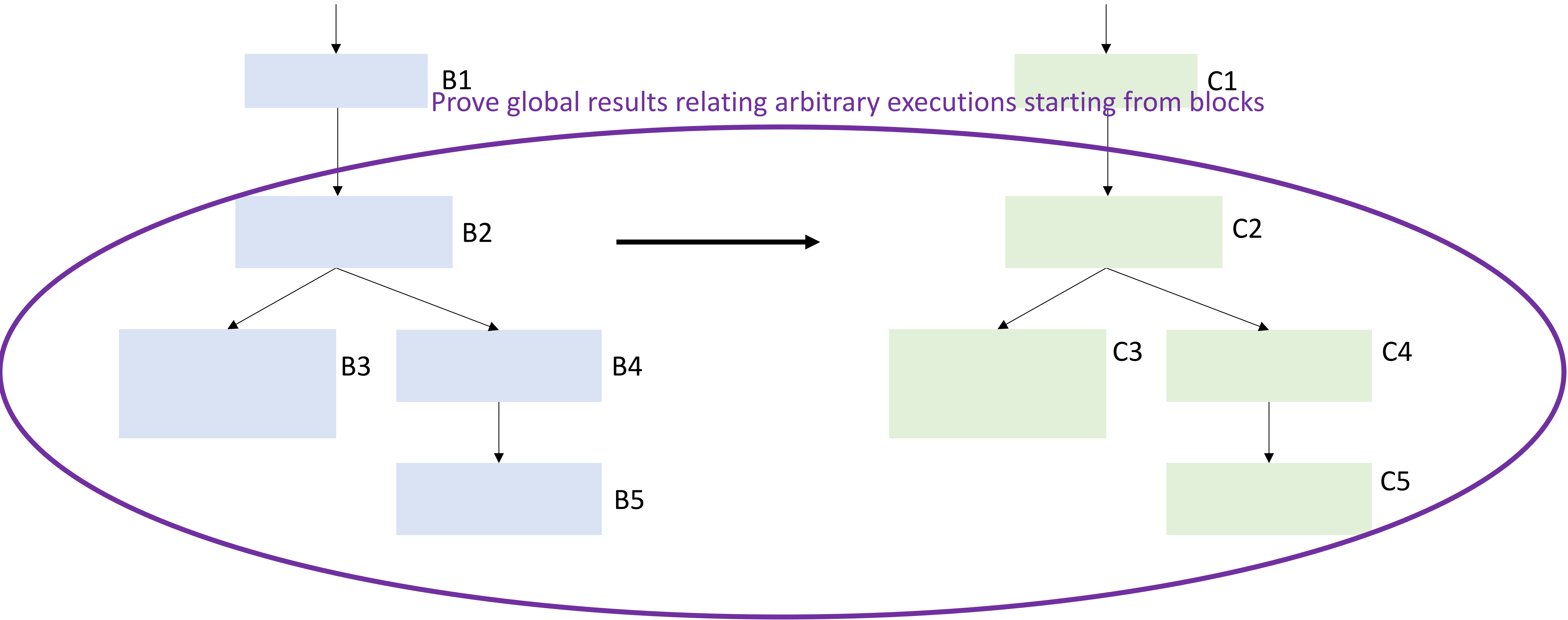
General Proof Generation Approach



General Proof Generation Approach



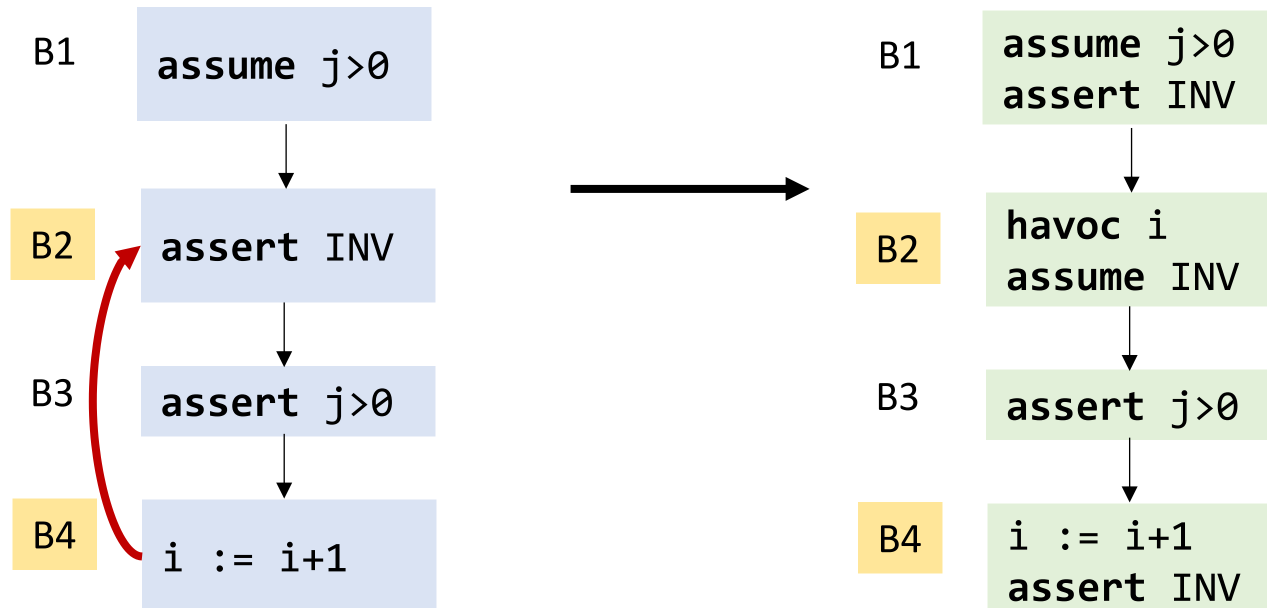
General Proof Generation Approach



Challenges for Proof Generation

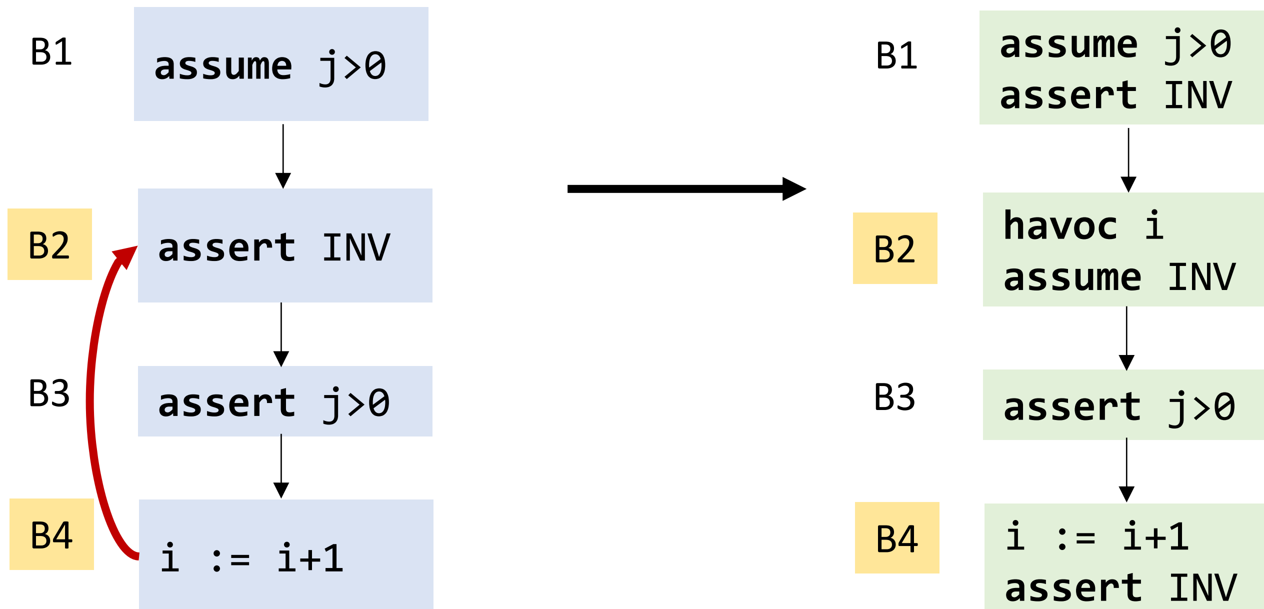
Challenges for Proof Generation

CFG Cycle elimination



Challenges for Proof Generation

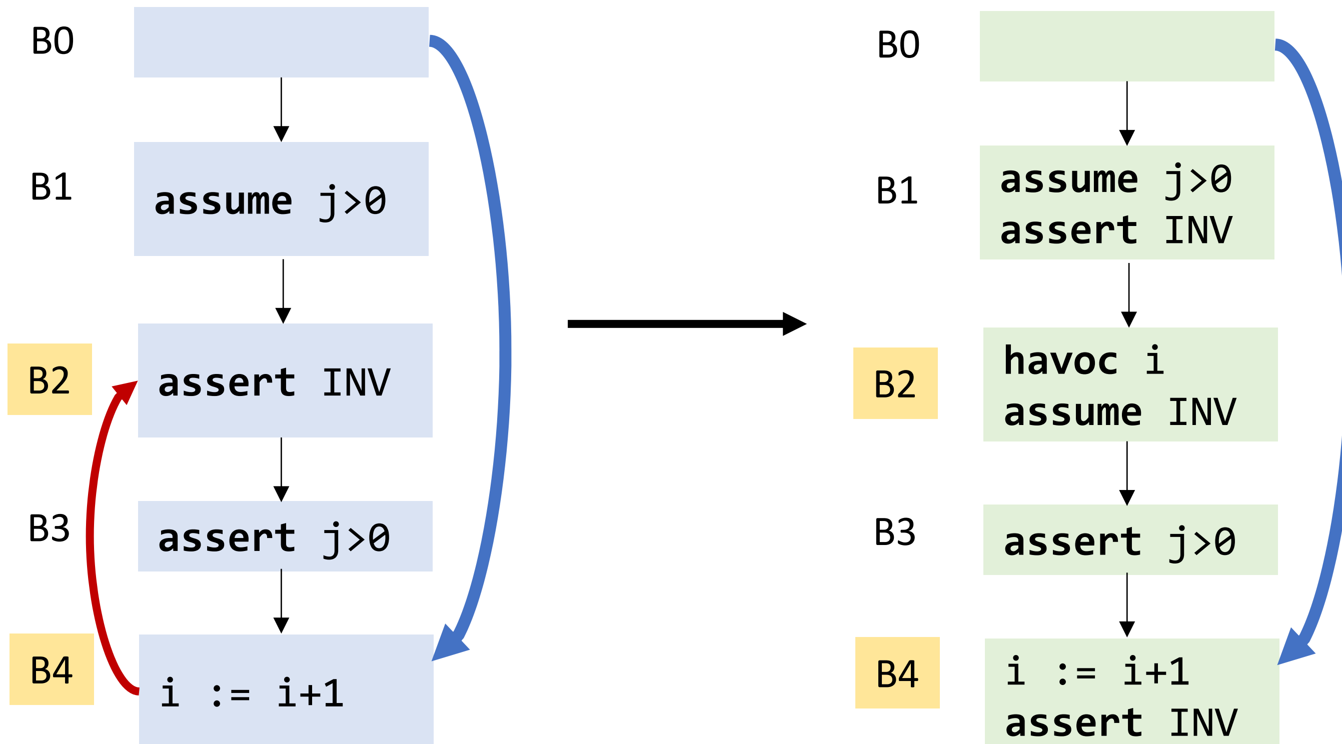
CFG Cycle elimination



Crucial property for proof:
Every execution that reaches B4 goes through B2
→ "B2 dominates B4"

Challenges for Proof Generation

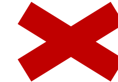
CFG Cycle elimination



Crucial property for proof:

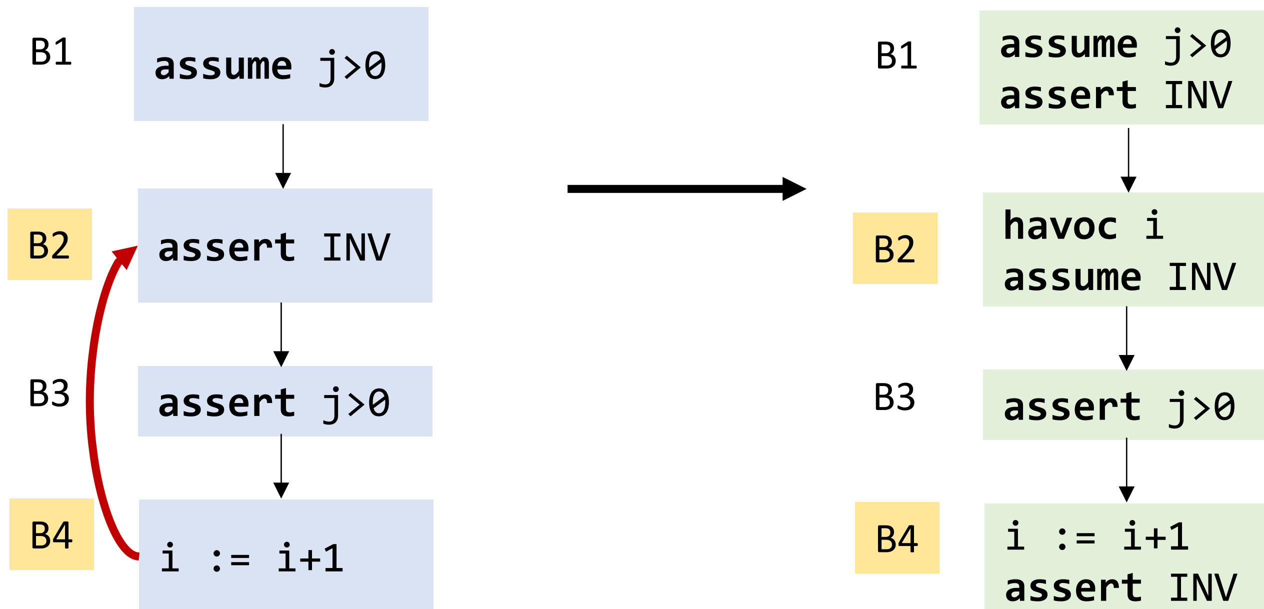
Every execution that reaches B4 goes through B2

→ "B2 dominates B4"



Challenges for Proof Generation

CFG Cycle elimination



Crucial property for proof:
Every execution that reaches B4 goes through B2
→ "B2 dominates B4"

Our generated proof does not require an explicit notion of domination

Challenges for Proof Generation

Assignment elimination

Proof relies on global property
→ how to express local results?

Challenges for Proof Generation

Assignment elimination

Proof relies on global property
→ how to express local results?

Weakest precondition generation

Encoding of type system

Concrete Numbers for Generated Proofs

Boogie Program		**Generated Isabelle Proof**	
File	LOC	LOC	Time to check [s]
MaxOfArray	22	2463	22.6
Plateau	50	2504	26.0
DutchFlag	76	4763	65.0
…	…	…	…

Overhead incurred by the generation of proofs is negligible

Conclusion

More details in CAV21 paper
"Formally Validating a Practical Verification Condition Generator"

Instrumented Boogie verifier
https://github.com/gauravpartha/boogie_proofgen/

Boogie formalization
https://github.com/gauravpartha/foundational_boogie/

Future work: Extend subset

- Boogie maps
- gotos and breaks in the AST-to-CFG phase
- dead variable elimination