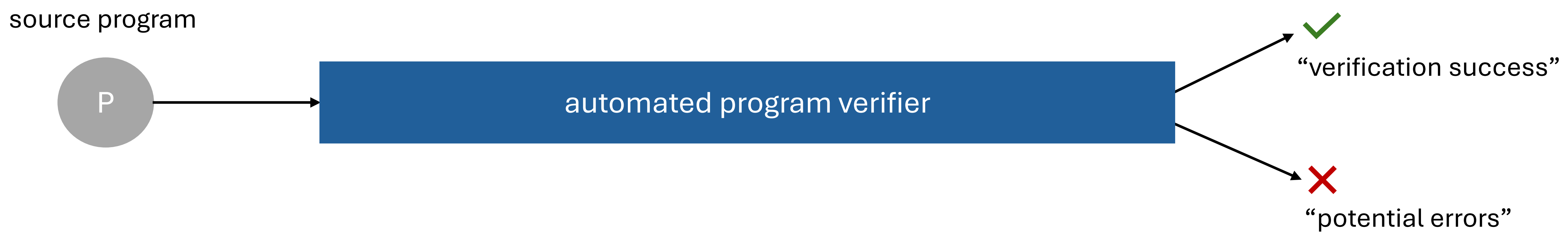
Towards Trustworthy Automated Program Verifiers: Formally Validating Translations into an Intermediate Verification Language

Gaurav Parthasarathy[1], Thibault Dardinier[1], Benjamin Bonneau[3], Peter Müller[1] and Alexander J. Summers[2]

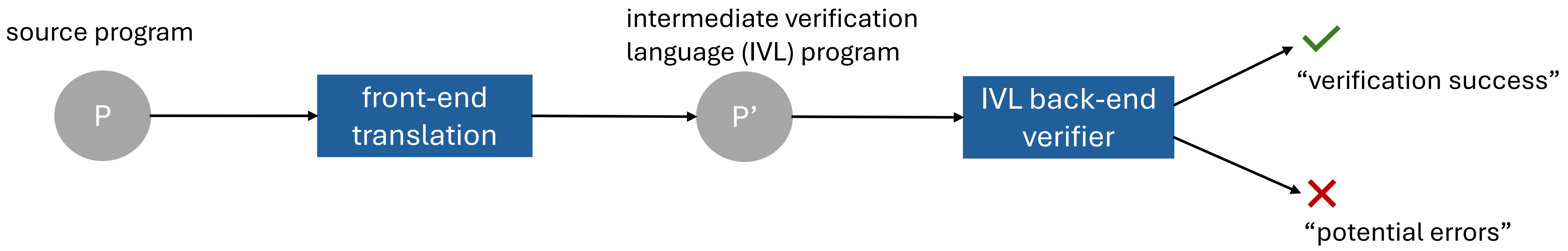
ETH zürich [1] UBC [2] Verimag [3]

PLDI 2024

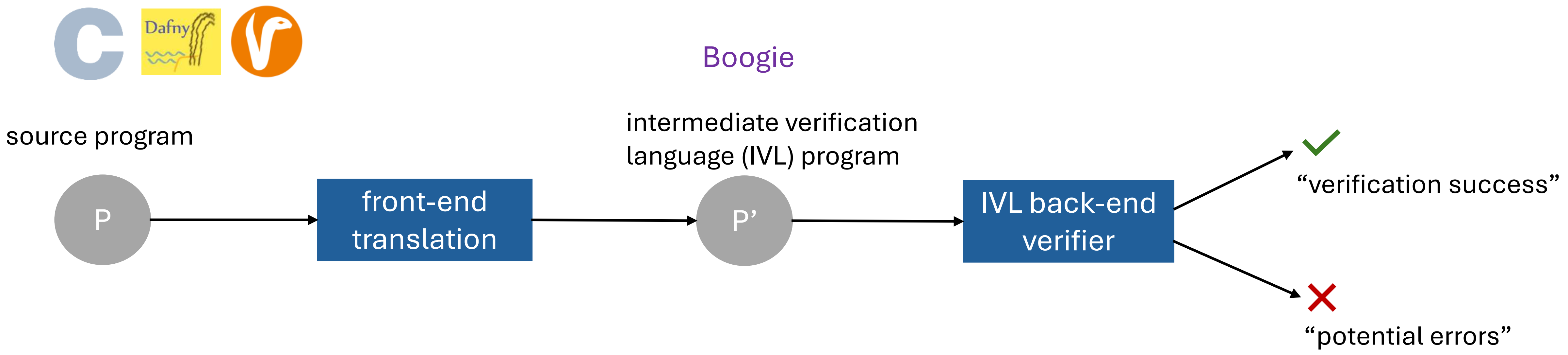
Automated Program Verifiers



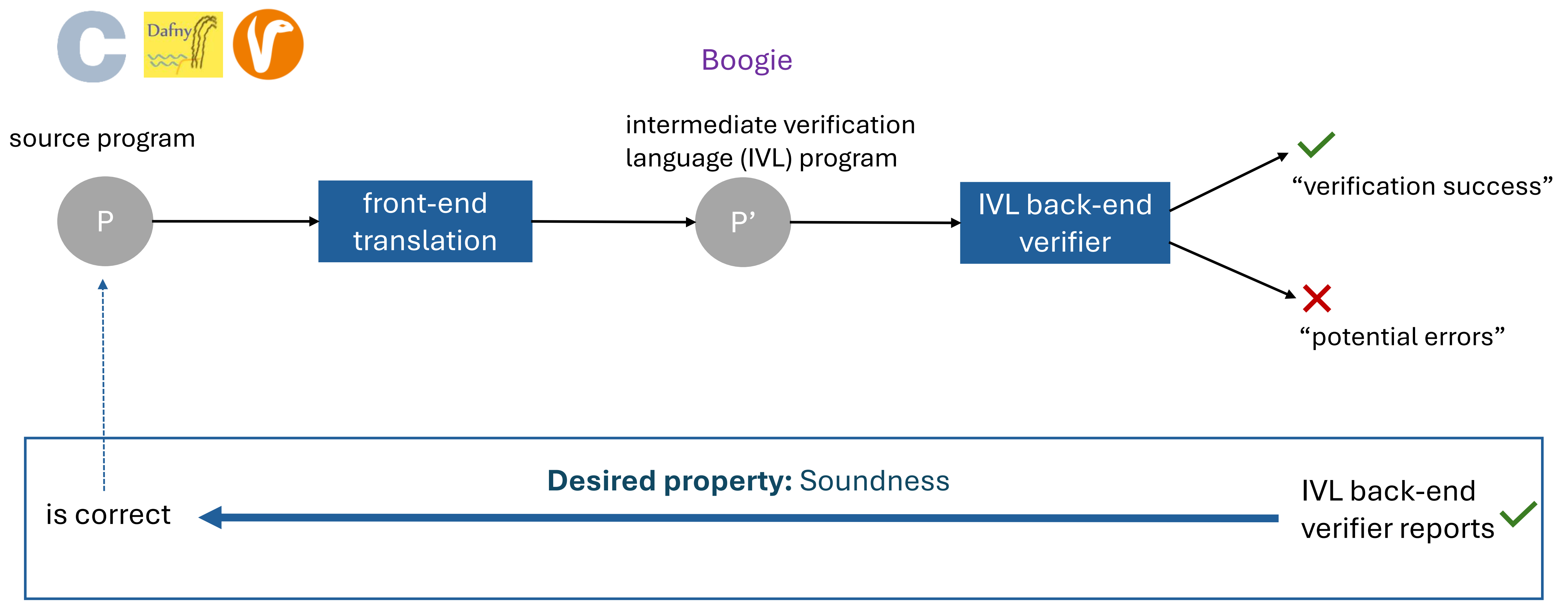
Translational Automated Program Verifiers



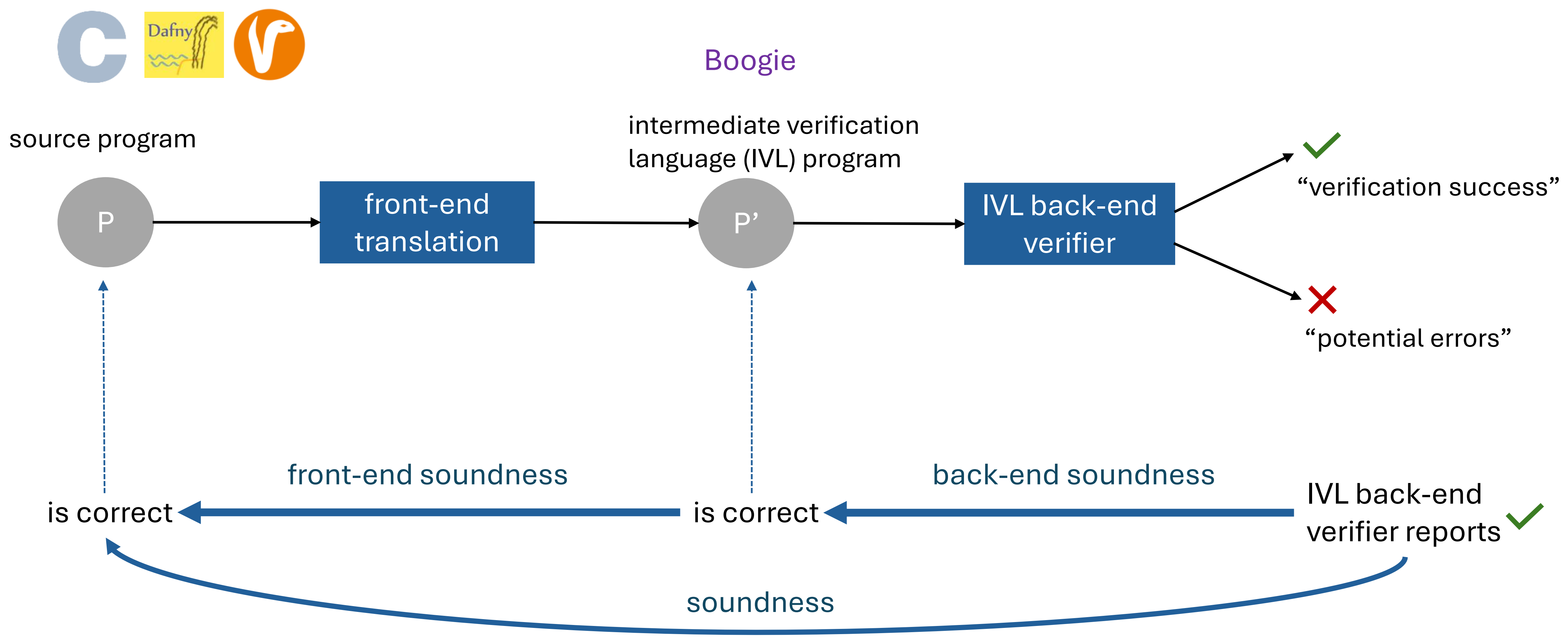
Translational Automated Program Verifiers



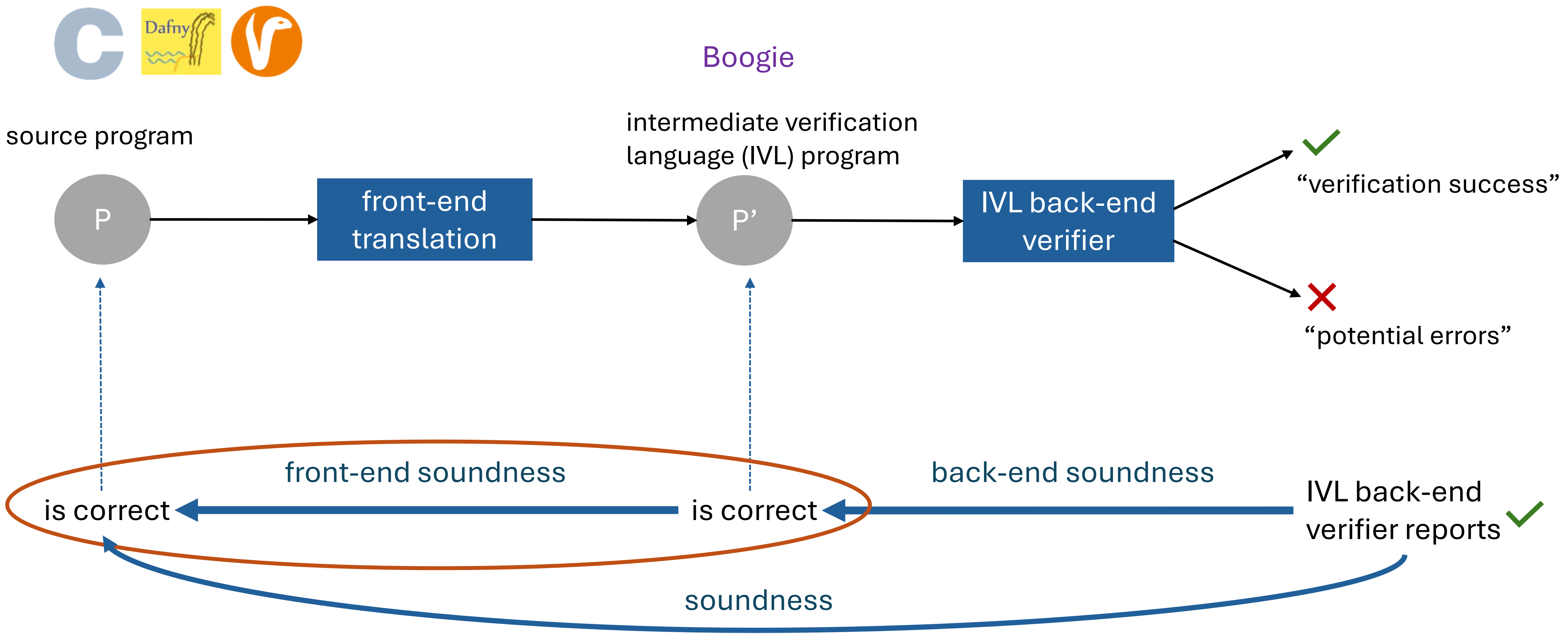
Translational Automated Program Verifiers



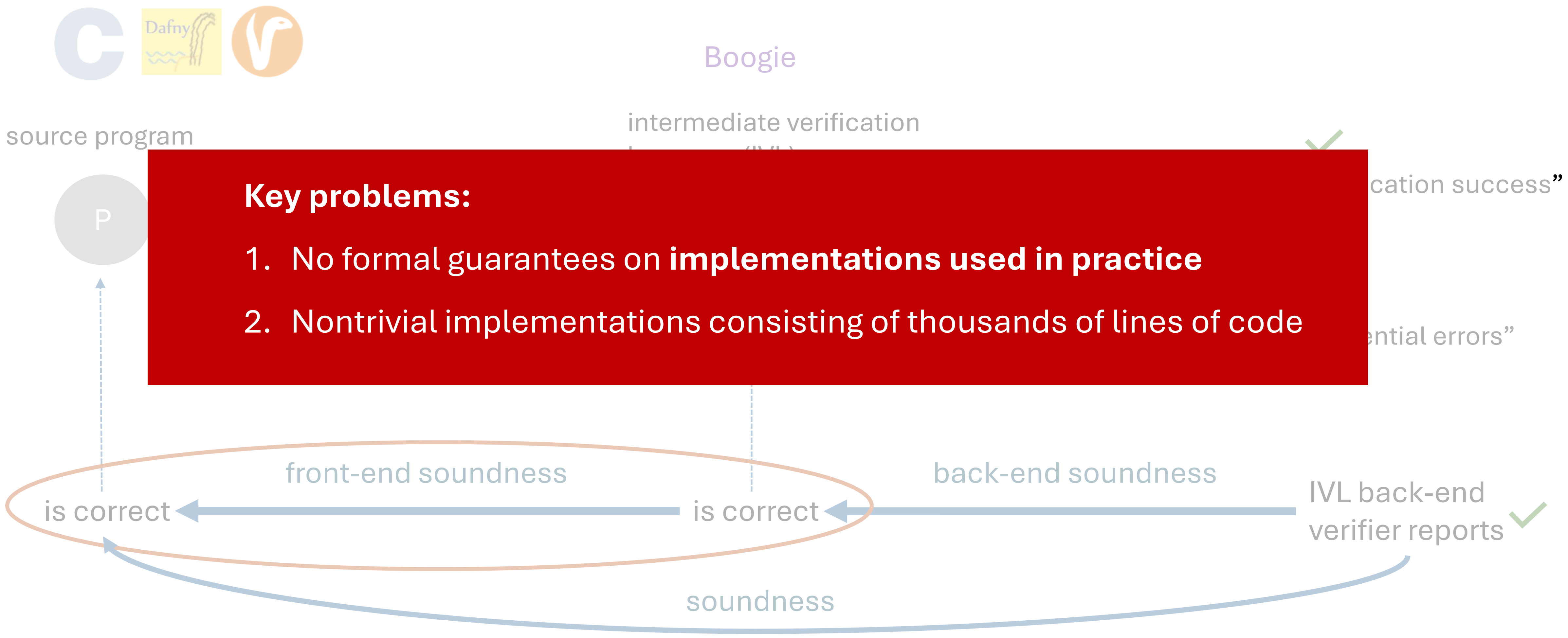
Translational Automated Program Verifiers



Translational Automated Program Verifiers



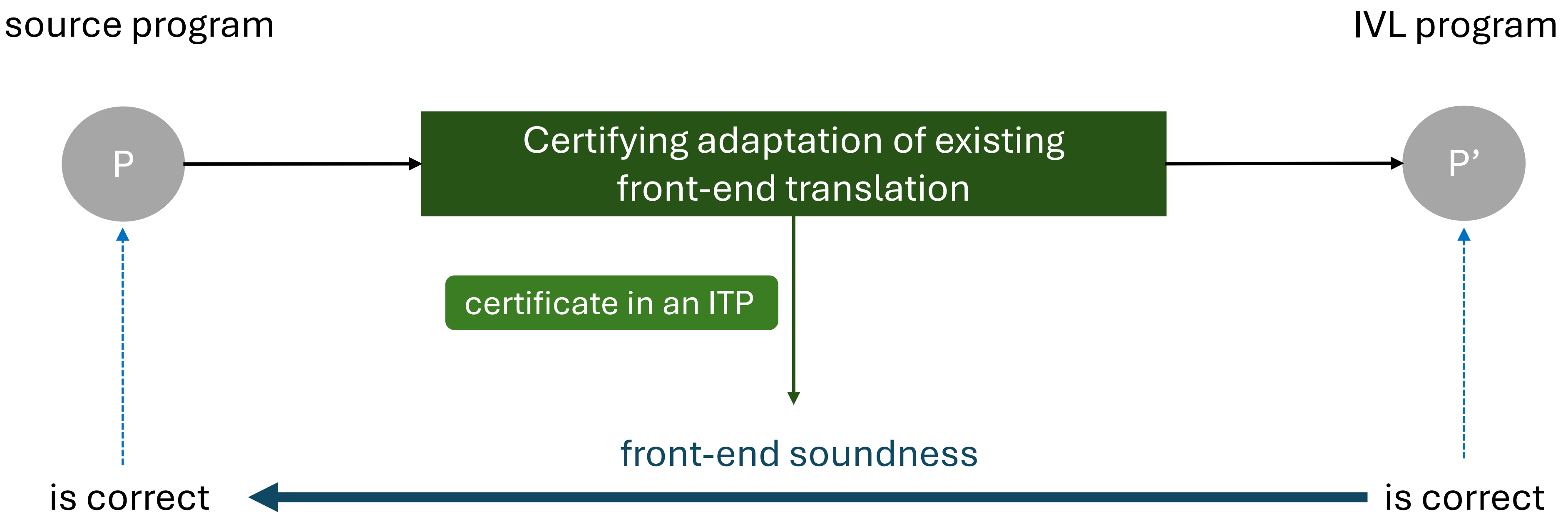
Translational Automated Program Verifiers



Key problems:

1. No formal guarantees on **implementations used in practice**
2. Nontrivial implementations consisting of thousands of lines of code

Our Approach: Foundational Per-Run Certification

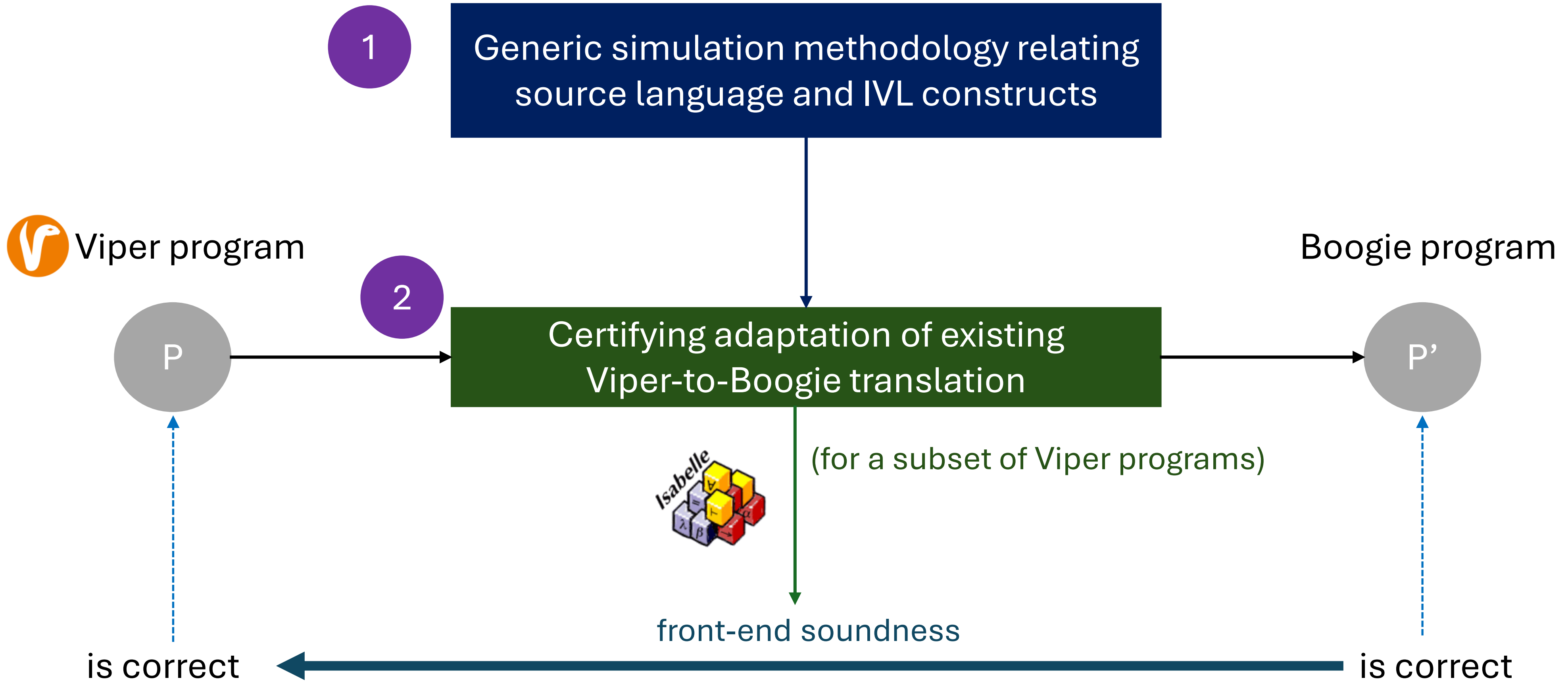


Provides formal guarantees to **existing** implementations

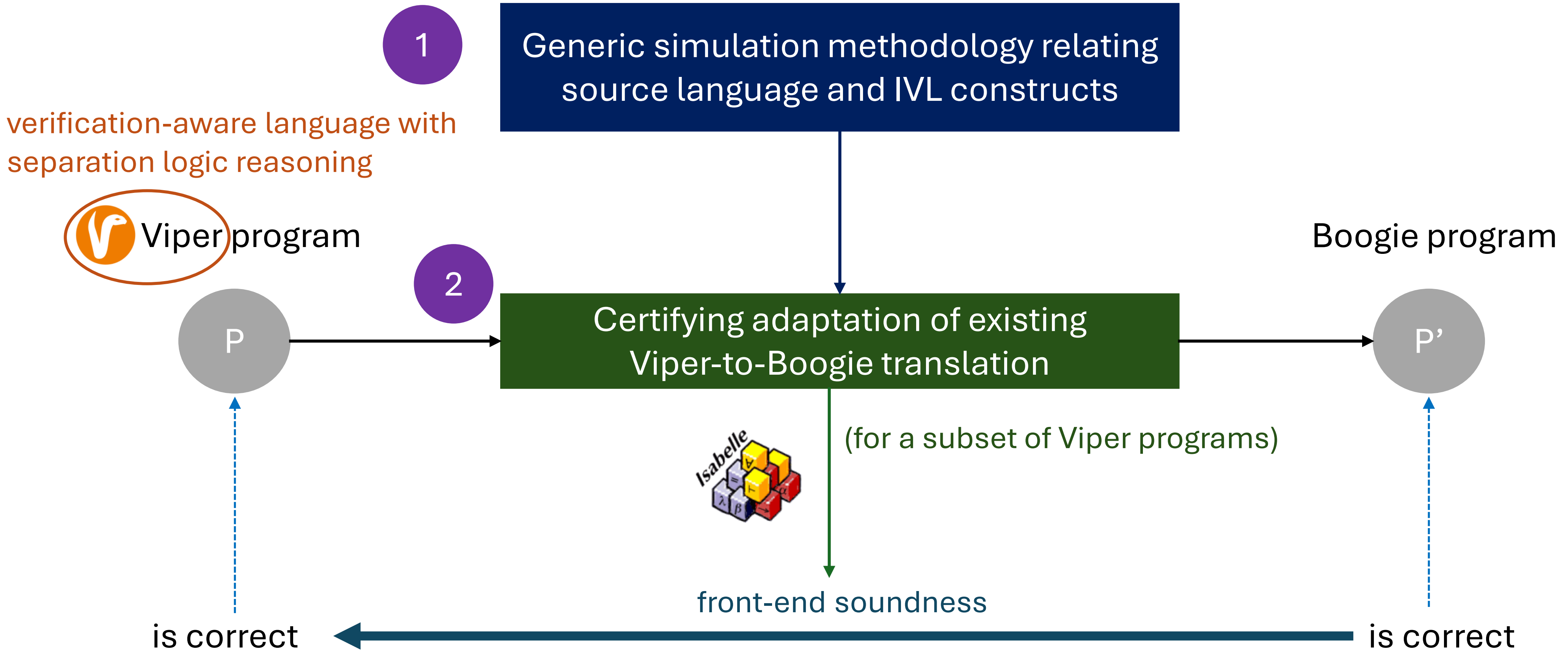
Key Contributions

1 Generic simulation methodology relating source language and IVL constructs

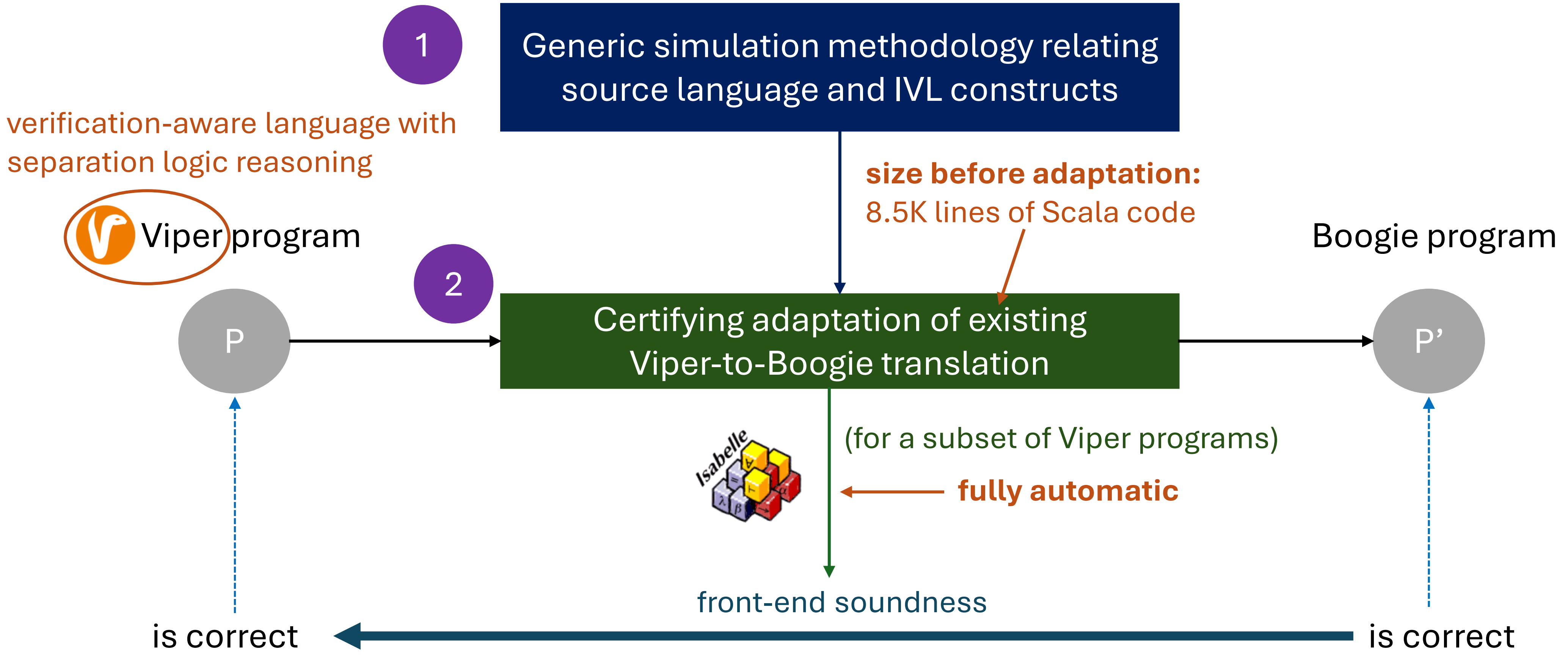
Key Contributions


1
Generic simulation methodology relating source language and IVL constructs
Viper program
2
P
Certifying adaptation of existing Viper-to-Boogie translation
Boogie program
P'
(for a subset of Viper programs)
Isabelle
front-end soundness
is correct
is correct

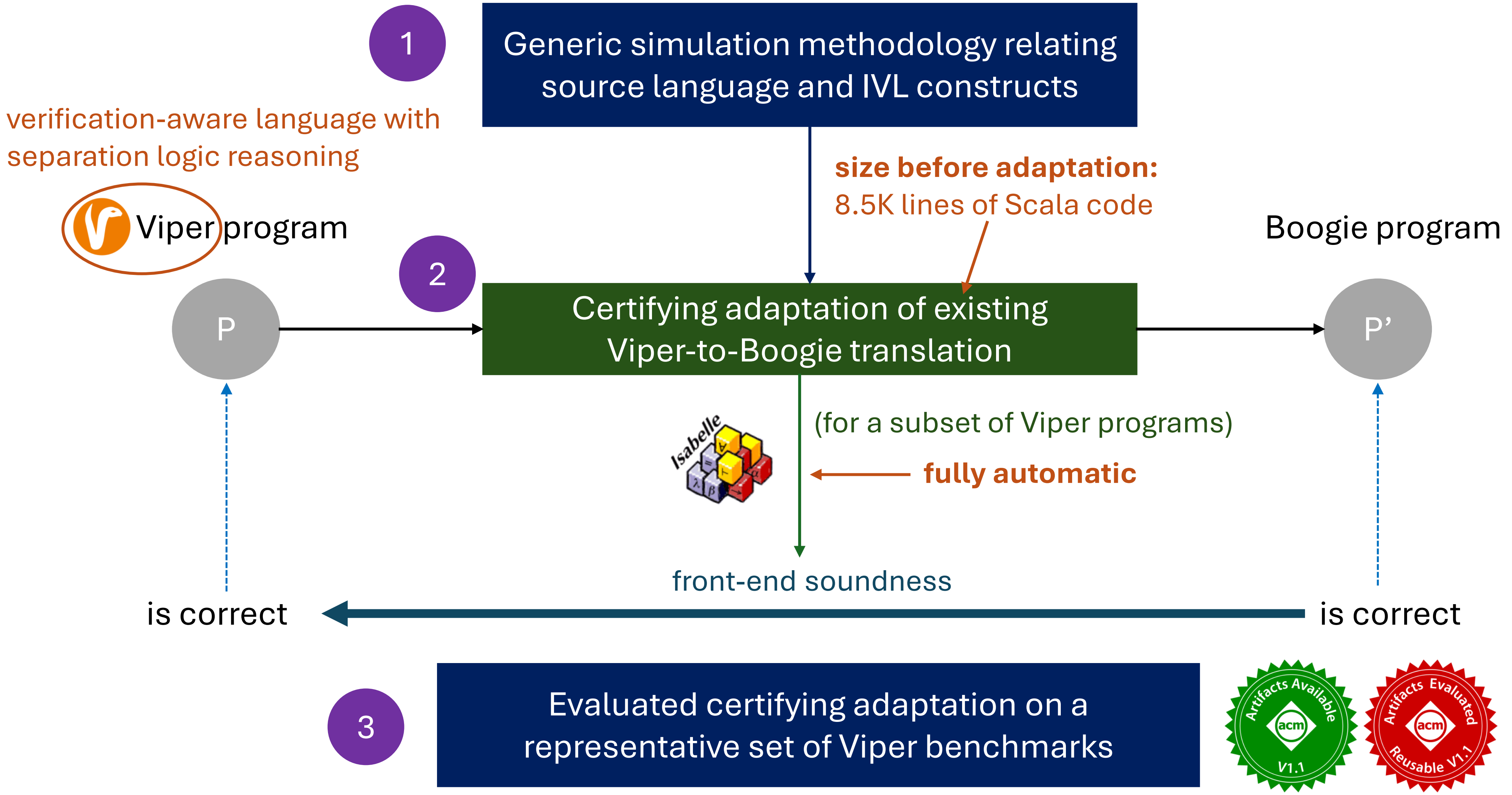
Key Contributions


1
Generic simulation methodology relating source language and IVL constructs
verification-aware language with separation logic reasoning
Viper program
Boogie program
2
P
Certifying adaptation of existing Viper-to-Boogie translation
P'
Isabelle
(for a subset of Viper programs)
front-end soundness
is correct
is correct

Key Contributions


1
Generic simulation methodology relating source language and IVL constructs
verification-aware language with separation logic reasoning
Viper program
size before adaptation:
8.5K lines of Scala code
Boogie program
2
P
Certifying adaptation of existing Viper-to-Boogie translation
P'
(for a subset of Viper programs)
fully automatic
front-end soundness
is correct
is correct

Key Contributions



1 Generic simulation methodology relating source language and IVL constructs

verification-aware language with separation logic reasoning

Viper program

size before adaptation: 8.5K lines of Scala code

Boogie program

2 Certifying adaptation of existing Viper-to-Boogie translation

P

P’

(for a subset of Viper programs)

fully automatic

front-end soundness

is correct

is correct

3 Evaluated certifying adaptation on a representative set of Viper benchmarks

Challenge 1: Semantic Gap

Challenge 1: Semantic Gap

	Viper	Boogie
State model	Heap, permissions for heap locations	Only variable store

Challenge 1: Semantic Gap

	Viper	Boogie
State model	Heap, permissions for heap locations	Only variable store
Execution model	1. Expressions can fail to evaluate 2. Complex statements	1. Expressions always evaluate 2. Simple statements

Challenge 1: Semantic Gap

	Viper	Boogie
State model	Heap, permissions for heap locations	Only variable store
Execution model	1. Expressions can fail to evaluate 2. Complex statements	1. Expressions always evaluate 2. Simple statements
Program logic	Flavor of separation logic	No advanced program logic

Challenge 1: Semantic Gap

Viper

```
exhale acc(x.f, q) &&
       y.g > x.f
```

Boogie

```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H’;
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

Challenge 1: Semantic Gap

Viper

separation logic assertion

```
exhale acc(x.f, q) &&
       y.g > x.f
```

Boogie

```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H';
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

Challenge 1: Semantic Gap

Viper

```
exhale acc(x.f, q) &&
       y.g > x.f
```

Viper state:
Heap and permissions for heap locations

Boogie

Viper state modeled explicitly via maps

```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H';
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

permission lookup

heap lookup

Challenge 1: Semantic Gap

Viper

```
exhale acc(x.f, q) &&
       y.g > x.f
```

Boogie

Success conditions explicit

```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H';
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

Challenge 1: Semantic Gap

Viper

```
exhale acc(x.f, q) &&
       y.g > x.f
```

Boogie

Operations modeled via axiomatizations

```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H’;
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

Constrained via Boogie axiom

Challenge 2: Diverse Translations

Viper

```
exhale acc(x.f, q) &&
       y.g > x.f
```

Boogie

Omitted under certain conditions

```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H’;
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

Challenge 3: Non-Locality

Challenge 3: Non-Locality

Viper

```
method m₁(args)
    requires pre
    ensures  post
{
  body
}
```

Boogie

```
procedure p₁(…)
{
   check pre and post well-formed

   body does not fail and respects pre + post

}
```

Challenge 3: Non-Locality

Viper

```
method m1(args)
    requires pre
    ensures  post
{
  body
}

method m2(args)
    requires ...
    ensures  ...
{
  m1(args) ------------------>
}
```

Boogie

```
procedure p1(…)
{
    check pre and post well-formed

    body does not fail and respects pre + post

}

procedure p2(…)
{
  …

    encode m1(args)
  …
}
```

Challenge 3: Non-Locality

Viper

```
method m_1(args)
    requires pre
    ensures  post
{
  body
}
```

```
method m_2(args)
    requires ...
    ensures  ...
{
  m_1(args)
}
```

relies on non-local check

Boogie

```
procedure p_1(…)
{
   check pre and post well-formed
   body does not fail and respects pre + post
}
```

```
procedure p_2(…)
{
  …
   encode m_1(args)
  …
}
```

High-Level Proof Strategy

Viper

```
method m₁
    …
method mₙ
```

Boogie

```
procedure p₁
    …
procedure pₙ
```

High-Level Proof Strategy

Viper

method m_1

…

method m_n

Boogie

procedure p_1

…

procedure p_n

spec(m_i) well-formed **and**

$\left(\begin{array}{l}\text{all specs well-formed } \Rightarrow \\ \text{body}(m_i) \text{ has no failing executions}\end{array}\right)$

$\longleftarrow$

body(p_i) has no failing executions

High-Level Proof Strategy

Viper

```
method m₁
      …
method mₙ
```

Boogie

```
procedure p₁
      …
procedure pₙ
```

spec(m_i) well-formed **and**
(all specs well-formed ⇒
body(m_i) has no failing executions) $\longleftarrow$ body(p_i) has no failing executions

$\Big\Downarrow$ for all i ∈ {1,…n}

no failing method executions $\xleftarrow{\text{front-end soundness}}$ no failing procedure executions

High-Level Proof Strategy

Viper

```
method m₁
    …
method mₙ
```

Boogie

```
procedure p₁
    …
procedure pₙ
```

Generate certificate automatically (challenges appear here)

$\downarrow$

body(p_i) simulates body(m_i)

spec(m_i) well-formed **and** (all specs well-formed ⇒ body(m_i) has no failing executions) $\longleftarrow$ body(p_i) has no failing executions

$\Downarrow$ for all i ∈ {1,...,n}

front-end soundness

no failing method executions $\longleftarrow$ no failing procedure executions

Simulation Decomposition

Viper

```
exhale acc(x.f, q) &&
       y.g > x.f
```

simulates ←

Boogie

```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H’;
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

Simulation Decomposition

Viper

```
exhale acc(x.f, q) &&
       y.g > x.f
```

simulates

Handling semantic gap:
Decompose into simulations of separate semantic concerns

Boogie

```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H’;
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

Simulation Decomposition

Viper

```
exhale acc(x.f, q) &&
       y.g > x.f
```

simulates

Handling semantic gap:
Decompose into simulations of separate semantic concerns

Boogie

```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H’;
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

Simulation Decomposition

Viper

```
exhale acc(x.f, q) &&
       y.g > x.f
```

simulates

acc(x.f, q)

y.g > x.f

exhale finalization

Handling semantic gap:
Decompose into simulations of separate semantic concerns

Boogie

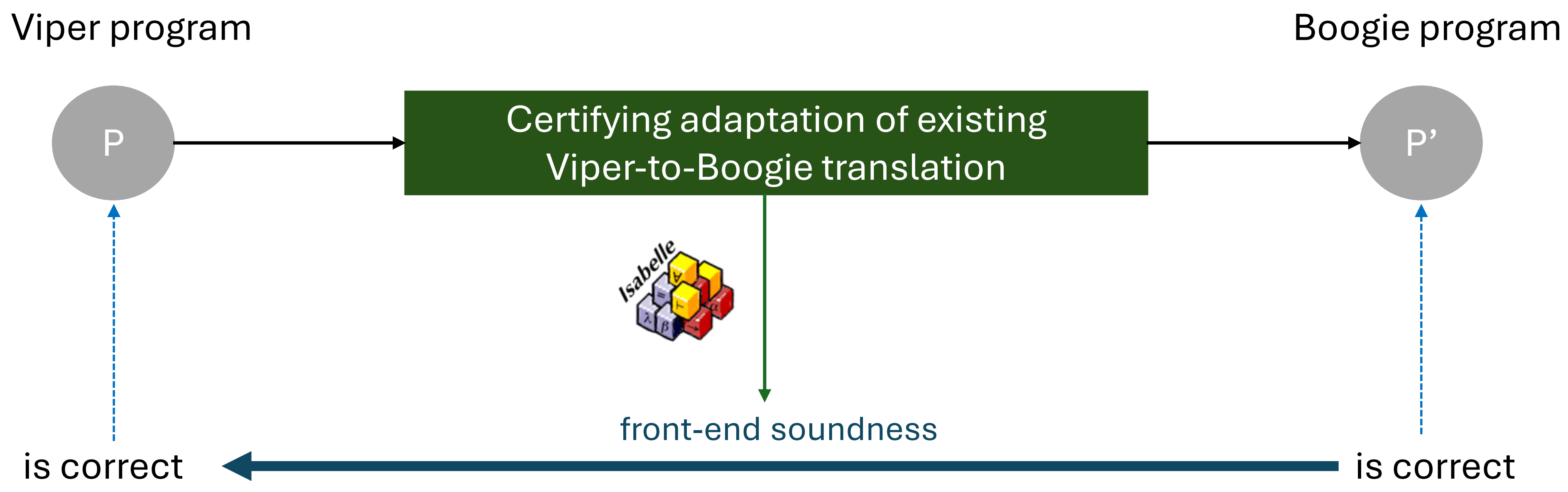
```
WM := M;
tmp := q;
assert tmp >= 0;
if(tmp != 0) {
  assert M[x,f] >= tmp;
}
M[x,f] -= tmp;
assert WM[y,g] > 0;
assert WM[x,f] > 0;
assert H[y,g] > H[x,f];
havoc H';
assume idOnPositive(H,H',M);
H := H';
assume GoodMask(M);
```

High-Level Proof Automation Strategy

1. Decompose simulation into small semantic concerns

2. Tactics for simulation of resulting small semantic concerns

Evaluation



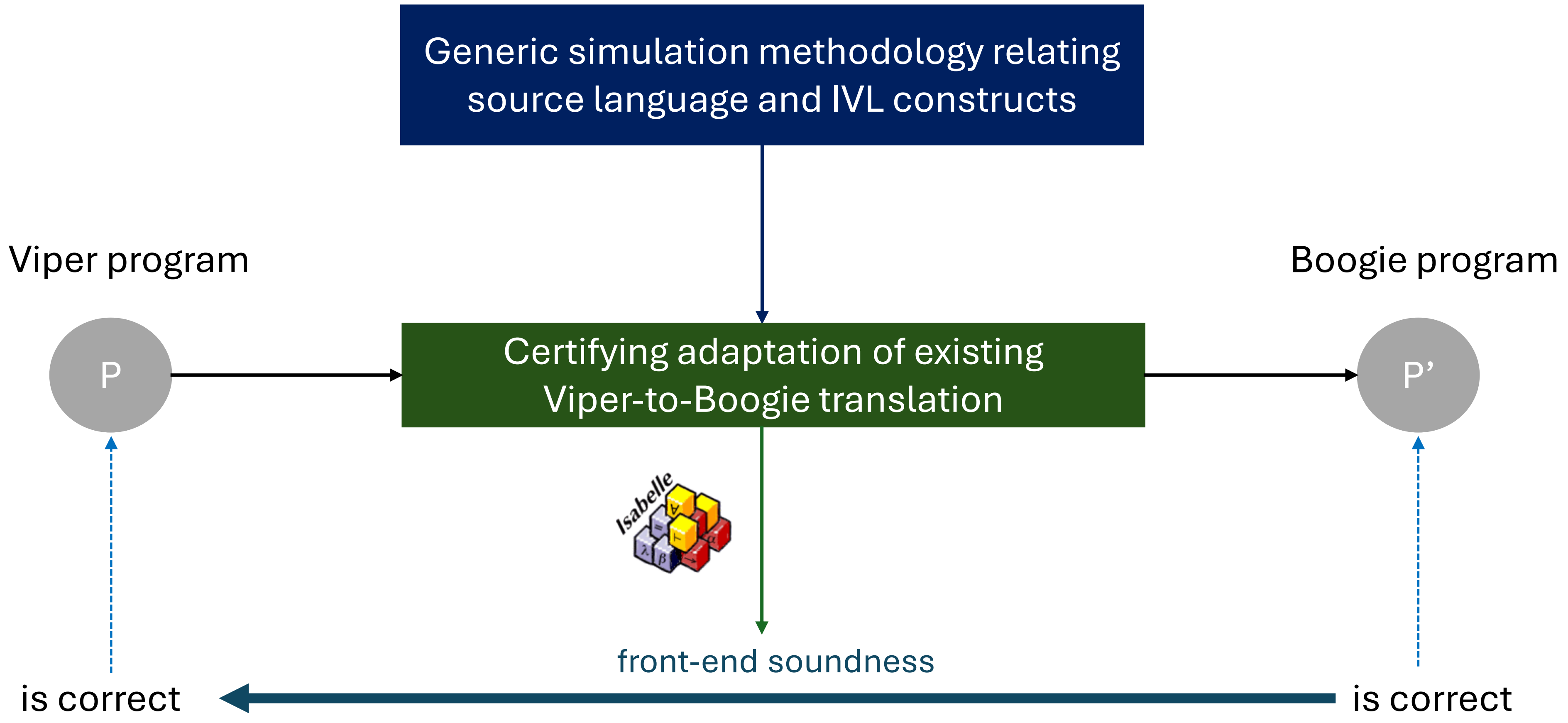
Evaluated certificate generation on 72 representative Viper programs

All generated certificates are successfully checked by Isabelle

Evaluation

File	Viper LoC	Boogie LoC	Isabelle LoC	Time to check certificate [s]
testHistory	205	1711	7035	126.3
defer	211	853	4717	60.6
inv-test	92	514	2596	56.5
darvas	414	2014	9545	242.4
banerjee	91	582	2800	38.4
kusters	112	583	3146	46.2

Conclusion


Generic simulation methodology relating source language and IVL constructs
Viper program
Boogie program
P
Certifying adaptation of existing Viper-to-Boogie translation
P’
Isabelle
front-end soundness
is correct
is correct