Viper

A Verification Infrastructure for Permission-based Reasoning

ETH zürich

Malte Schwerhoff

19th October 2016, Prague

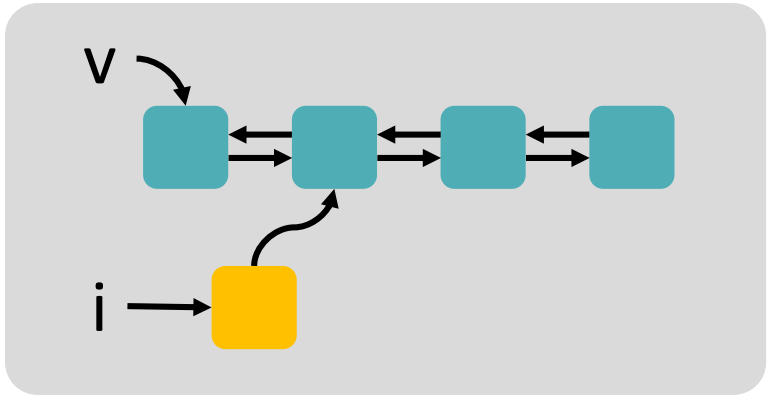
Modular, Deductive & Automated Verifiers

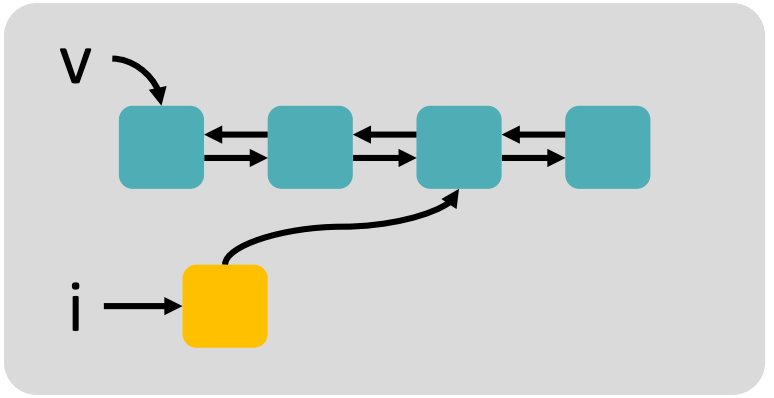
SMT solvers

Verification methodologies

Images: https://www.flickr.com/photos/our-planet/5062099696/in/photostream/
https://pixabay.com/en/gps-navigation-garmin-device-304842/
https://pixabay.com/en/car-sports-car-wedding-car-49278/

Frame Problem





$$\frac{\{P\}\ C\ \{Q\}}{\{P \wedge R\}\ C\ \{Q \wedge R\}}$$

Frame Problem

```
class Account {
  method deposit(n: Int)
    requires 0 < n
    ensures  balance() == old(balance()) + n
  { … }
}
```

```
method client(a: Account, l: List)
{
  var tmp := l.length()
  a.deposit(200)
  assert tmp == l.length()
}
```

Frame Problem

```
class Account {
  transactions: List

  method deposit(n: Int)
    requires 0 < n
    ensures  balance() == old(balance()) + n
  { transaction.append(n) }



}
```

```
method client(a: Account, l: List)
{
  var tmp := l.length()
  a.deposit(200)
  assert tmp == l.length()
}
```

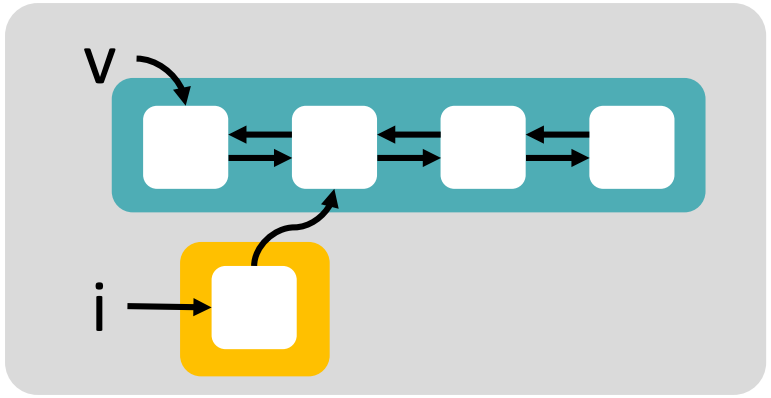
Frame Problem

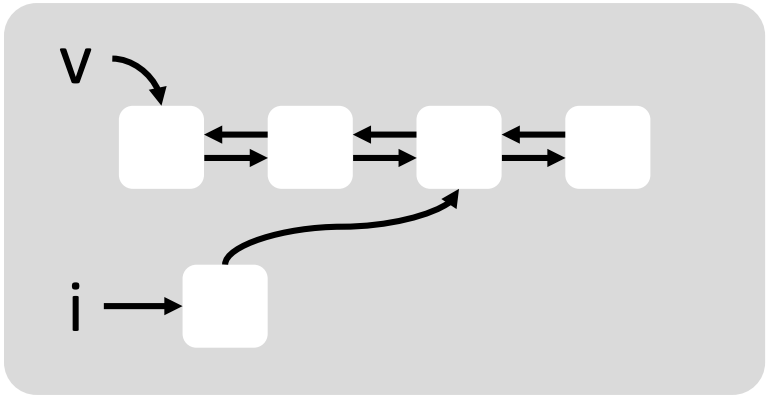
```
class Account {
  transactions: List

  method deposit(n: Int)
    requires 0 < n
    ensures  balance() == old(balance()) + n
  { transaction.append(n) }

  function getTransactions(): List
  { transactions }
}
```

```
method client(a: Account, l: List)
{
  var tmp := l.length()
  a.deposit(200)
  assert tmp == l.length()
}
```

```
var a := new Account()
client(a, a.getTransactions())
```

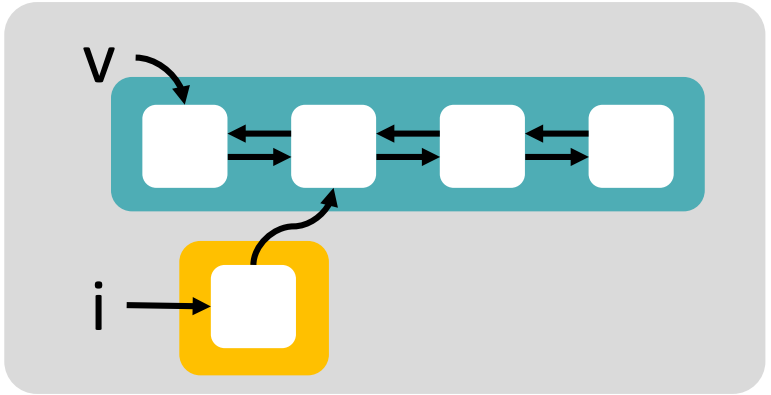
Footprints





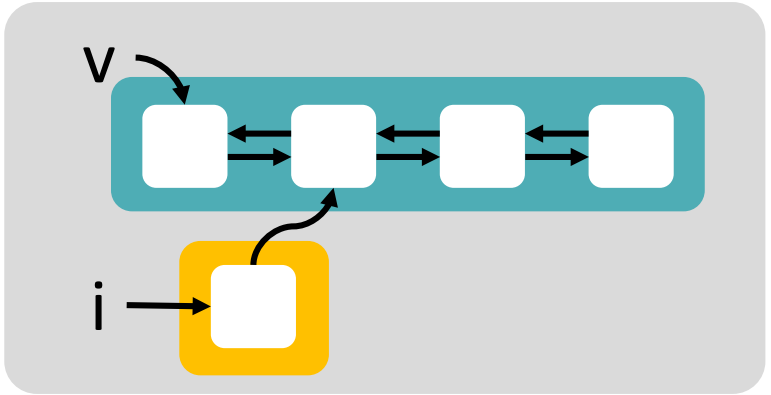
$$\frac{\{\,P\,\}\ C\ \{\,Q\,\} \qquad \text{footprint}(C) \cap \text{footprint}(R) = \emptyset}{\{\,P \wedge R\,\}\ C\ \{\,Q \wedge R\,\}}$$

Explicit Footprints



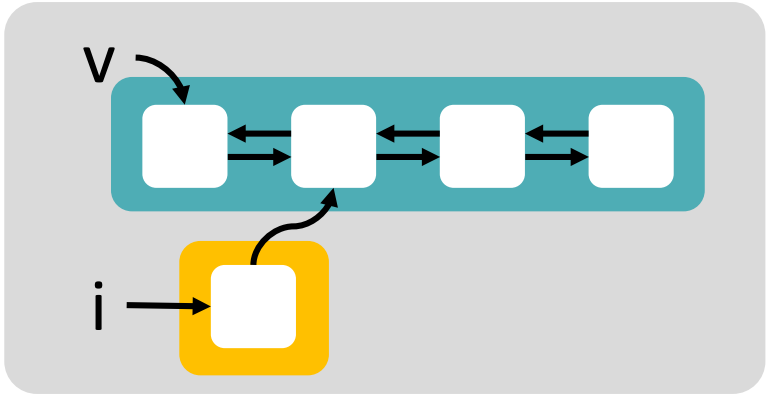
$$\frac{\{\,P\,\}\ C\ \{\,Q\,\} \qquad \text{modifies}(C) \cap \text{reads}(R) = \emptyset}{\{\,P \wedge R\,\}\ C\ \{\,Q \wedge R\,\}}$$

Implicit Footprints



$$\frac{\{\,P\,\}\ C\ \{\,Q\,\}}{\{\,P \wedge R\,\}\ C\ \{\,Q \wedge R\,\}}$$

Implicit Footprints



$$\frac{\{\,P\,\}\ C\ \{\,Q\,\}}{\{\,P * R\,\}\ C\ \{\,Q * R\,\}}$$

Implicit Footprints: Example

```
class Account {
  bal: Int

  method deposit(n: Int)
    requires 0 < n ∧ acc(bal)
    ensures  acc(bal)
  { bal := bal + n }
}
```

```
class List {
  len: Int

  function length(): Int
    requires acc(len)
  { len }
}
```

Implicit Footprints: Example

```
class Account {
  bal: Int

  method deposit(n: Int)
    requires 0 < n ∧ acc(bal)
    ensures  acc(bal)
  { bal := bal + n }
}
```

```
class List {
  len: Int

  function length(): Int
    requires acc(len)
  { len }
}
```


```
method client(a: Account, l: List)
  requires acc(a.bal) * acc(l.len)
{
  var tmp := l.length()
  a.deposit(200)
  assert tmp == l.length()
}
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}$$

Implicit Footprints: Example

```
class Account {
  bal: Int

  method deposit(n: Int)
    requires 0 < n ∧ acc(bal)
    ensures  acc(bal)
  { bal := bal + n }
}
```

```
class List {
  len: Int

  function length(): Int
    requires acc(len)
  { len }
}
```


```
method client(a: Account, l: List)
  requires acc(a.bal) * acc(l.len)
{
  var tmp := l.length()
  a.deposit(200)
  assert tmp == l.length()
}
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}$$

Implicit Footprints: Example

```
class Account {
  bal: Int

  method deposit(n: Int)
    requires 0 < n ∧ acc(bal)
    ensures  acc(bal)
  { bal := bal + n }
}
```

a.bal
B

l.len
L

```
method client(a: Account, l: List)
  requires acc(a.bal) * acc(l.len)
{
  var tmp := l.length()
  a.deposit(200)
  assert tmp == l.length()
}
```

```
class List {
  len: Int

  function length(): Int
    requires acc(len)
  { len }
}
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}$$

Implicit Footprints: Example

```
class Account {
  bal: Int

  method deposit(n: Int)
    requires 0 < n ∧ acc(bal)
    ensures  acc(bal)
  { bal := bal + n }
}
```

a.bal
?

l.len
L

```
method client(a: Account, l: List)
  requires acc(a.bal) * acc(l.len)
{
  var tmp := l.length()
  a.deposit(200)
  assert tmp == l.length()
}
```

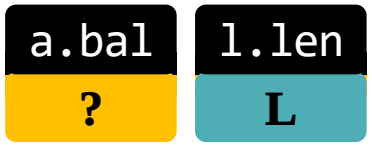
```
class List {
  len: Int

  function length(): Int
    requires acc(len)
  { len }
}
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}$$

Implicit Footprints: Example

```
class Account {
  bal: Int

  method deposit(n: Int)
    requires 0 < n ∧ acc(bal)
    ensures  acc(bal)
  { bal := bal + n }
}
```

```
class List {
  len: Int

  function length(): Int
    requires acc(len)
  { len }
}
```



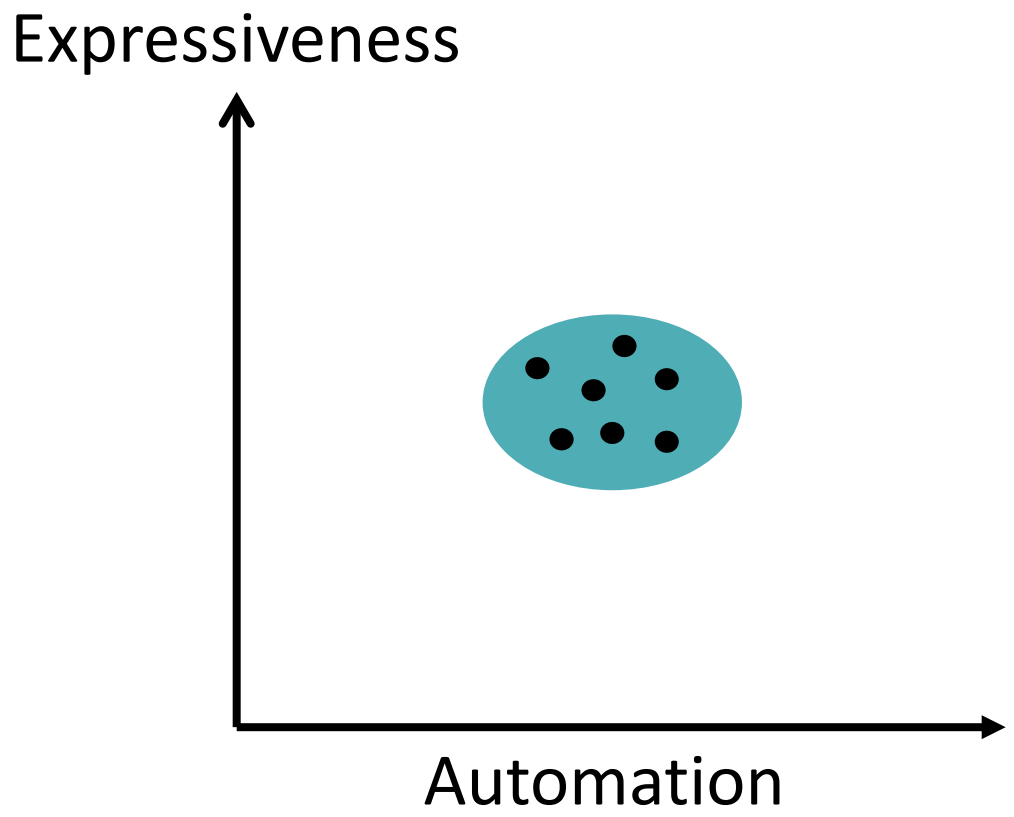

```
method client(a: Account, l: List)
  requires acc(a.bal) * acc(l.len)
{
  var tmp := l.length()
  a.deposit(200)
  assert tmp == l.length()
}
```

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}$$

Tool Support

No permissions

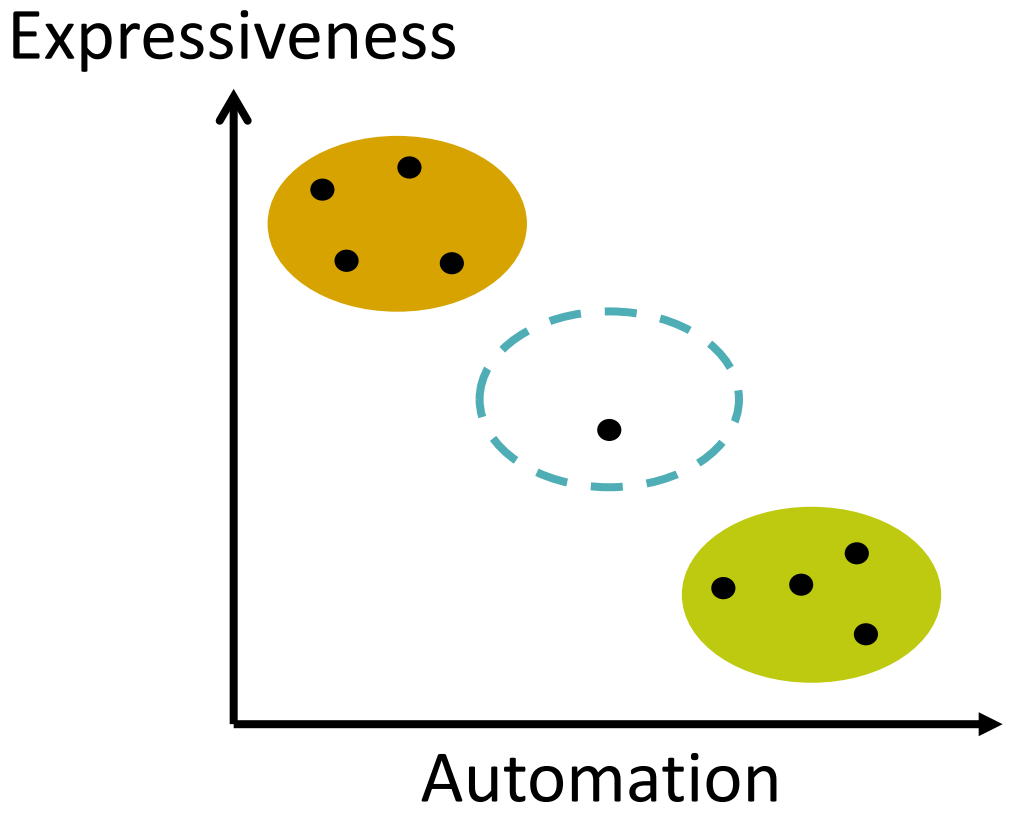
- AutoProof
- Dafny
- Frama-C
- KeY2
- Spec#
- VCC
- VERL



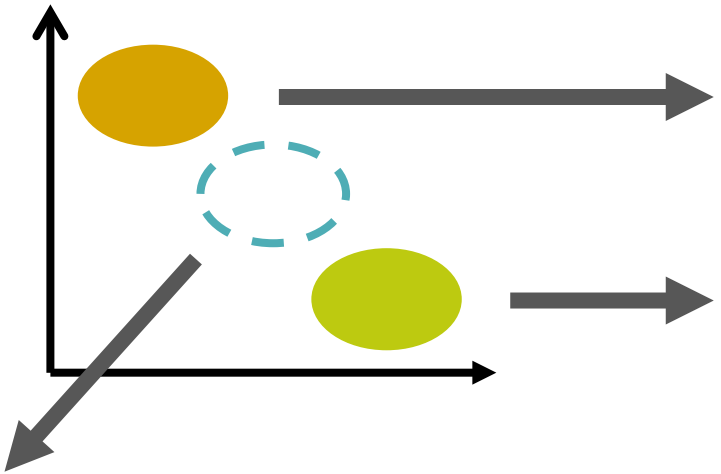
Tool Support

Permissions

- Bedrock
- Chalice
- FCSL/Coq
- Grasshopper
- Infer
- jStar
- SmallFoot
- VeriFast
- Ynot



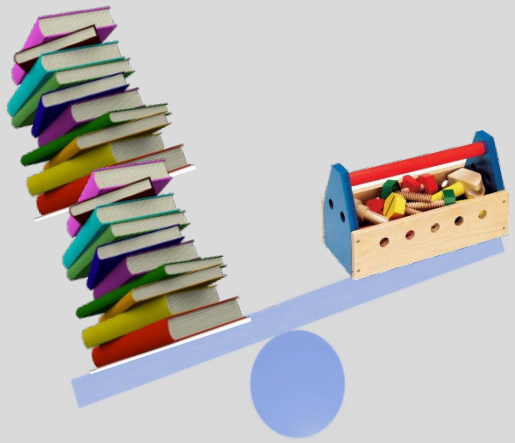
Consequences of Insufficient Tool Support



significant manual effort per application

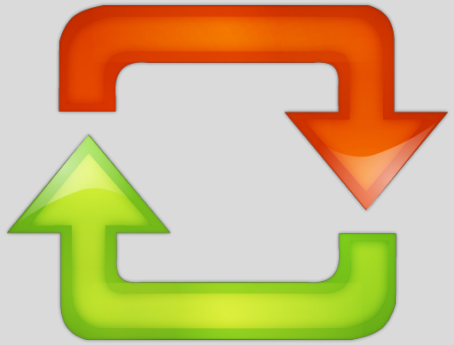
simple properties only

popular, but hardly any permission-based tools



Verification efforts do not benefit fully from advances in theory

Theory does not receive feedback from applications

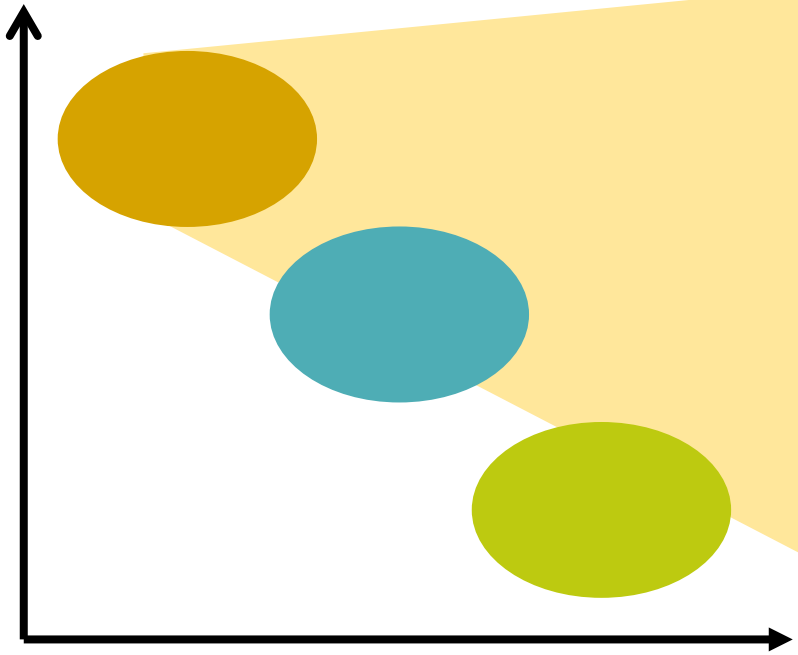


Images: https://www.flickr.com/photos/61423903@N06/7357608430
https://www.flickr.com/photos/rustedhammer/2298322271
http://www.iconarchive.com/show/misc-icons-by-tooschee/Sync-icon.html

Facilitate the

1. development of tools
2. prototyping of encodings

for permission-based verification

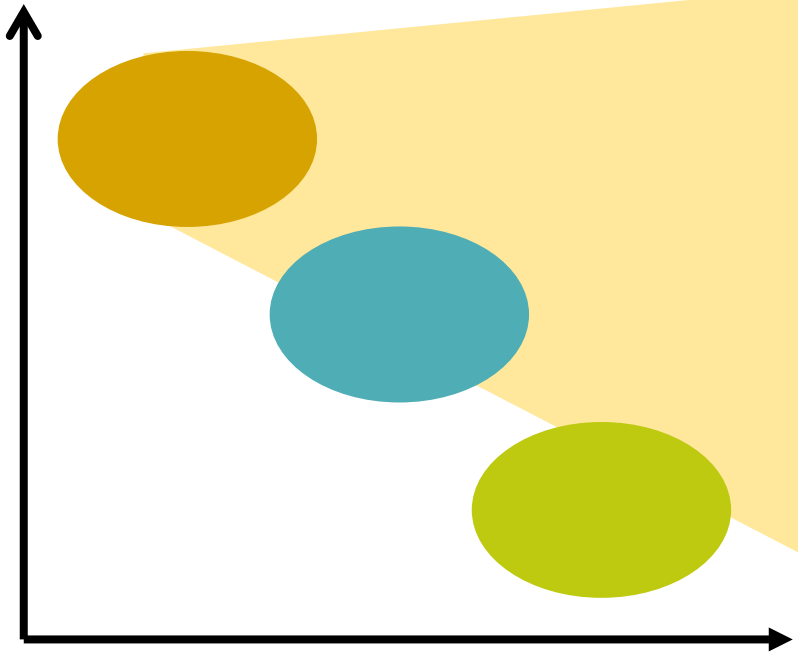


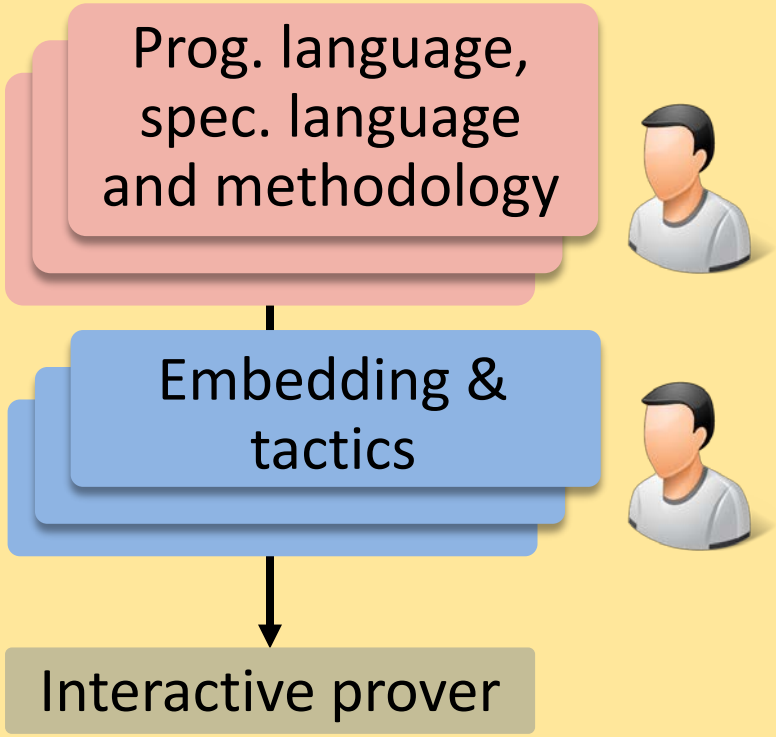
Prog. language, spec. language and methodology

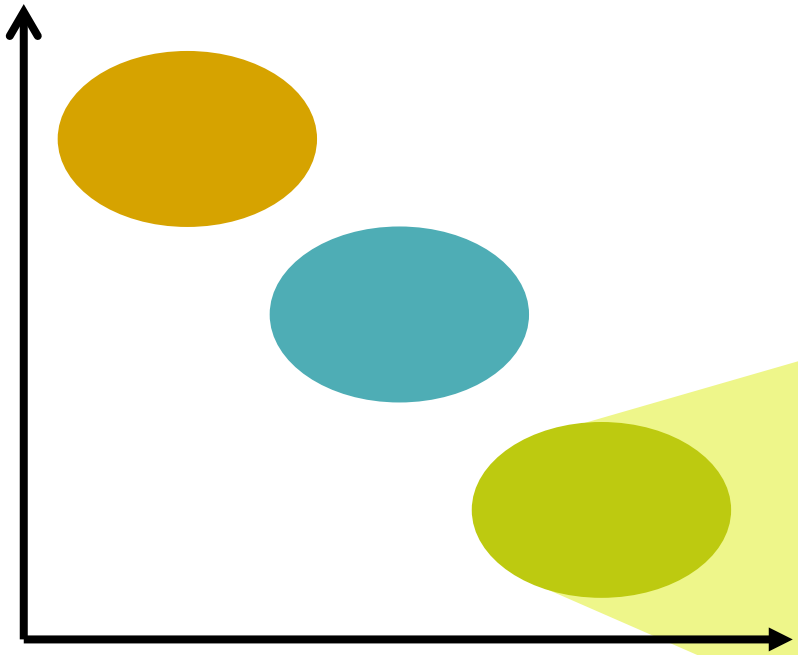
Embedding & tactics

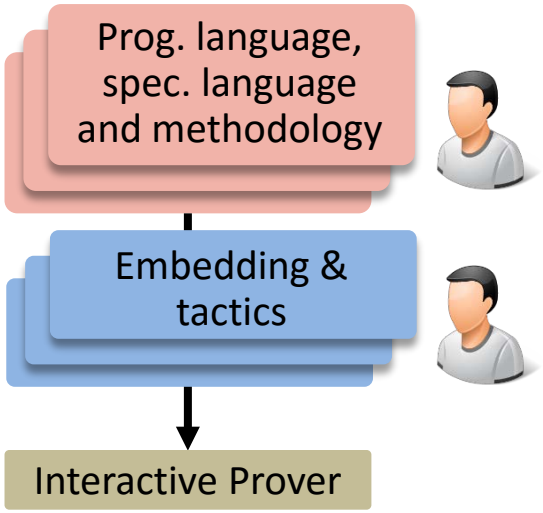
Interactive prover

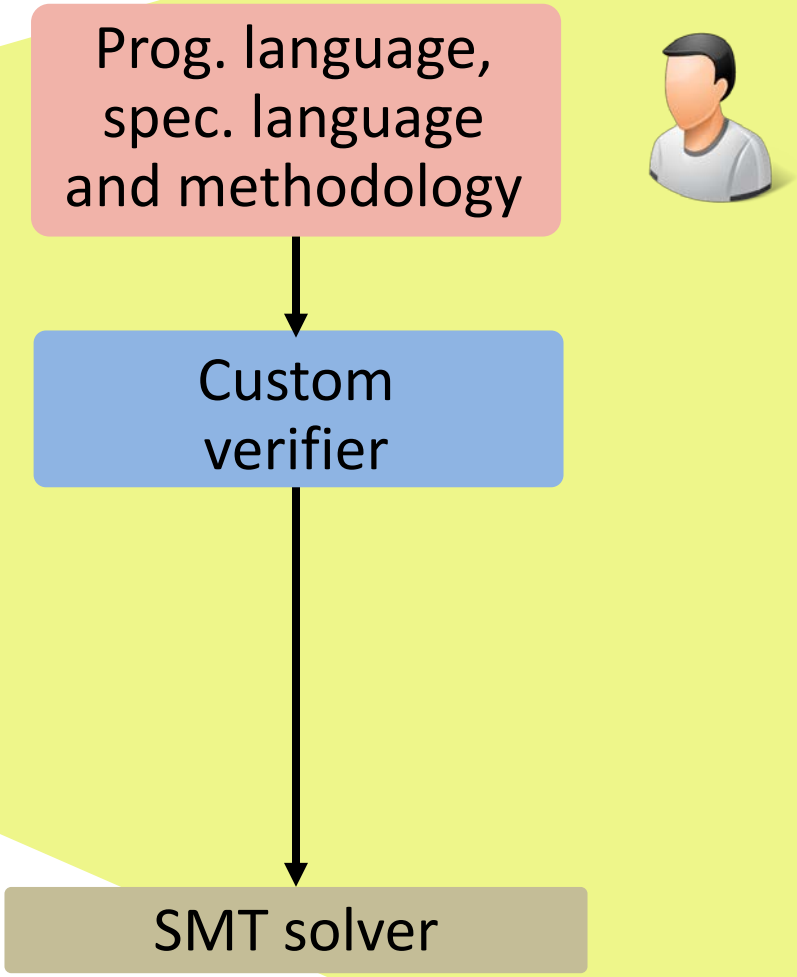
Icons: http://www.iconarchive.com/show/vista-people-icons-by-icons-land.html
http://www.icons-land.com/

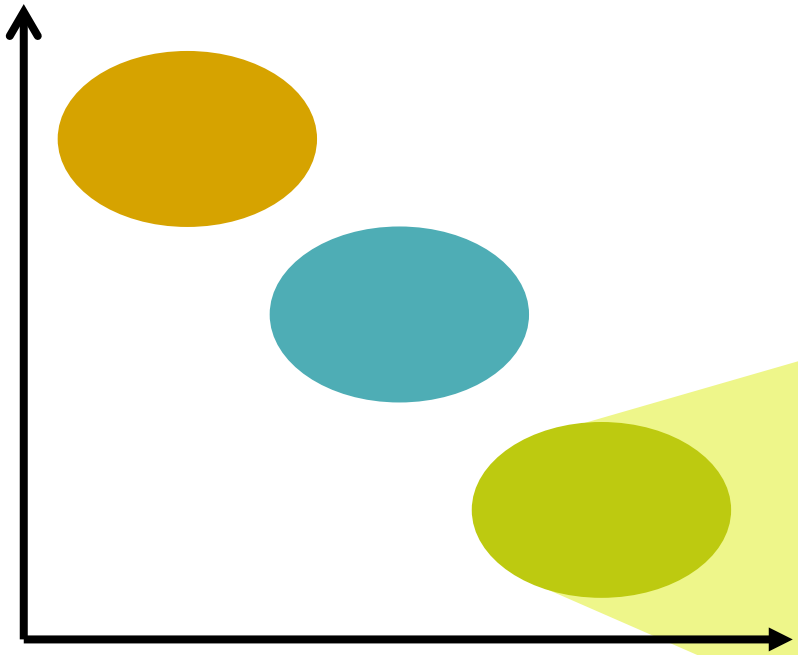


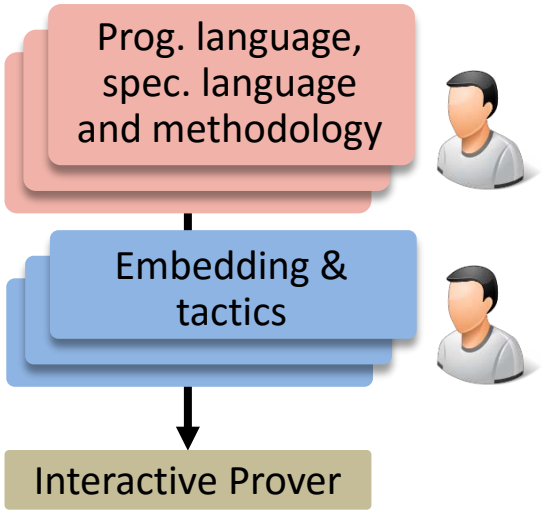

Prog. language, spec. language and methodology
Embedding & tactics
Interactive prover

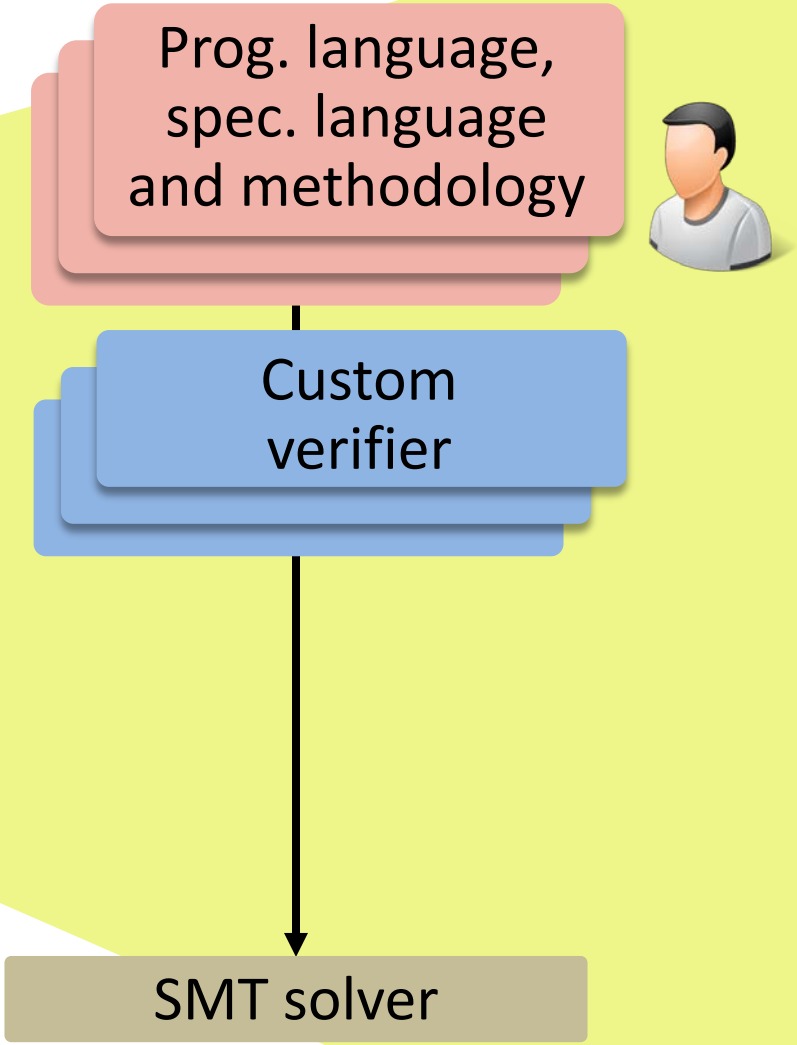


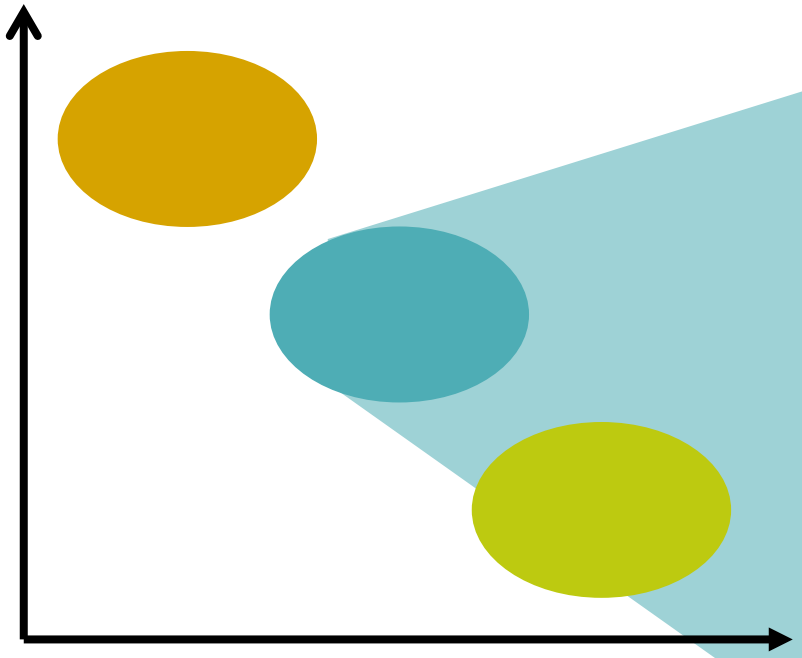

Prog. language, spec. language and methodology
Embedding & tactics
Interactive Prover

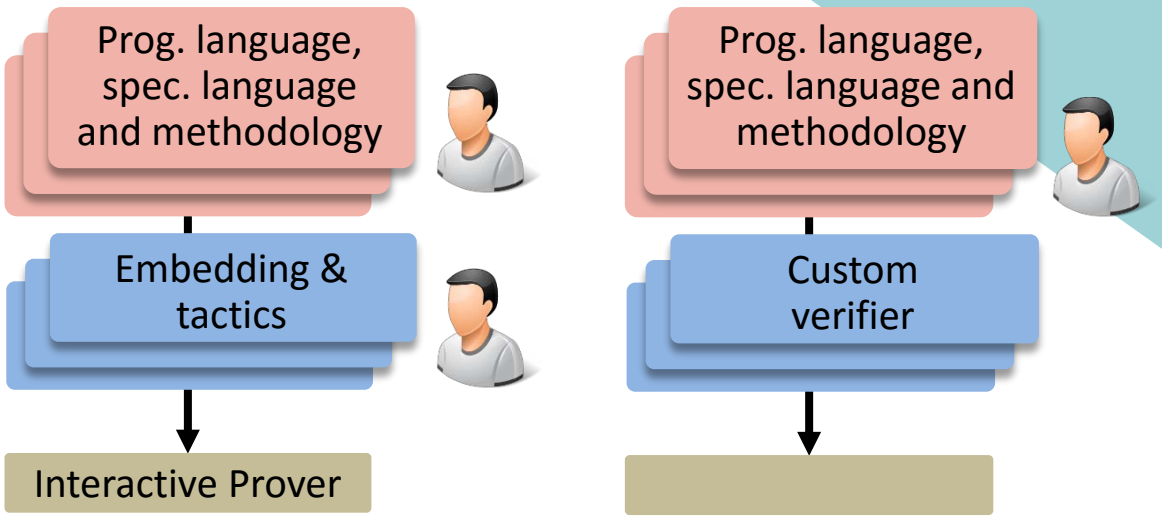

Prog. language, spec. language and methodology
Custom verifier
SMT solver

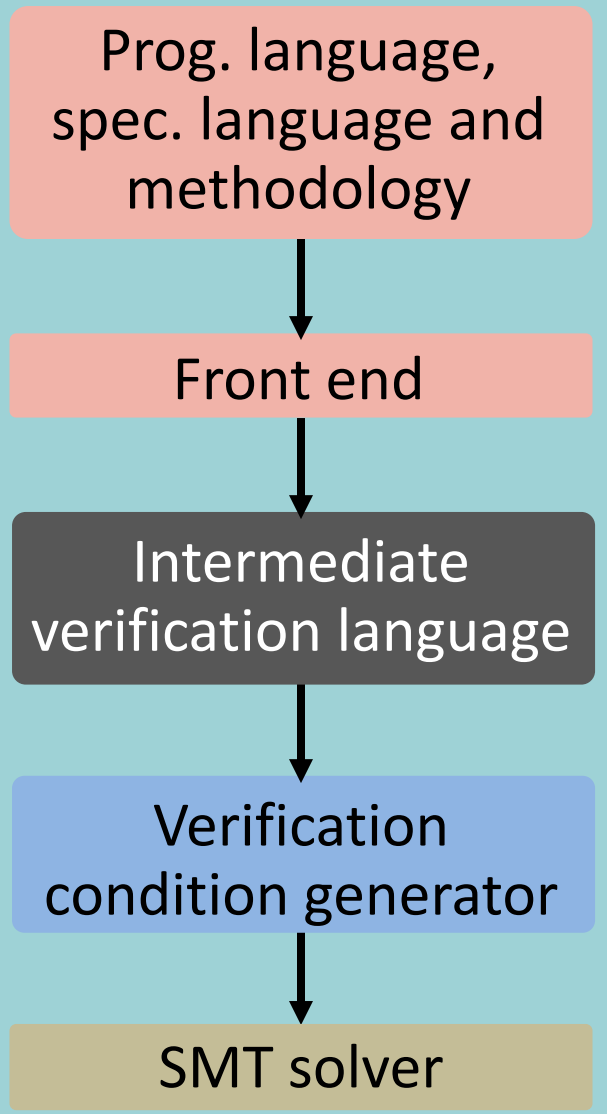


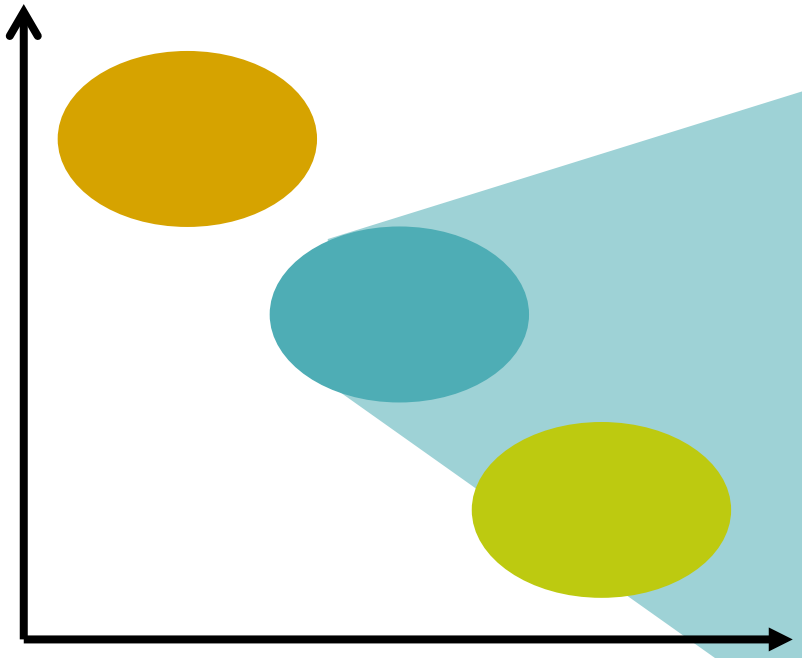

Prog. language, spec. language and methodology
Embedding & tactics
Interactive Prover

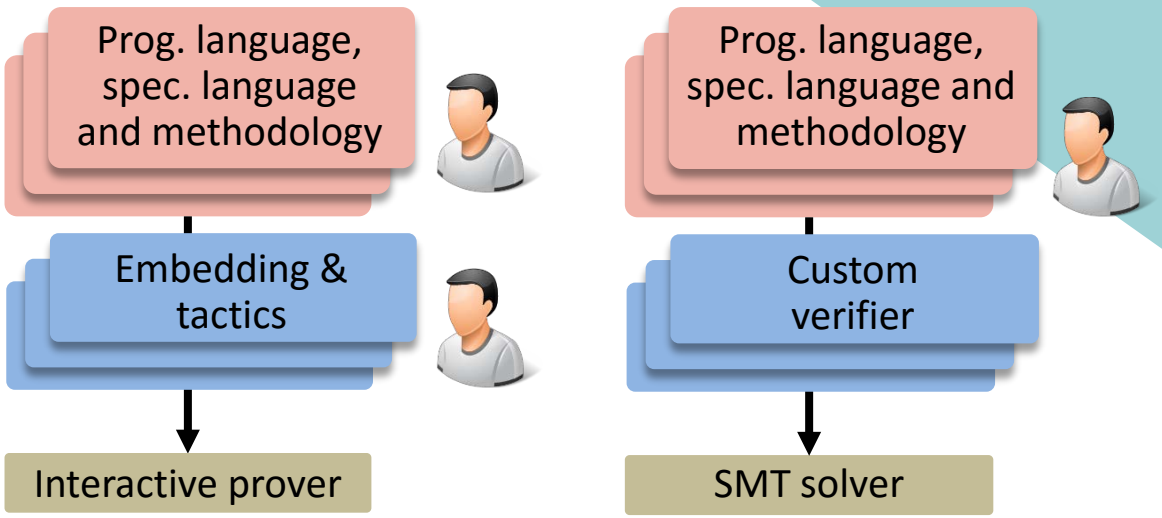

Prog. language, spec. language and methodology
Custom verifier
SMT solver

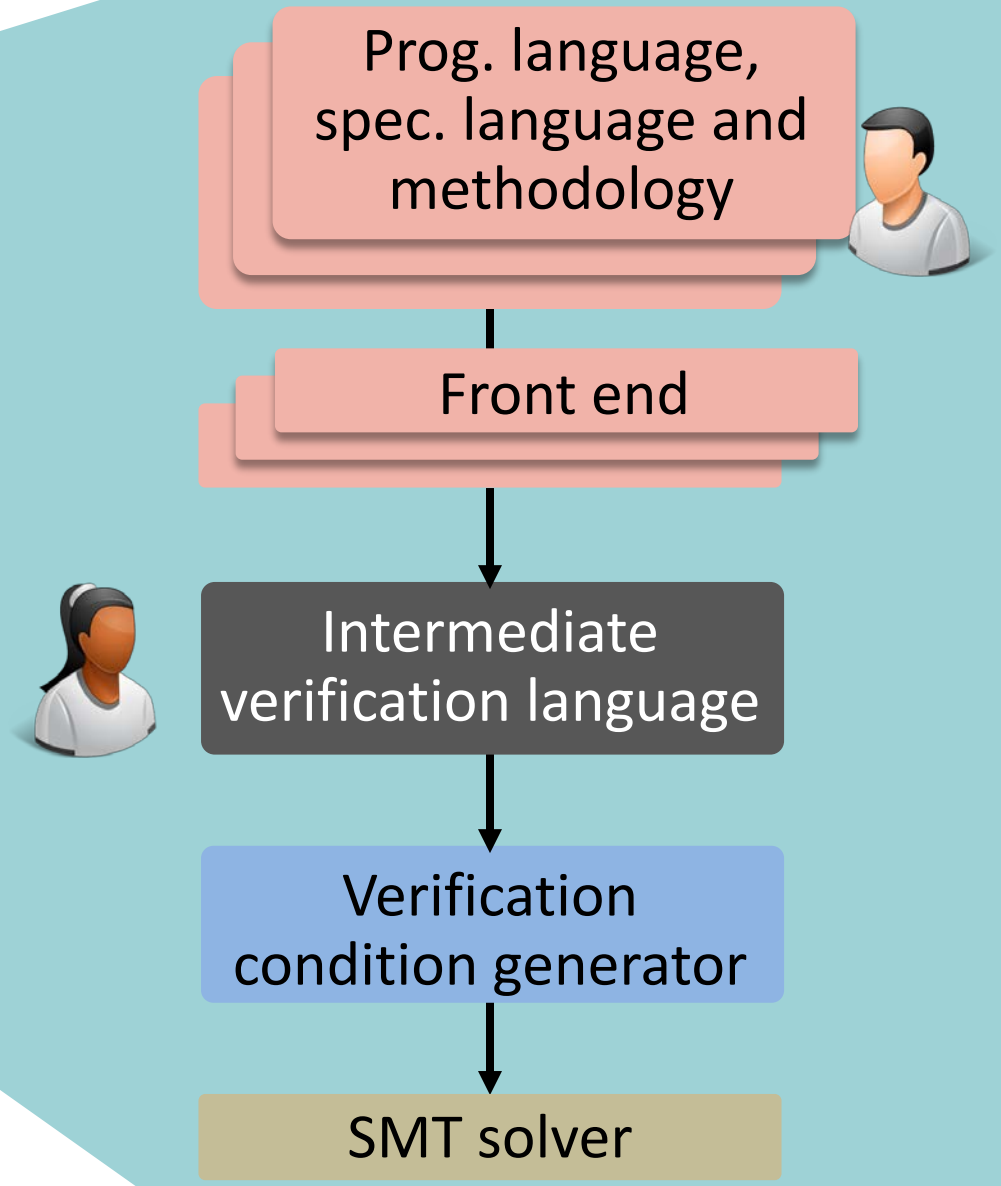



Prog. language, spec. language and methodology
Embedding & tactics
Interactive Prover
Prog. language, spec. language and methodology
Custom verifier

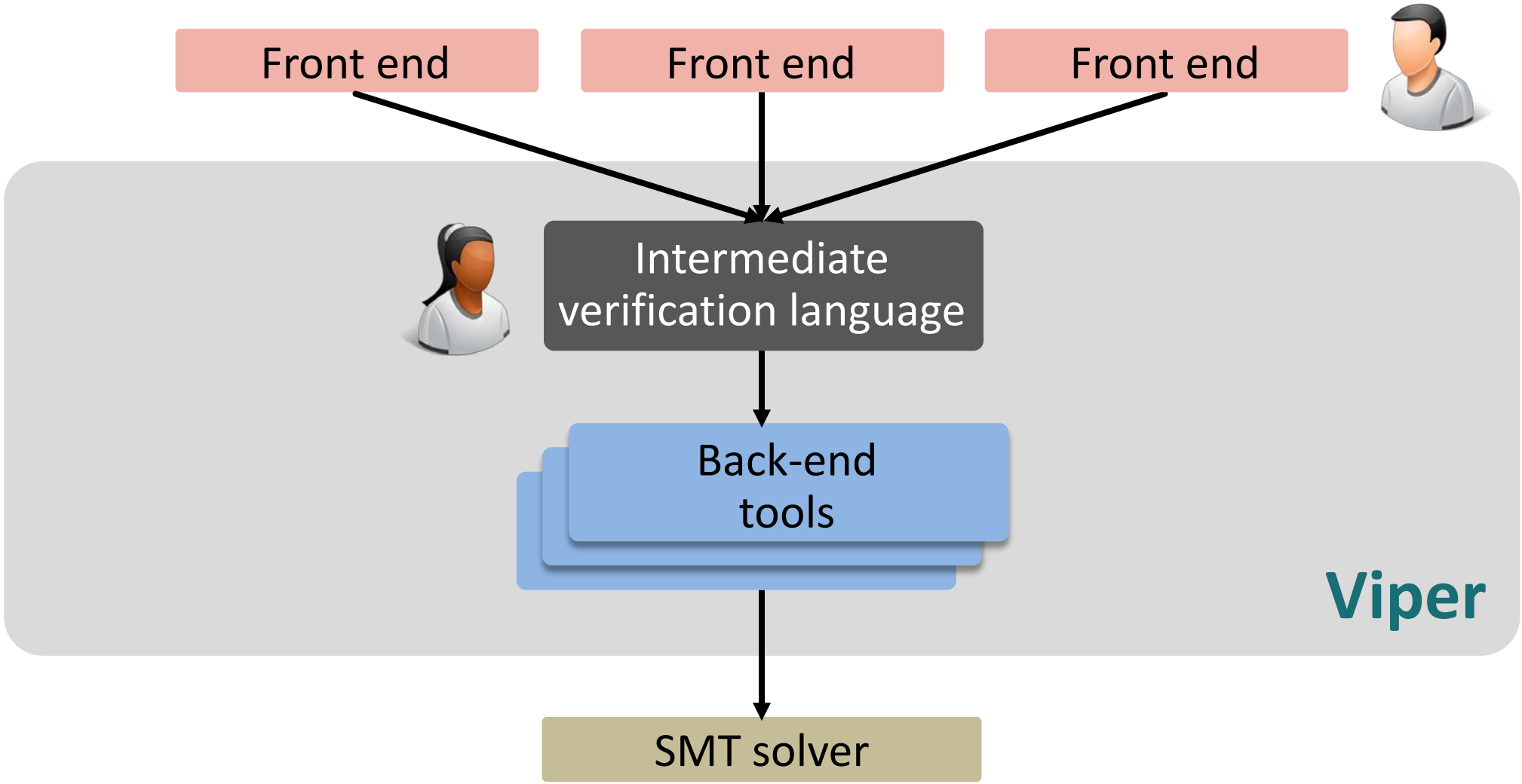

Prog. language, spec. language and methodology
Front end
Intermediate verification language
Verification condition generator
SMT solver



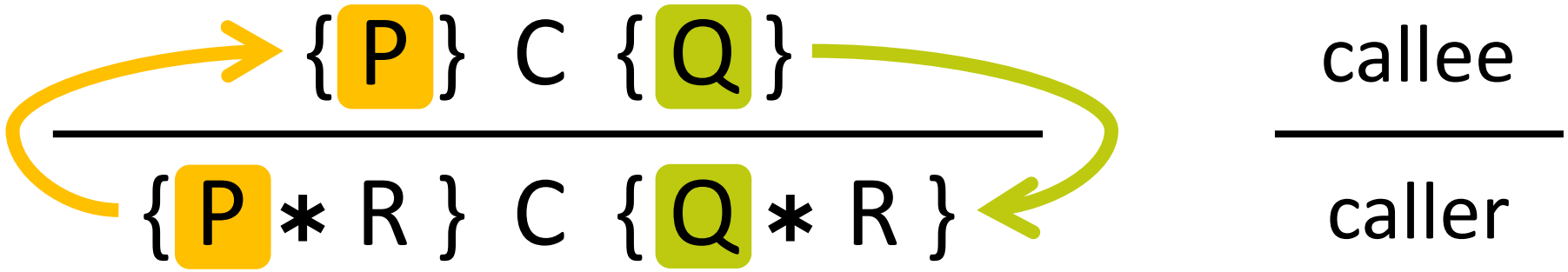

Prog. language, spec. language and methodology
Embedding & tactics
Interactive prover
Prog. language, spec. language and methodology
Custom verifier
SMT solver


Prog. language, spec. language and methodology
Front end
Intermediate verification language
Verification condition generator
SMT solver

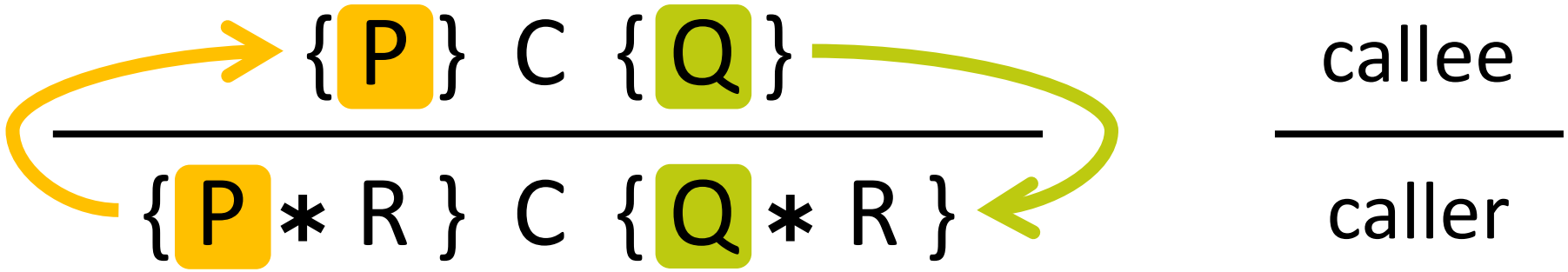
Viper


Front end
Front end
Front end
Intermediate verification language
Back-end tools
Viper
SMT solver

Permission Transfer



Inhale and Exhale



`exhale P`

- assert value constraints
- check and remove permissions
- havoc newly-inaccessible locations

`inhale Q`

- obtain permissions
- assume value constraints

Example: Permission Transfer

```
method deposit(n: Int)
  requires 0 < n ∧ acc(bal)
  ensures  acc(bal)
```

```
method client(a: Account, l: List)
  requires acc(a.bal) * acc(l.len)
{
  var tmp := l.length()

  a.deposit(200)

  assert tmp == l.length()
}
```

```
method client(a: Account, l: List)
{
  inhale acc(a.bal) * acc(l.len)
  var tmp := l.length()
  exhale 0 < 200 ∧ acc(a.bal)
    // a.deposit(200)
  inhale acc(a.bal)
  assert tmp == l.length()
}
```

Modelling Thread Interleavings

```
class Counter {
  @GuardedBy("this");
  int val;









}
```

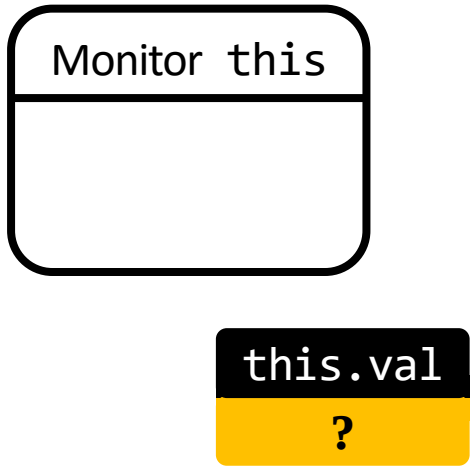
Modelling Thread Interleavings

```
class Counter {
  @GuardedBy("this");
  int val;

void fails() {
  synchronized(this) {
    val = 5;
  }

  synchronized(this) {
    assert val == 5;
  }
}
```


```
field val: Int

method fails(this: Ref)
{
  inhale acc(this.val)
  this.val := 5
  exhale acc(this.val)

  inhale acc(this.val)
  assert this.val == 5
  exhale acc(this.val)
}
```

Modelling Thread Interleavings

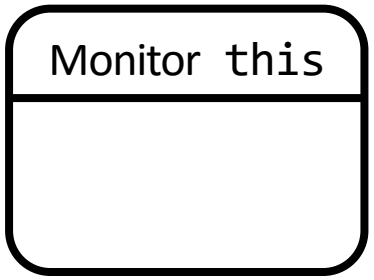
```
class Counter {
  @GuardedBy("this");
  int val;

void fails() {
  synchronized(this) {
    val = 5;
  }

  synchronized(this) {
    assert val == 5;
  }
}
```




```
field val: Int

method fails(this: Ref)
{
  inhale acc(this.val)
  this.val := 5
  exhale acc(this.val)

  inhale acc(this.val)
  assert this.val == 5
  exhale acc(this.val)
}
```

Modelling Thread Interleavings

```
class Counter {
  @GuardedBy("this");
  int val;

void fails() {
  synchronized(this) {
    val = 5;
  }

  synchronized(this) {
    assert val == 5;
  }
}
```




```
field val: Int

method fails(this: Ref)
{
  inhale acc(this.val)
  this.val := 5
  exhale acc(this.val)

  inhale acc(this.val)
  assert this.val == 5
  exhale acc(this.val)
}
```

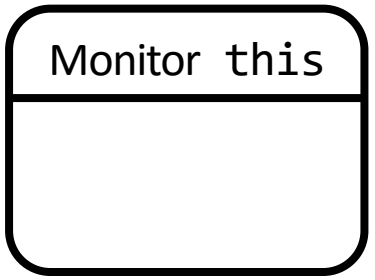
Modelling Thread Interleavings

```
class Counter {
  @GuardedBy("this");
  int val;

void fails() {
  synchronized(this) {
    val = 5;
  } // end

  synchronized(this) {
    assert val == 5;
  }
}
```


```
field val: Int

method fails(this: Ref)
{
  inhale acc(this.val)
  this.val := 5
  exhale acc(this.val)

  inhale acc(this.val)
  assert this.val == 5
  exhale acc(this.val)
}
```

Modelling Thread Interleavings

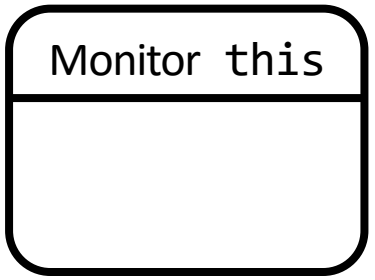
```
class Counter {
  @GuardedBy("this");
  int val;

void fails() {
  synchronized(this) {
    val = 5;
  }

  synchronized(this) {
    assert val == 5;
  }
}
```




```
field val: Int

method fails(this: Ref)
{
  inhale acc(this.val)
  this.val := 5
  exhale acc(this.val)

  inhale acc(this.val)
  assert this.val == 5
  exhale acc(this.val)
}
```

Modelling Thread Interleavings

```
class Counter {
  @GuardedBy("this");
  int val;

void fails() {
  synchronized(this) {
    val = 5;
  }

  synchronized(this) {
    assert val == 5;
  }
}
```

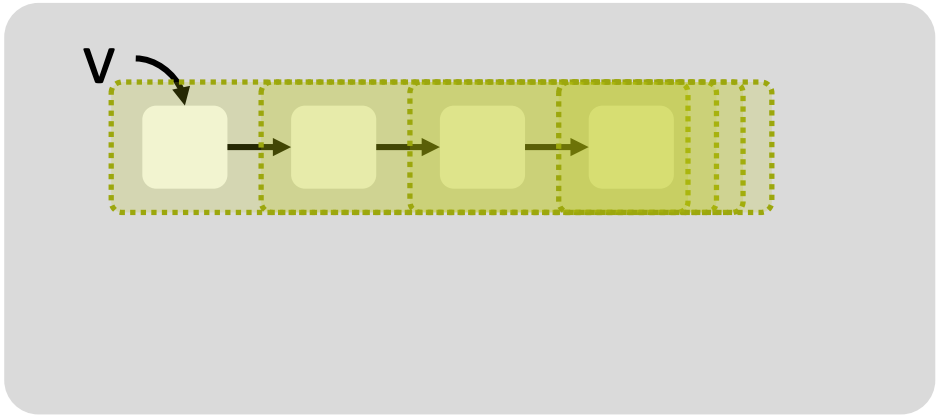



```
field val: Int

method fails(this: Ref)
{
  inhale acc(this.val)
  this.val := 5
  exhale acc(this.val)

  inhale acc(this.val)
  assert this.val == 5
  exhale acc(this.val)
}
```

Demo Example (Pseudo-Java)

```
class Counter {
  @GuardedBy(“this”);
  @MonitorInvariant(“old(val) <= val”);
  int val;

  synchronized void inc() {
    val++;
  }
}
```

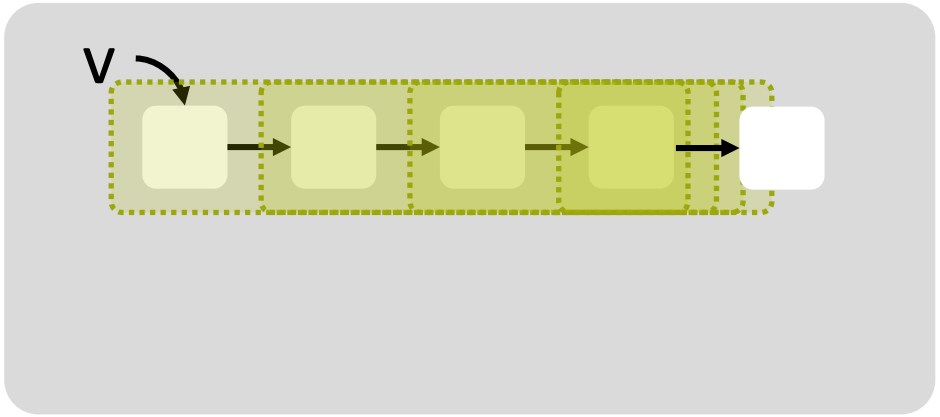
Recursive Predicates




```
predicate list(this: Ref) {
  this != null ==>
    acc(this.data) &&
    acc(this.next) &&
    list(this.next)
}
```

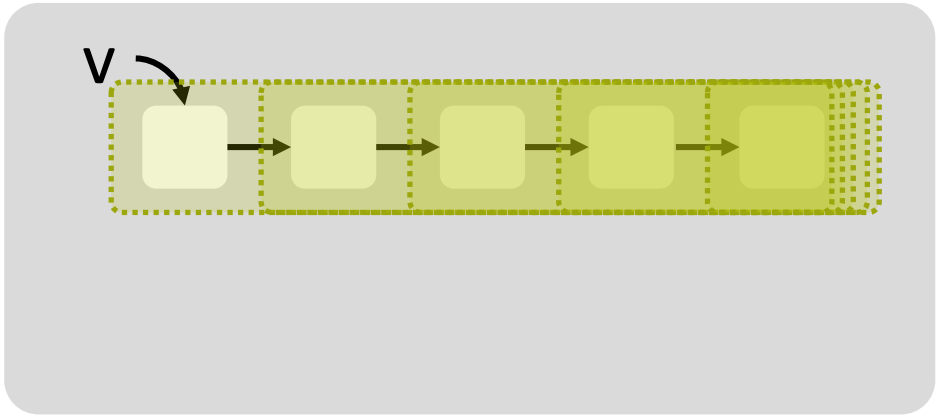
```
unfold list(this)
// access this.data
// and this.next
fold list(this)
```

Recursive Predicates: Limitations



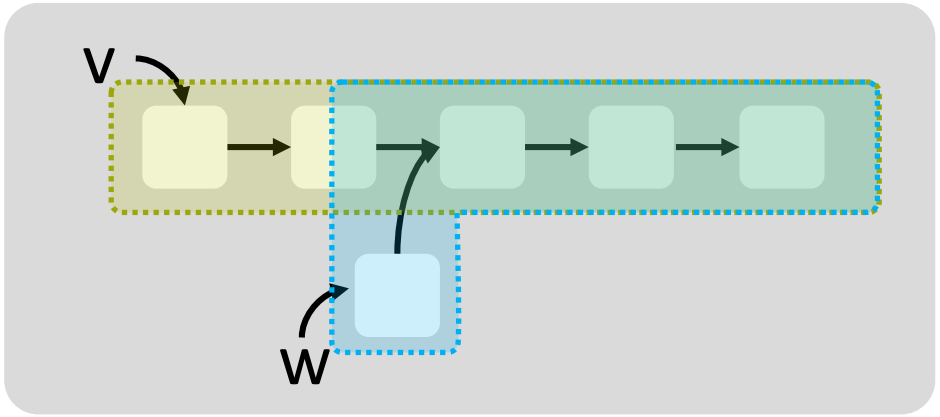
1. Extending footprints

Recursive Predicates: Limitations



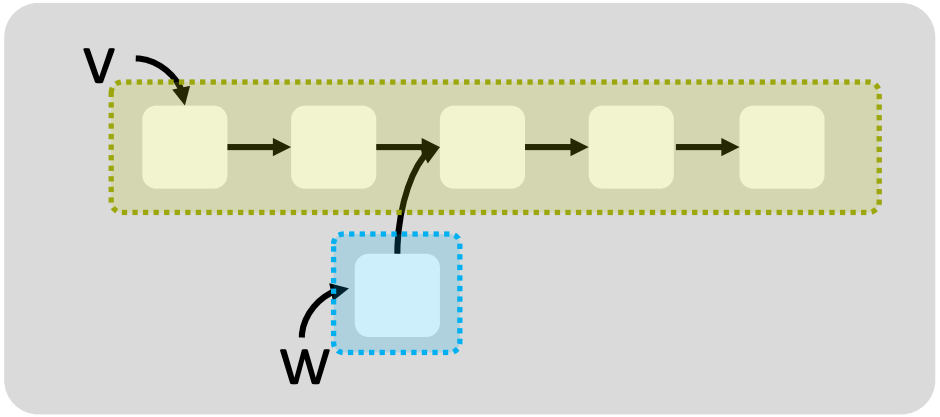
1. Extending footprints

Recursive Predicates: Limitations



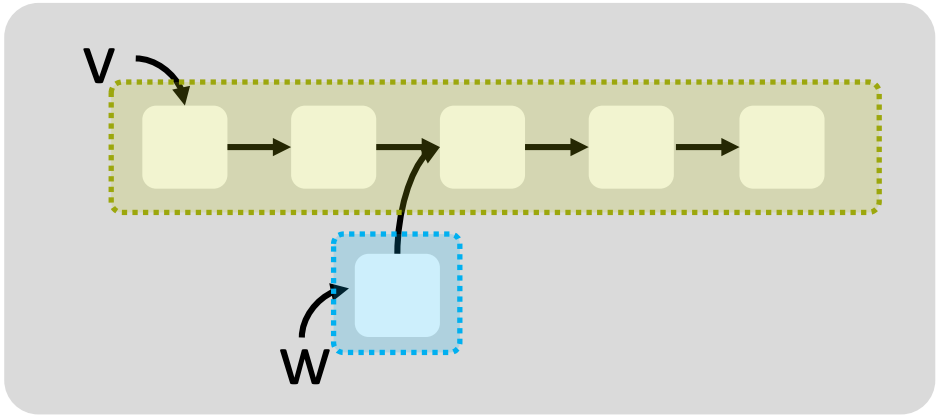
1. Extending footprints
2. Sharing

Recursive Predicates: Limitations



1. Extending footprints
2. Sharing

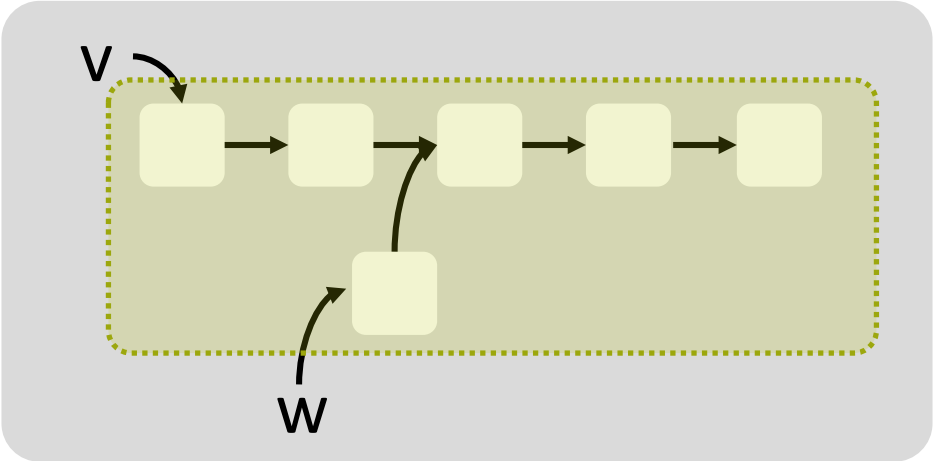
Recursive Predicates: Limitations



1. Extending footprints
2. Sharing
3. Traversal order

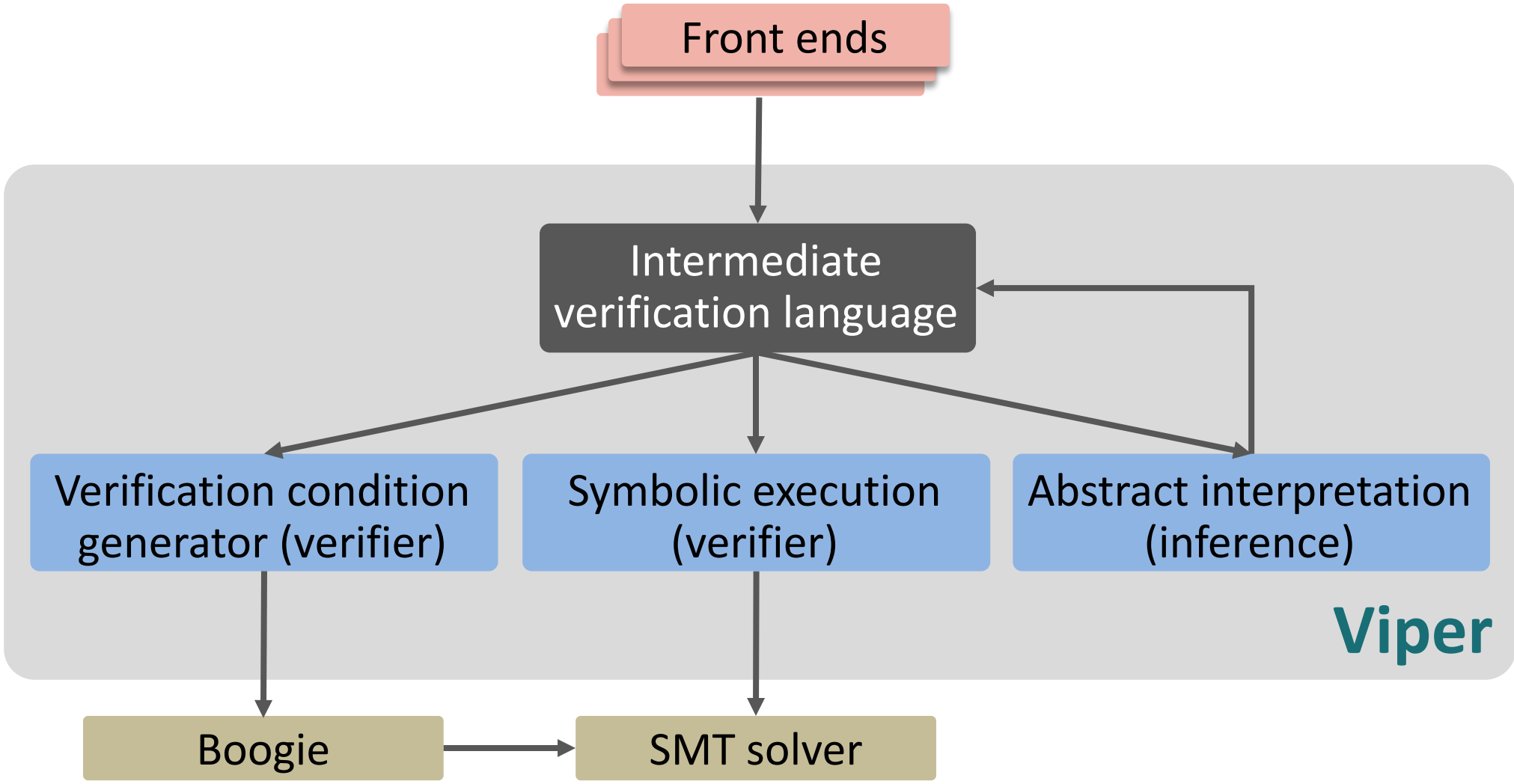
Explicit Footprints with Permissions

```
predicate list(nodes: Set[Ref]) {
  forall n ∈ nodes ::
    acc(n.data) &&
    acc(n.next) &&
    (n.next != null ==>
        n.next ∈ nodes)
}
```

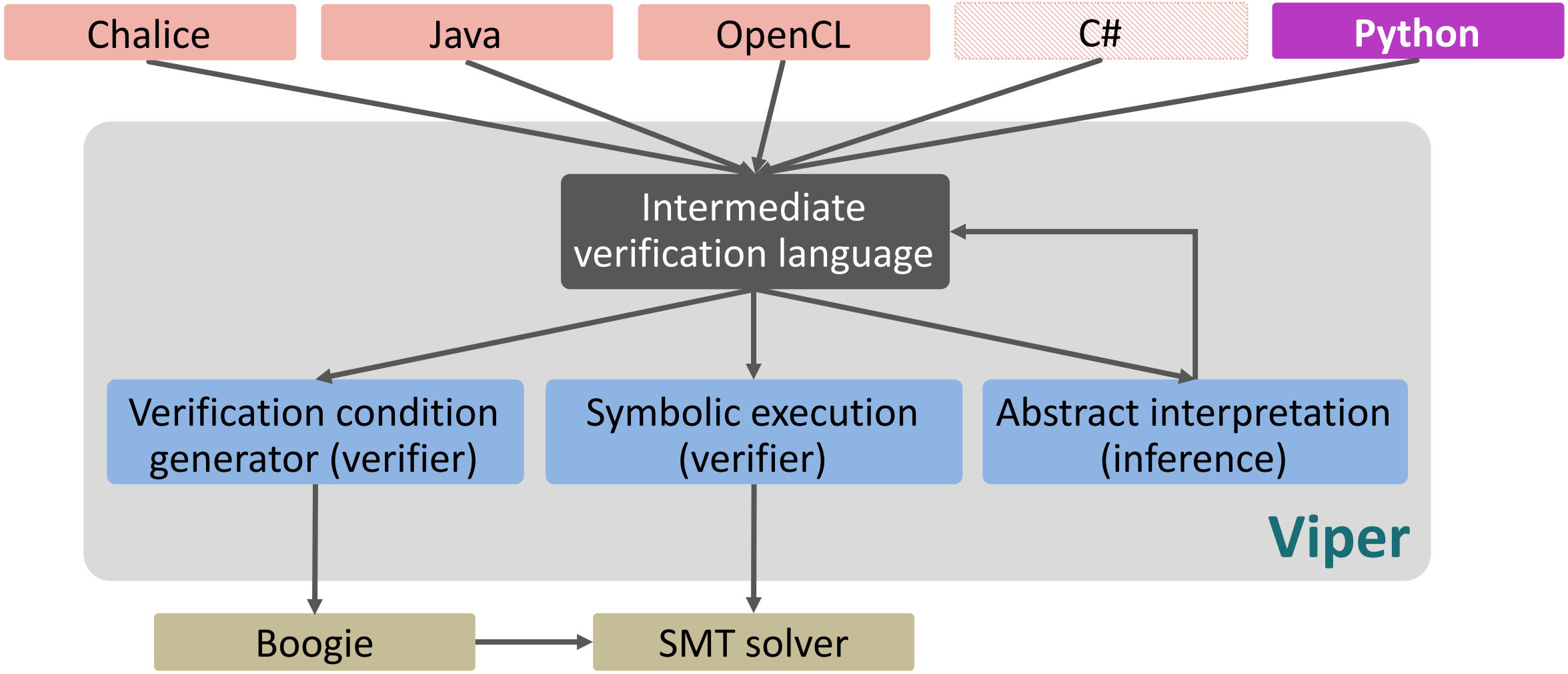


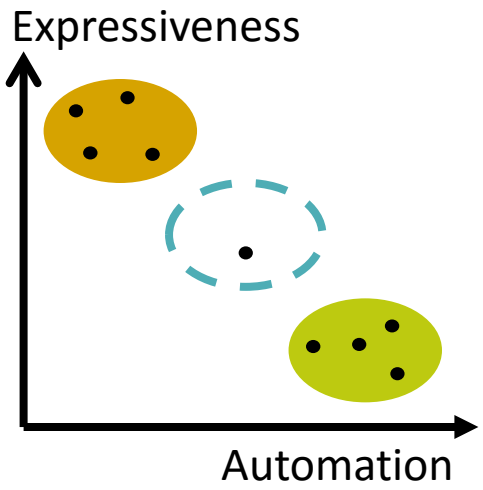

```
list(nodes) &&
v ∈ nodes && w ∈ nodes
```

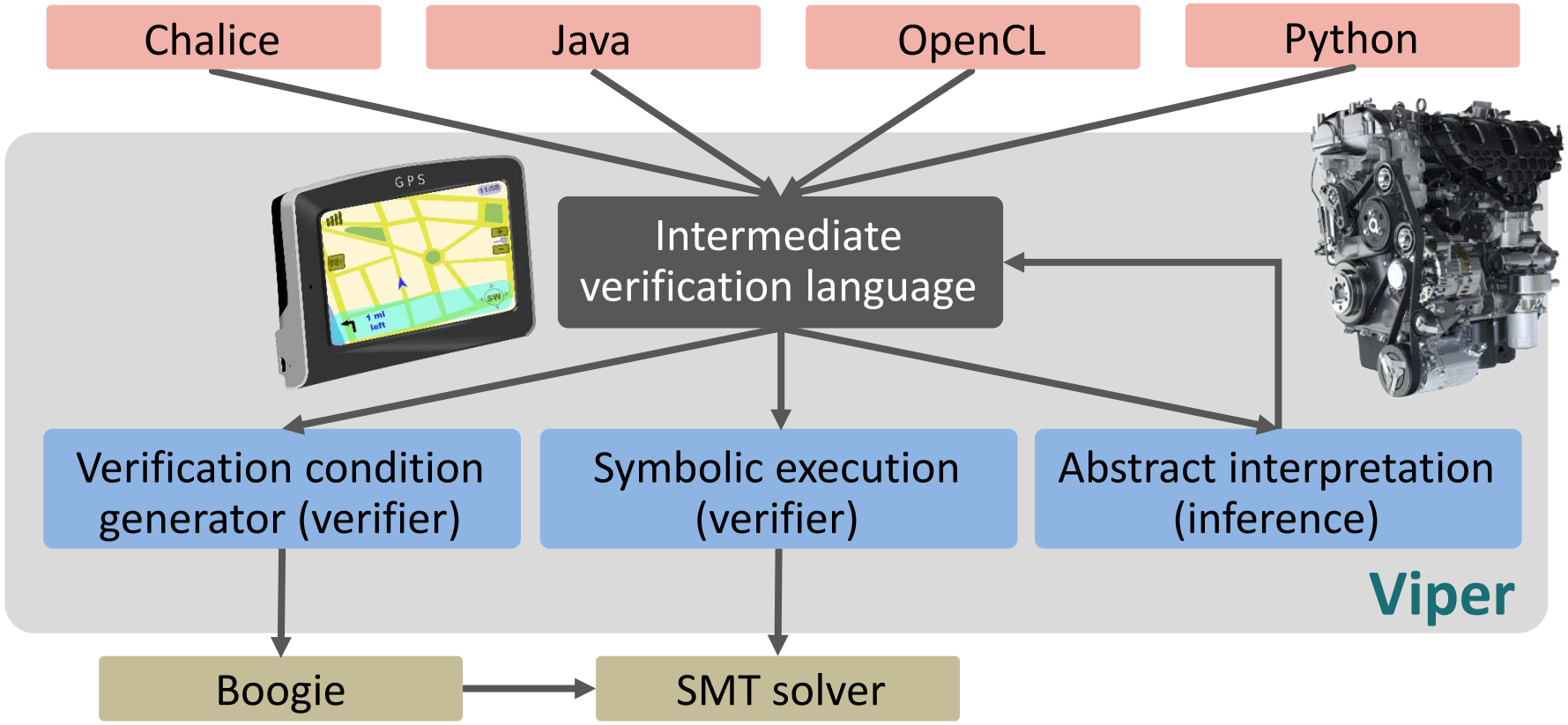
Viper

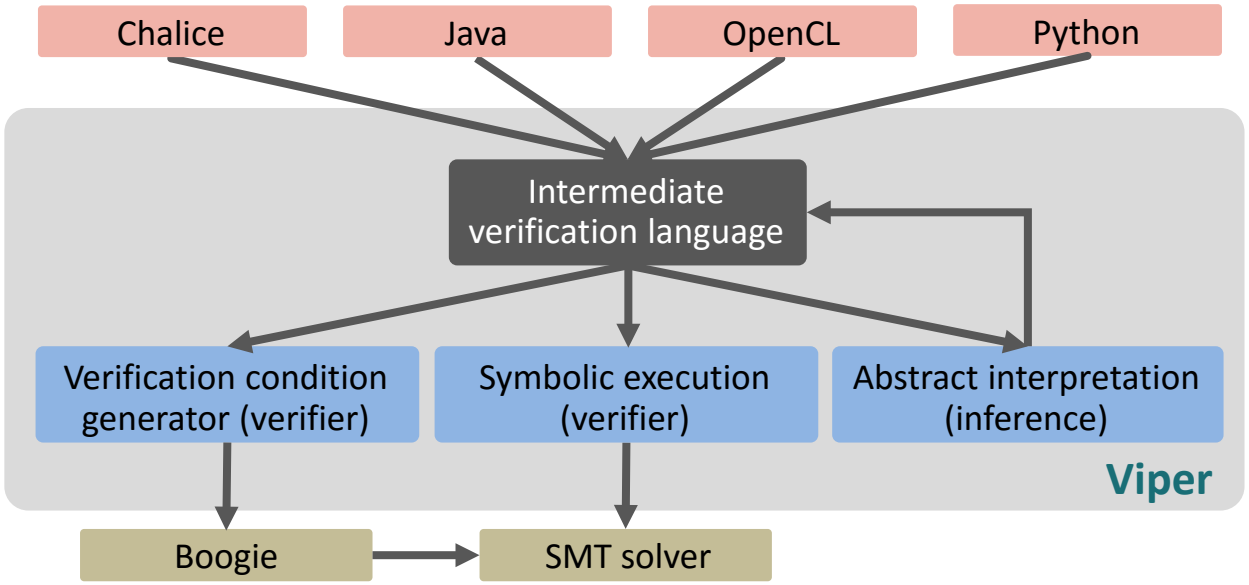

Front ends
Intermediate verification language
Verification condition generator (verifier)
Symbolic execution (verifier)
Abstract interpretation (inference)
Viper
Boogie
SMT solver

Viper


Chalice
Java
OpenCL
C#
Python
Intermediate verification language
Verification condition generator (verifier)
Symbolic execution (verifier)
Abstract interpretation (inference)
Viper
Boogie
SMT solver

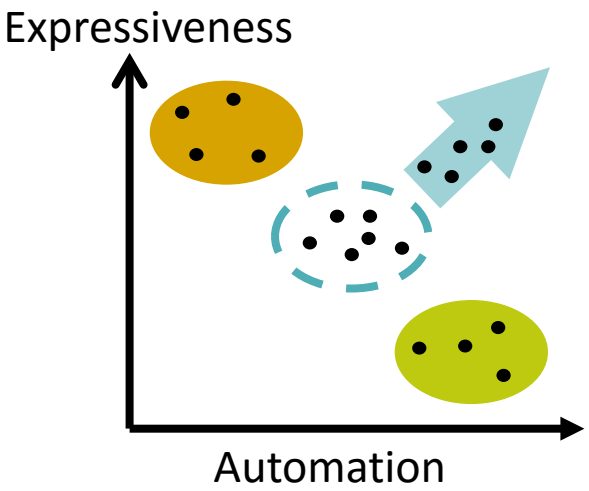

Expressiveness
Automation


Chalice
Java
OpenCL
Python
GPS
Intermediate verification language
Verification condition generator (verifier)
Symbolic execution (verifier)
Abstract interpretation (inference)
Viper
Boogie
SMT solver



VIPER

http://viper.ethz.ch



backup slides

1

Explicit Footprints: Example

```
class Account {
  bal: Int

  method deposit(n: Int)
    requires 0 < n
    modifies {this}
  { bal := bal + n }
}
```

```
class List {
  len: Int

  function length(): Int
    reads {this}
  { len }
}
```

Explicit Footprints: Example

```
class Account {
  bal: Int

  method deposit(n: Int)
    requires 0 < n
    modifies {this}
  { bal := bal + n }
}
```

```
class List {
  len: Int

  function length(): Int
    reads {this}
  { len }
}
```

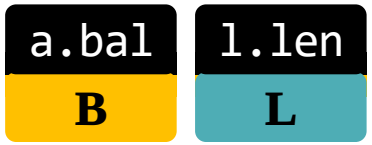
```
method client(a: Account, l: List)
{
  var tmp := l.length()
  a.deposit(200)
  assert tmp == l.length()
}
```

$$\frac{\{\,P\,\}\ C\ \{\,Q\,\} \qquad \text{modifies}(C) \cap \text{reads}(R) = \emptyset}{\{\,P \wedge R\,\}\ C\ \{\,Q \wedge R\,\}}$$

2

Permission Transfer

```
method deposit(n: Int)
  requires 0 < n ∧ acc(bal)
  ensures  acc(bal)
```




```
method client(a: Account, l: List)
  requires acc(a.bal) * acc(l.len)
{
  var tmp := l.length()

  a.deposit(200)

  assert tmp == l.length()
}
```

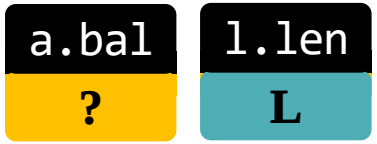
Permission Transfer

```
method deposit(n: Int)
  requires 0 < n ∧ acc(bal)
  ensures  acc(bal)
```


```
method client(a: Account, l: List)
  requires acc(a.bal) * acc(l.len)
{
  var tmp := l.length()

  a.deposit(200)

  assert tmp == l.length()
}
```

Permission Transfer

```
method deposit(n: Int)
  requires 0 < n ∧ acc(bal)
  ensures  acc(bal)
```




```
method client(a: Account, l: List)
  requires acc(a.bal) * acc(l.len)
{
  var tmp := l.length()

  a.deposit(200)

  assert tmp == l.length()
}
```

Inhale and Exhale

```
method deposit(n: Int)
  requires 0 < n ∧ acc(bal)
  ensures  acc(bal)
```

```
method client(a: Account, l: List)
{
  inhale acc(a.bal) * acc(l.len)
  var tmp := l.length()
  exhale 0 < 200 ∧ acc(a.bal)
    // a.deposit(200)
  inhale acc(a.bal)
  assert tmp == l.length()
}
```

3

Permissions in Viper

Permissions denoted by accessibility predicates

```
acc(e.f)
```

Expressions may be heap-dependent

```
acc(e.f) && e.f > 0
```

Support for fractional permissions

```
acc(e.f, ½)
```

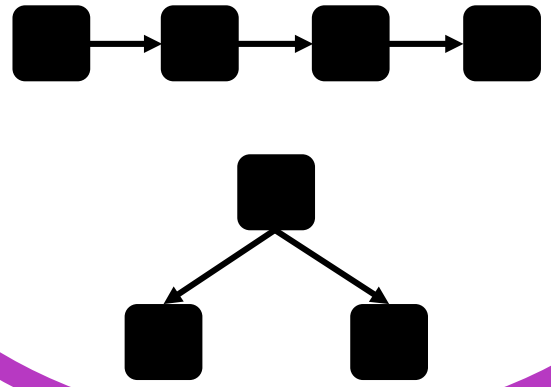
Conjunction is multiplicative (as in separation logic)

```
acc(e.f, ½) && acc(e.f, ½)
```

4

Unbounded Data Structures

Unidirectional

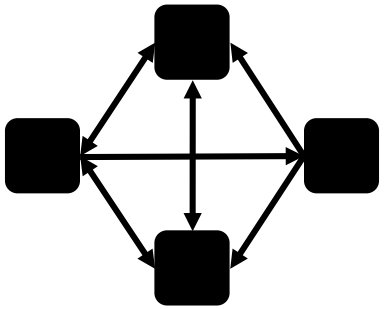


Multi-directional

Random Access

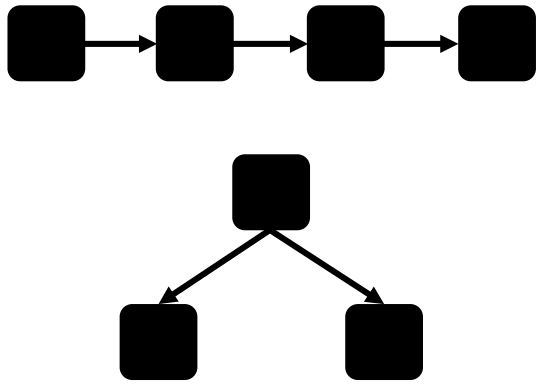
specify via
recursive predicate

Unstructured



Unbounded Data Structures

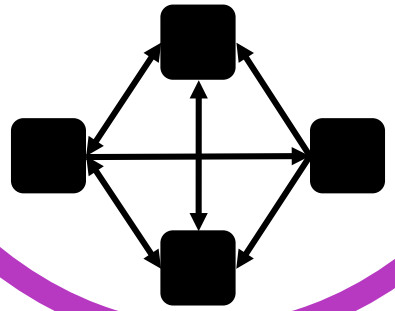
Unidirectional



Multi-directional

Random Access

Unstructured



specify via
iterated separating
conjunction