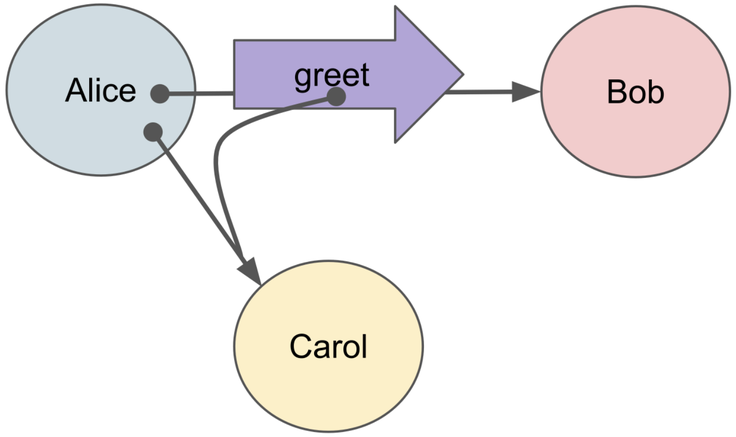
VeTSS 2024

Reasoning with Object Capabilities

Sophia Drossopoulou, Imperial College London



James Noble

Susan Eisenbach

Julian Mackay

need new encapsulation features

need new encapsulation features

Ownership types?

need new encapsulation features

Ownership types?

… WHAT should these features guarantee?

Background,
Problem,
OCAP,
Our remit

The Background:

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

B2: External objects may call methods on internal objects.

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

B2: External objects may call methods on internal objects.

B3: Internal objects may call methods on external objects.

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

B2: External objects may call methods on internal objects.

B3: Internal objects may call methods on external objects.

The Problem: Mitigate the arising uncertainty.

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

B2: External objects may call methods on internal objects.

B3: Internal objects may call methods on external objects.

The Problem: Mitigate the arising uncertainty.

OCAP Solution: Capability … transferable right to perform an operation; capability is a reference; it cannot be forged; tranferred only through call

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

B2: External objects may call methods on internal objects.

B3: Internal objects may call methods on external objects.

The Problem: Mitigate the arising uncertainty.

OCAP Solution: Capability … transferable right to perform an operation; capability is a reference; it cannot be forged; tranferred only through call

Our remit:

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

B2: External objects may call methods on internal objects.

B3: Internal objects may call methods on external objects.

The Problem: Mitigate the arising uncertainty.

OCAP Solution: Capability … transferable right to perform an operation; capability is a reference; it cannot be forged; tranferred only through call

Our remit:

R1: Specify that external access to capability necessary for effect.

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

B2: External objects may call methods on internal objects.

B3: Internal objects may call methods on external objects.

The Problem: Mitigate the arising uncertainty.

OCAP Solution: Capability … transferable right to perform an operation; capability is a reference; it cannot be forged; tranferred only through call

Our remit:

R1: Specify that external access to capability necessary for effect.

R2: Prove module's adherence to specification.

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

B2: External objects may call methods on internal objects.

B3: Internal objects may call methods on external objects.

The Problem: Mitigate the arising uncertainty.

OCAP Solution: Capability … transferable right to perform an operation; capability is a reference; it cannot be forged; tranferred only through call

Our remit:

R1: Specify that external access to capability necessary for effect.

R2: Prove module's adherence to specification.

R3: Prove calls from internal to external object.

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

B2: External objects may call methods on internal objects.

B3: Internal objects may call methods on external objects.

The Problem: Mitigate the arising uncertainty.

OCAP Solution: Capability … transferable right to perform an operation; capability is a reference; it cannot be forged; tranferred only through call

Our remit:

R1: Specify that external access to capability necessary for effect.

R2: Prove module's adherence to specification.

R3: Prove calls from internal to external object.

Not our remit: Forbid external access to capabilty.

Background,
Problem,
OCAP,
Our remit

— in a diagram —

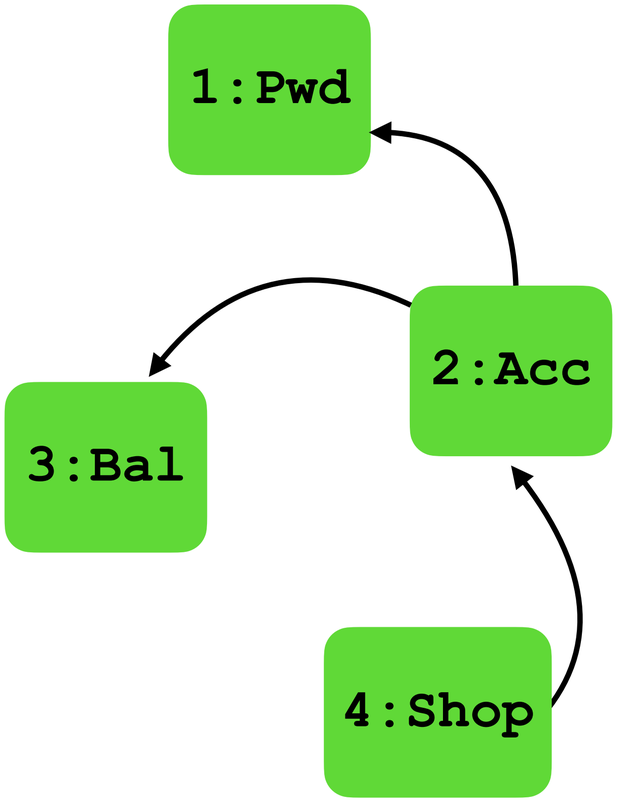
The Background:

The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.

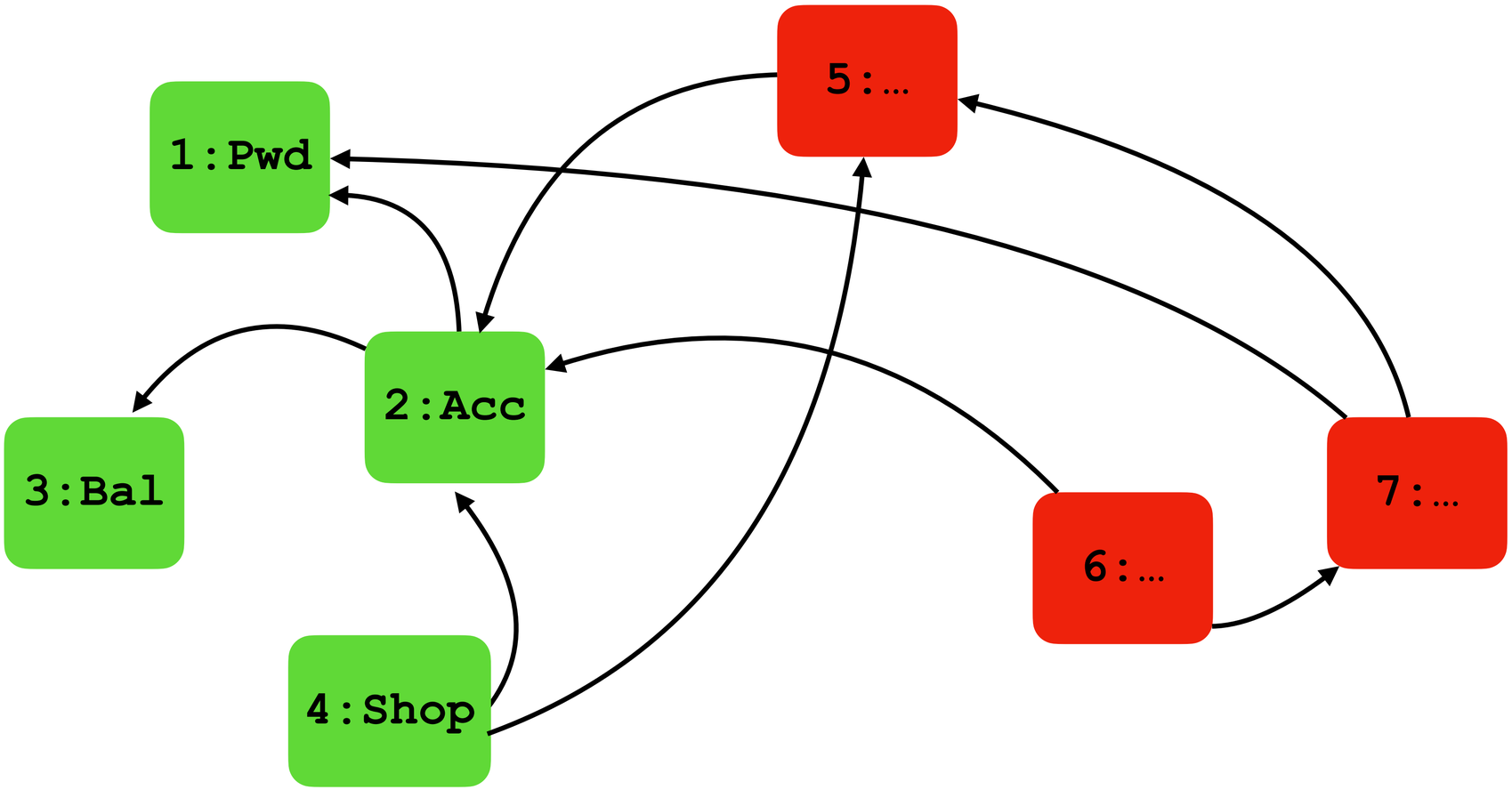
The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.



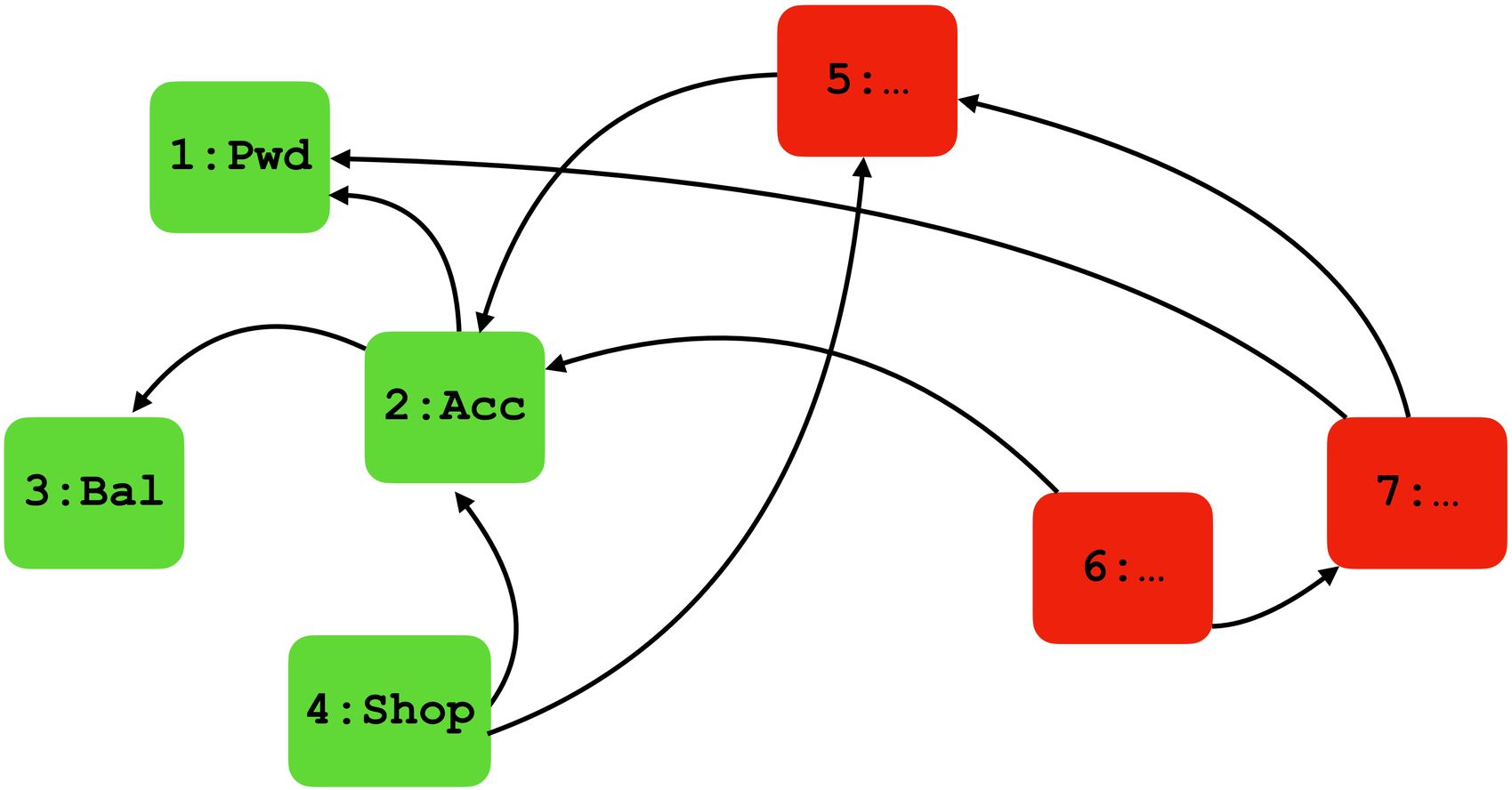
The Background:

B1: Internal (trusted) and External (untrusted) objects are intertwined.



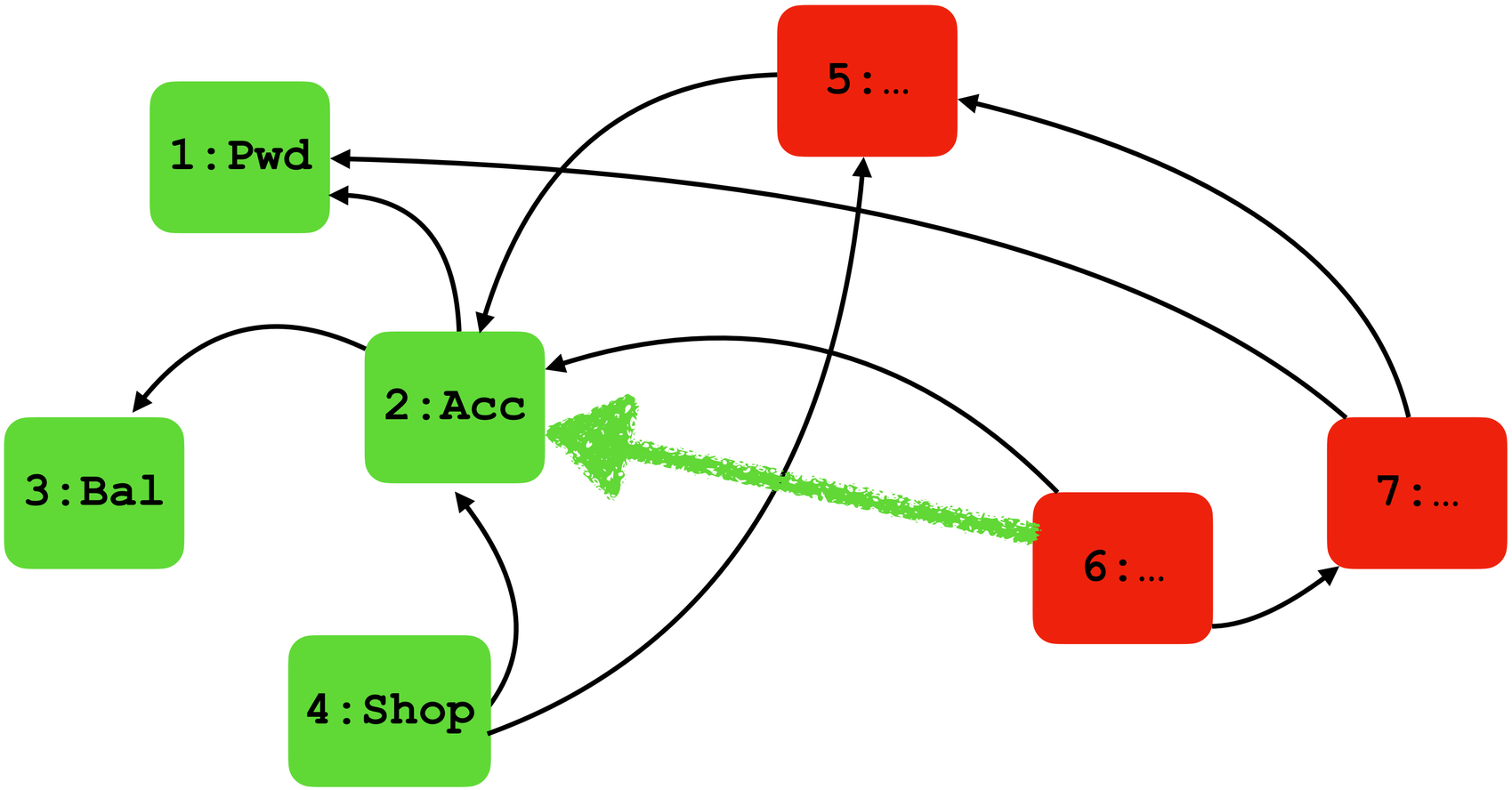
The Background:

B2: External objects may call methods on internal objects.



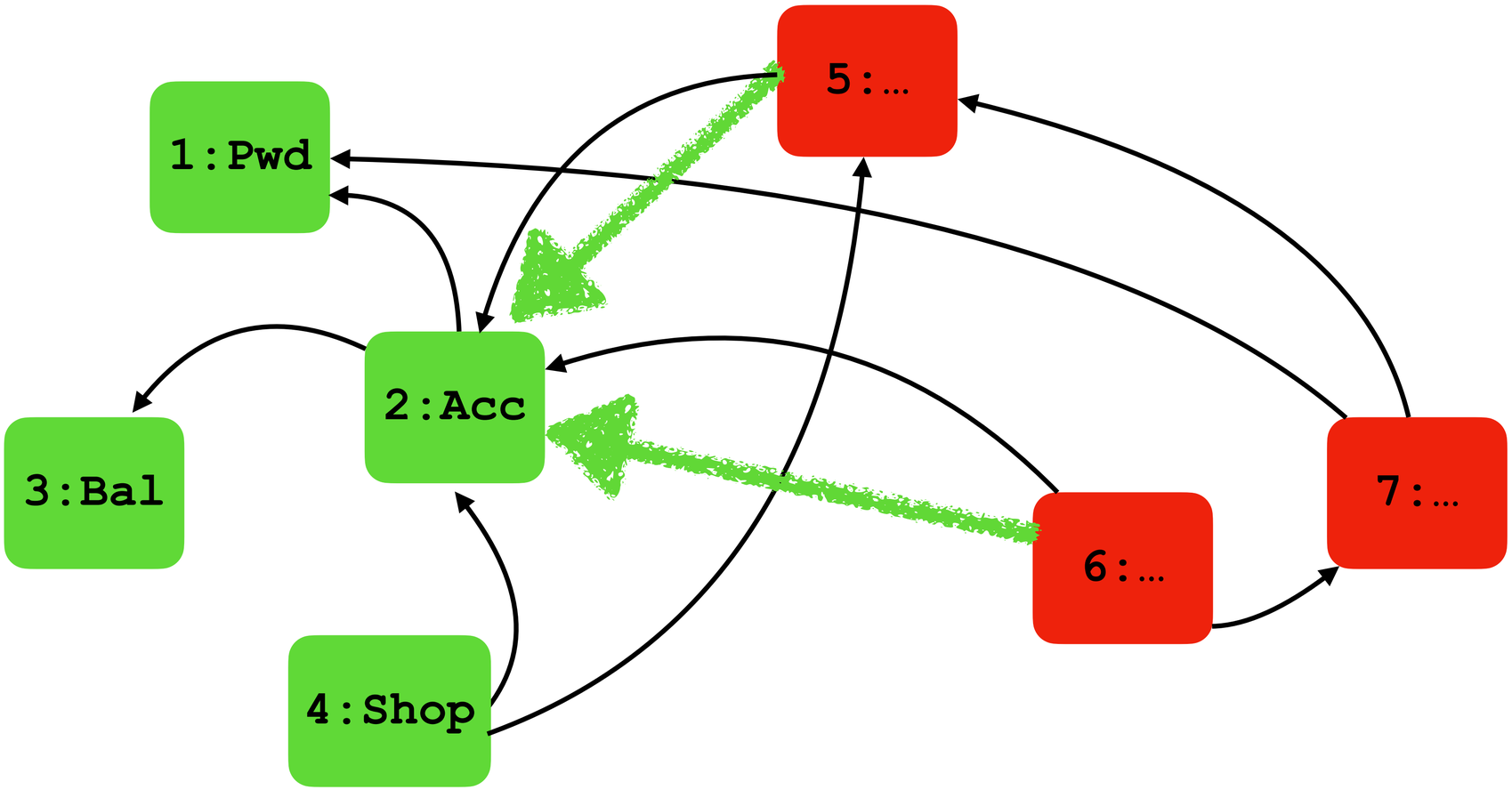
The Background:

B2: External objects may call methods on internal objects.



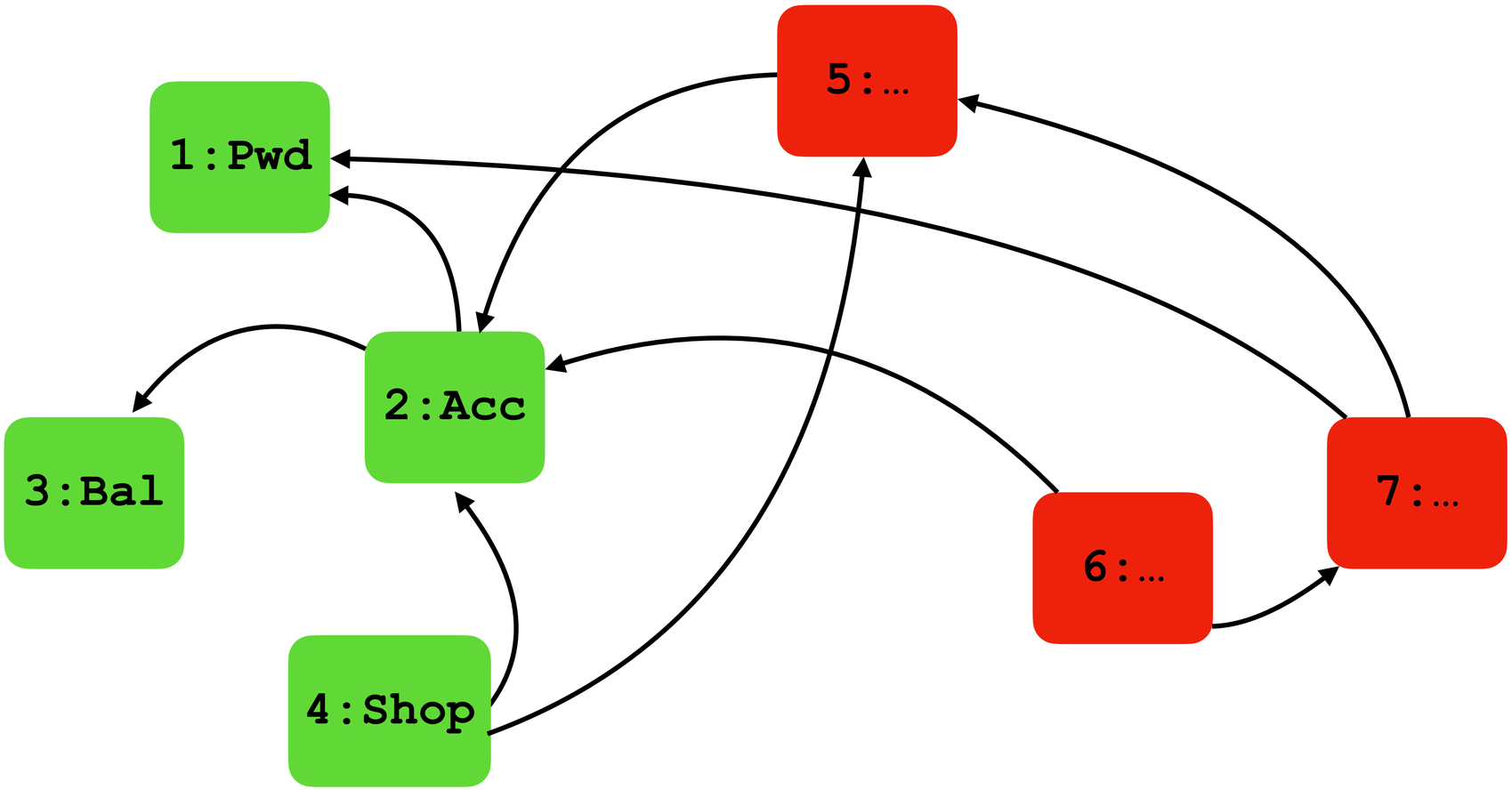
The Background:

B2: External objects may call methods on internal objects.



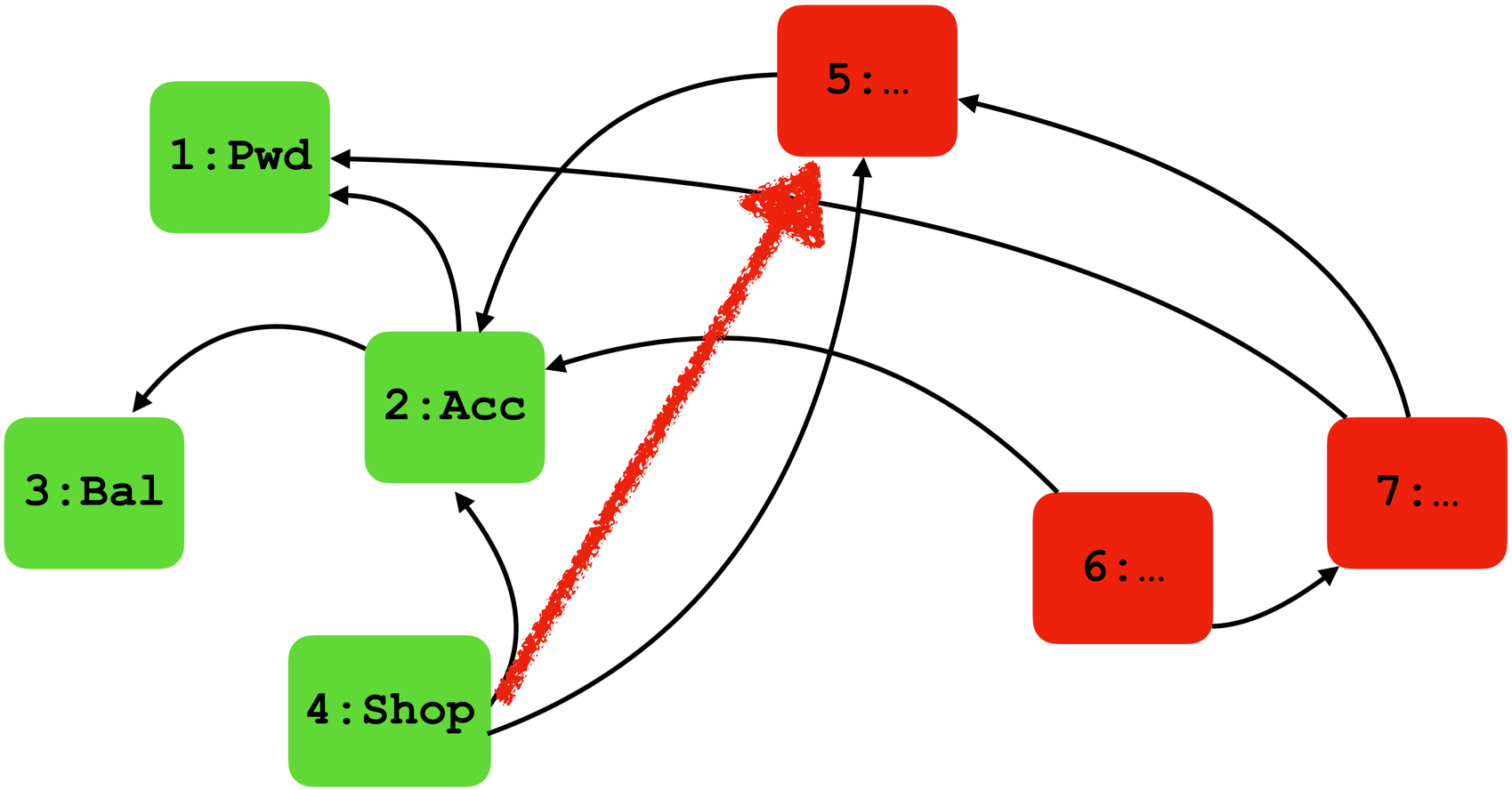
The Background:

B3: Internal objects may call methods on external objects.



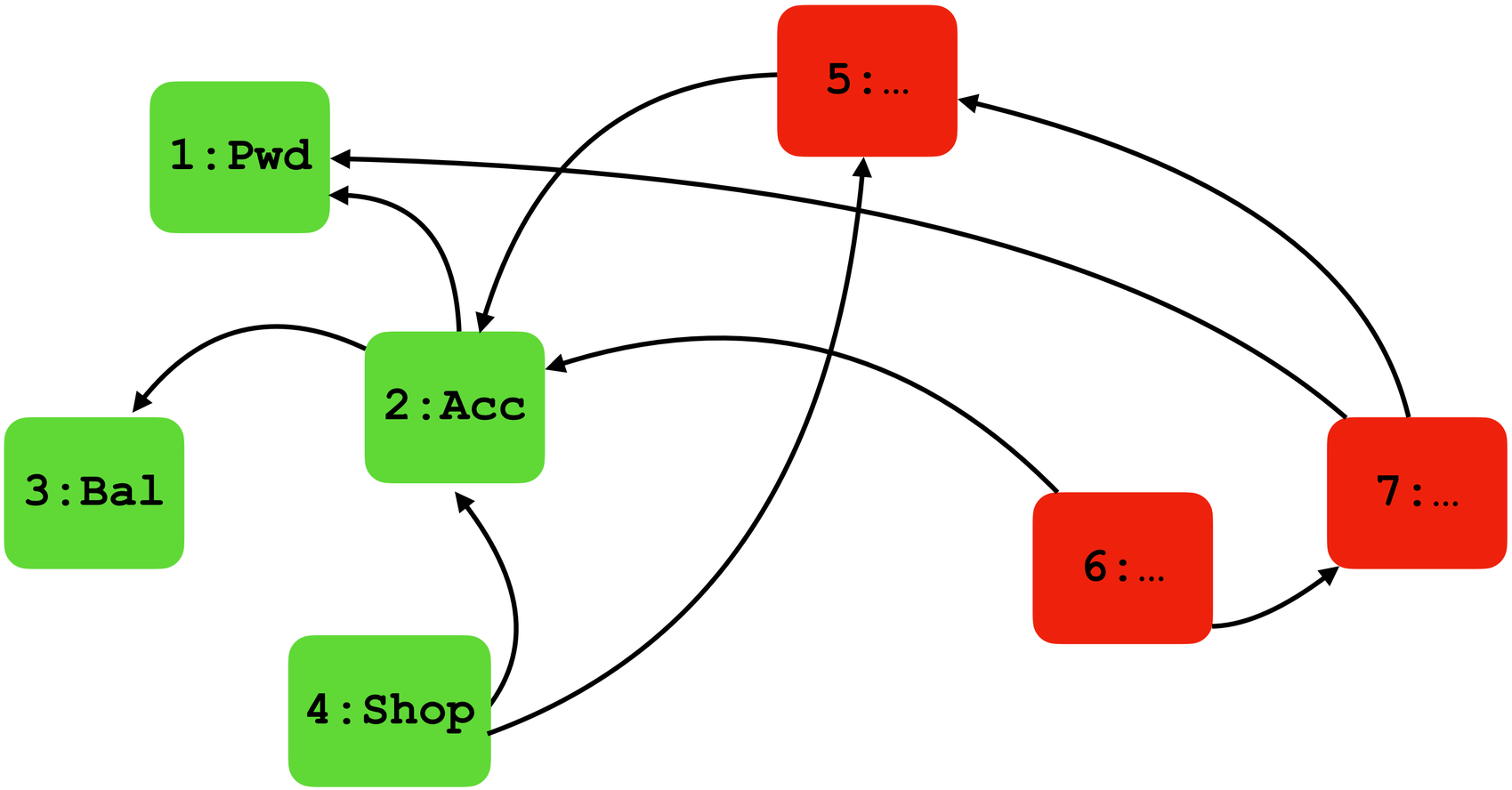
The Background:

B3: Internal objects may call methods on external objects.



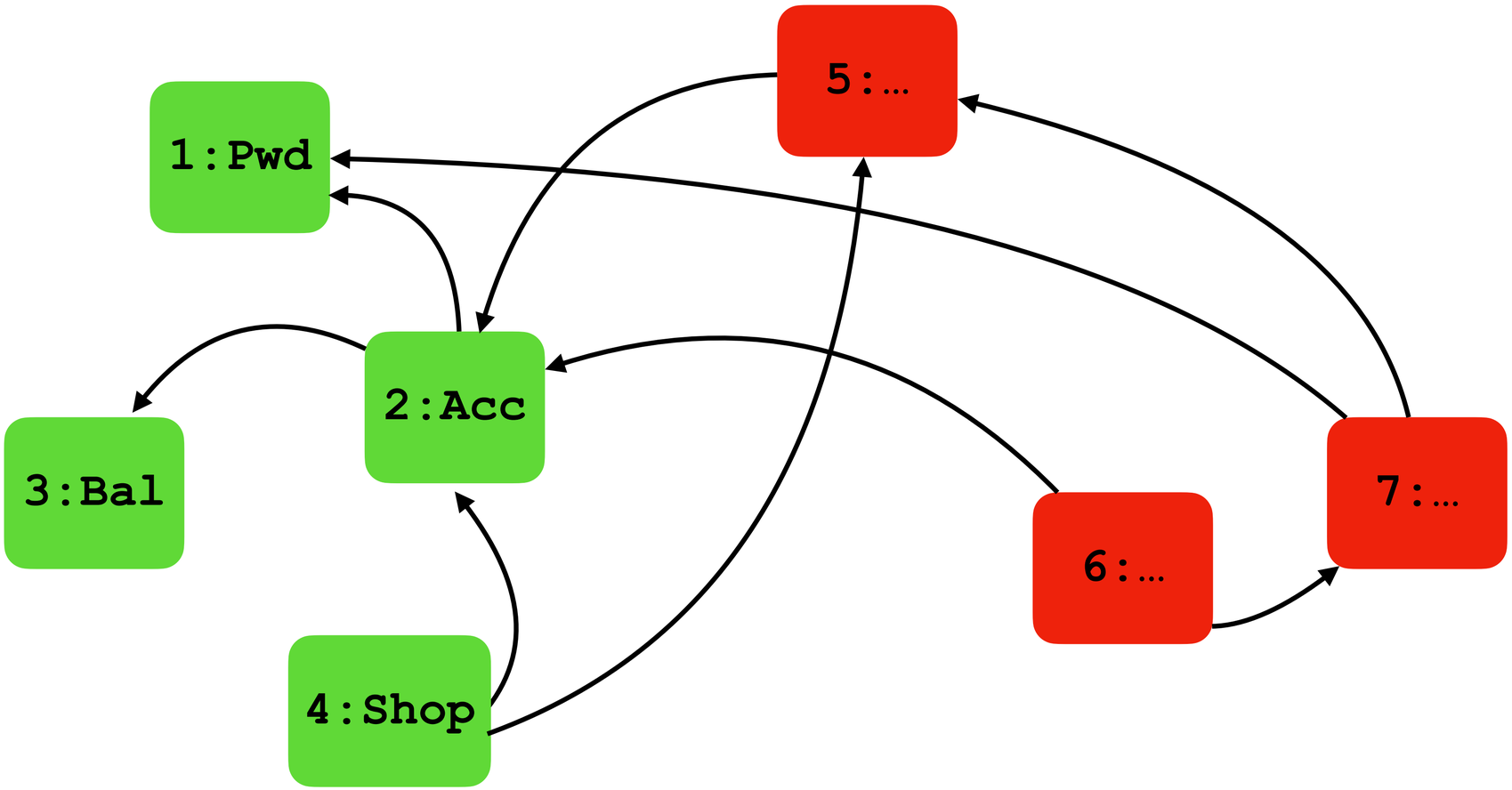
The Problem:

OCAP Solution:



The Problem: Mitigate the arising uncertainty.

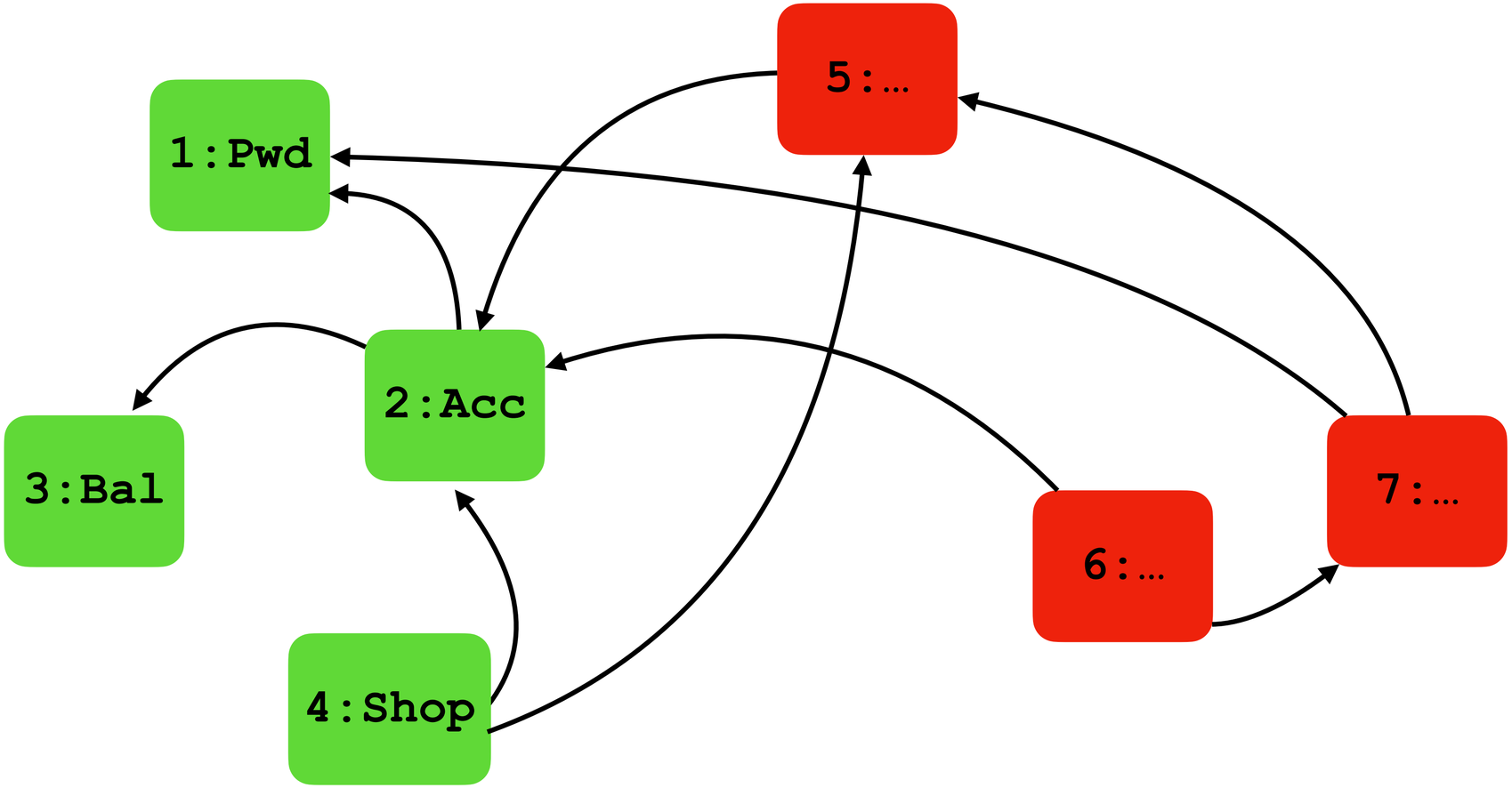
OCAP Solution: Capability … transferable right to perform an operation; capability is a reference; it cannot be forged; tranferred only through call



The

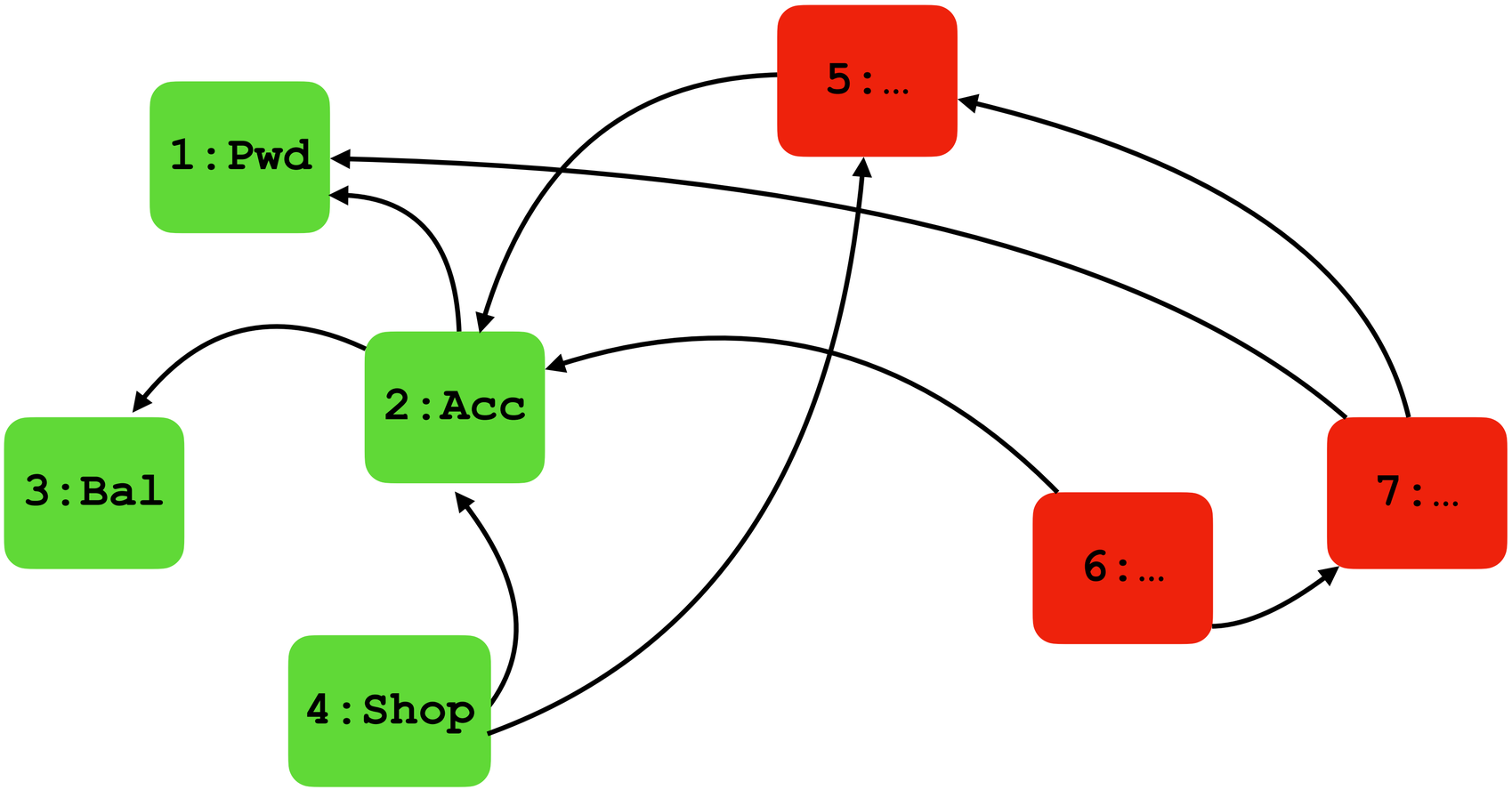
OCA

Capability as *guard* for a certain effect.



Our remit:

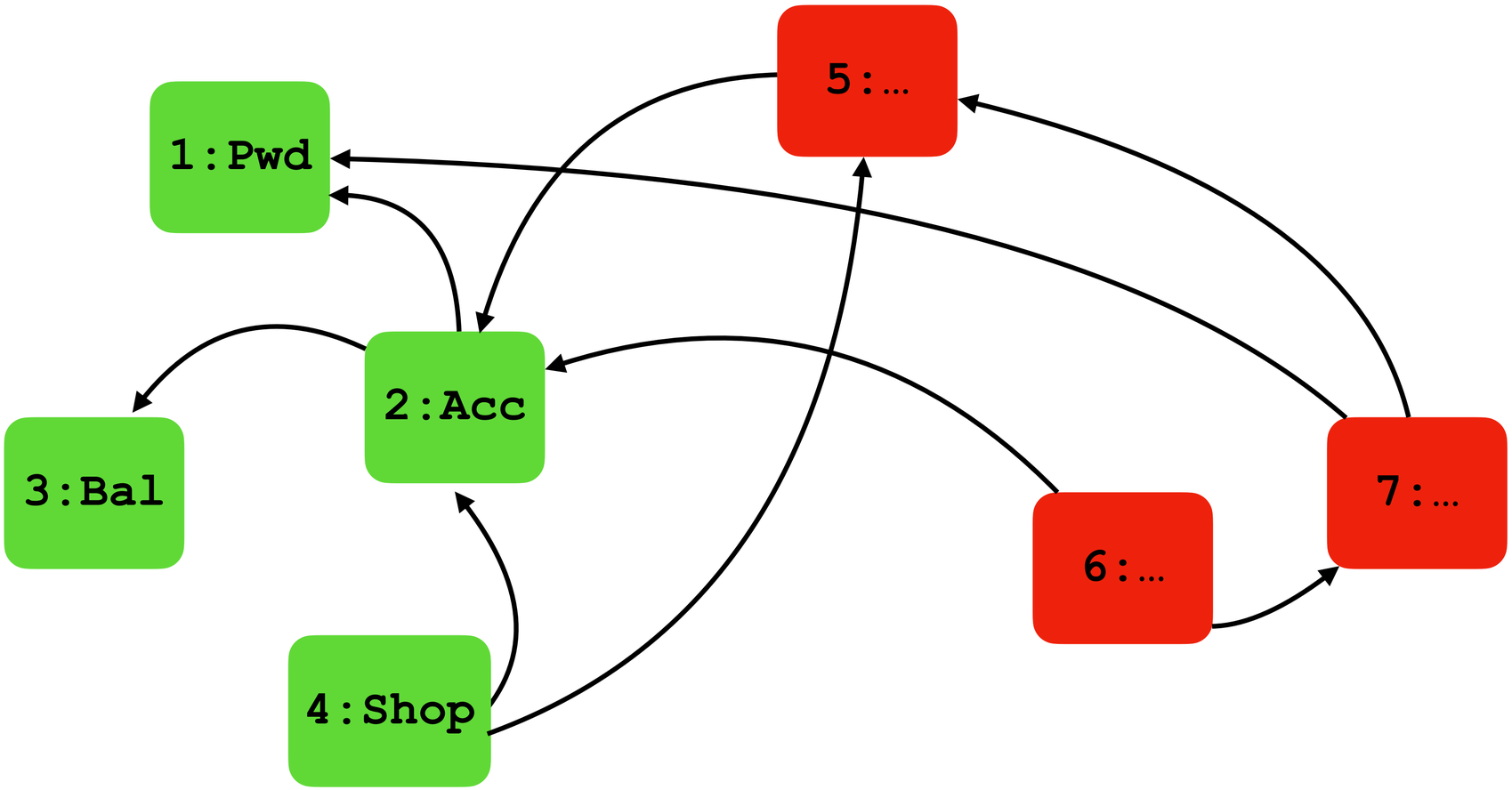
R1: Specify that eventlual. external access to capability necessary for effect.



Our remit:

R1: Specify that eventlual. external access to capability necessary for effect.

S1: without eventl. access to account no change in balance

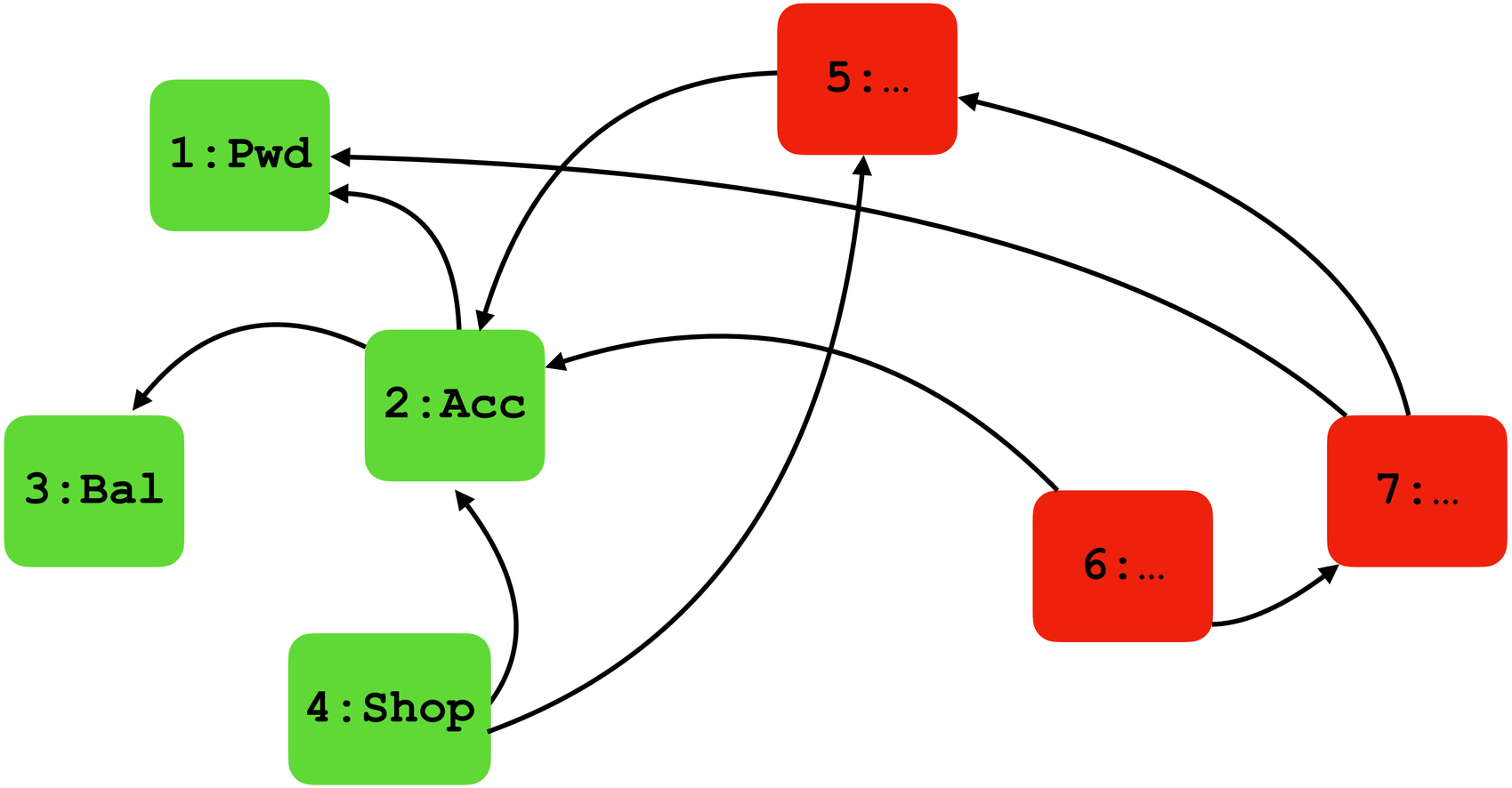


Our remit:

R1: Specify that eventlual. external access to capability necessary for effect.

S1: without eventl. access to account no change in balance

S2: without eventl. access to password no decrease in balance

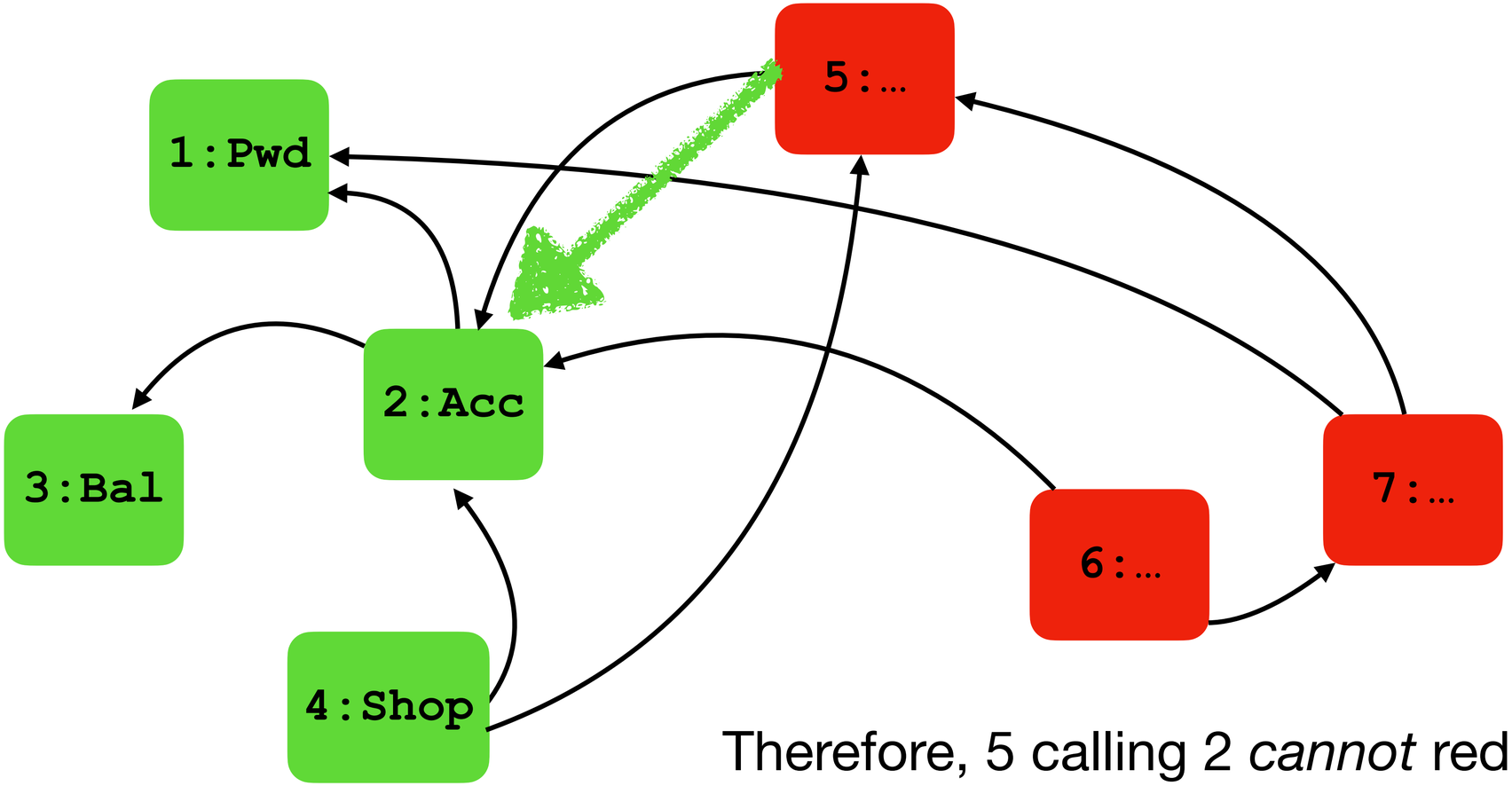


Our remit:

R1: Specify that eventlual. external access to capability necessary for effect.

S1: without eventl. access to account no change in balance

S2: without eventl. access to password no decrease in balance



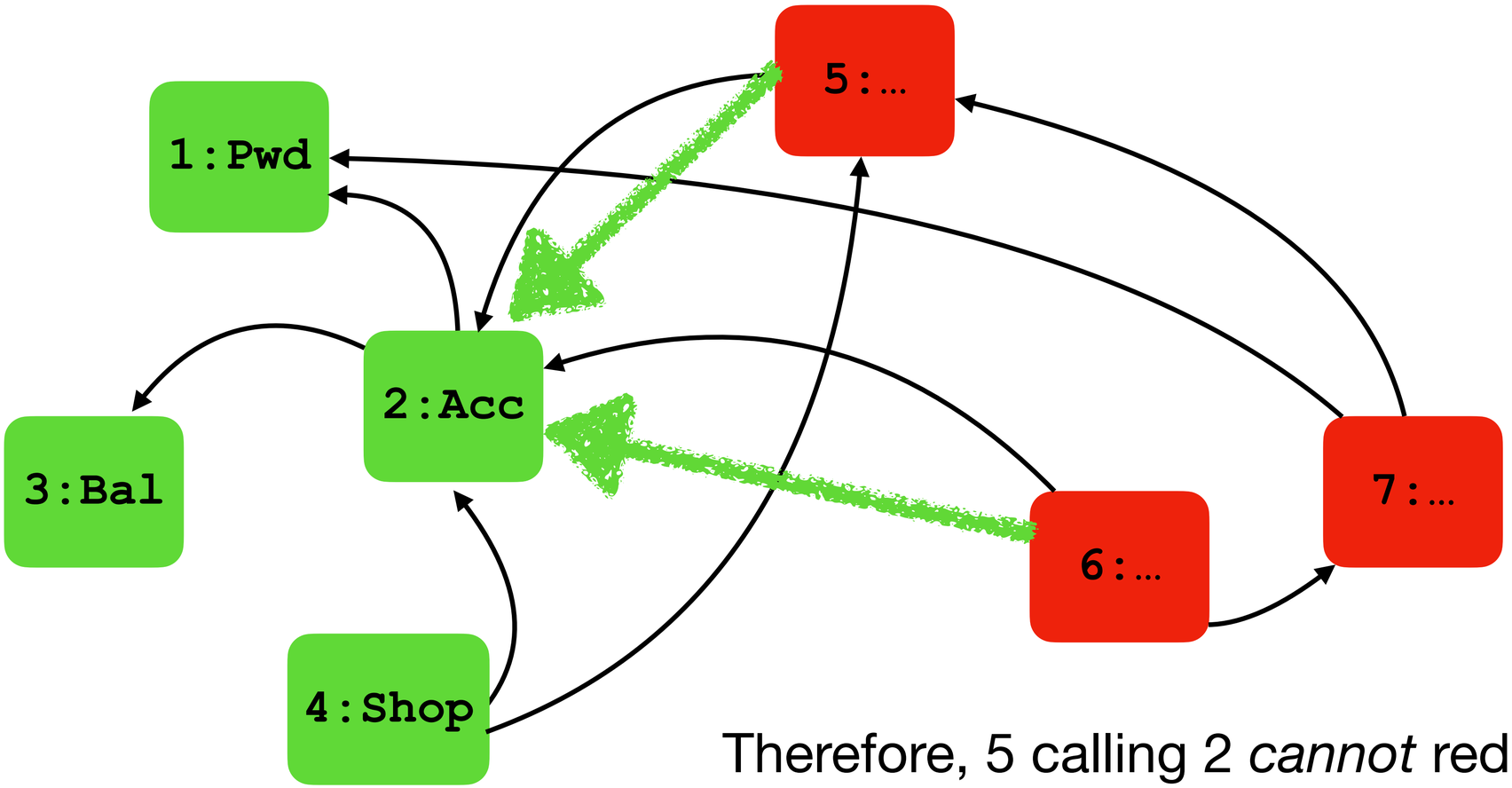
Therefore, 5 calling 2 *cannot* reduce 2.balance.

Our remit:

R1: Specify that eventlual. external access to capability necessary for effect.

S1: without eventl. access to account no change in balance

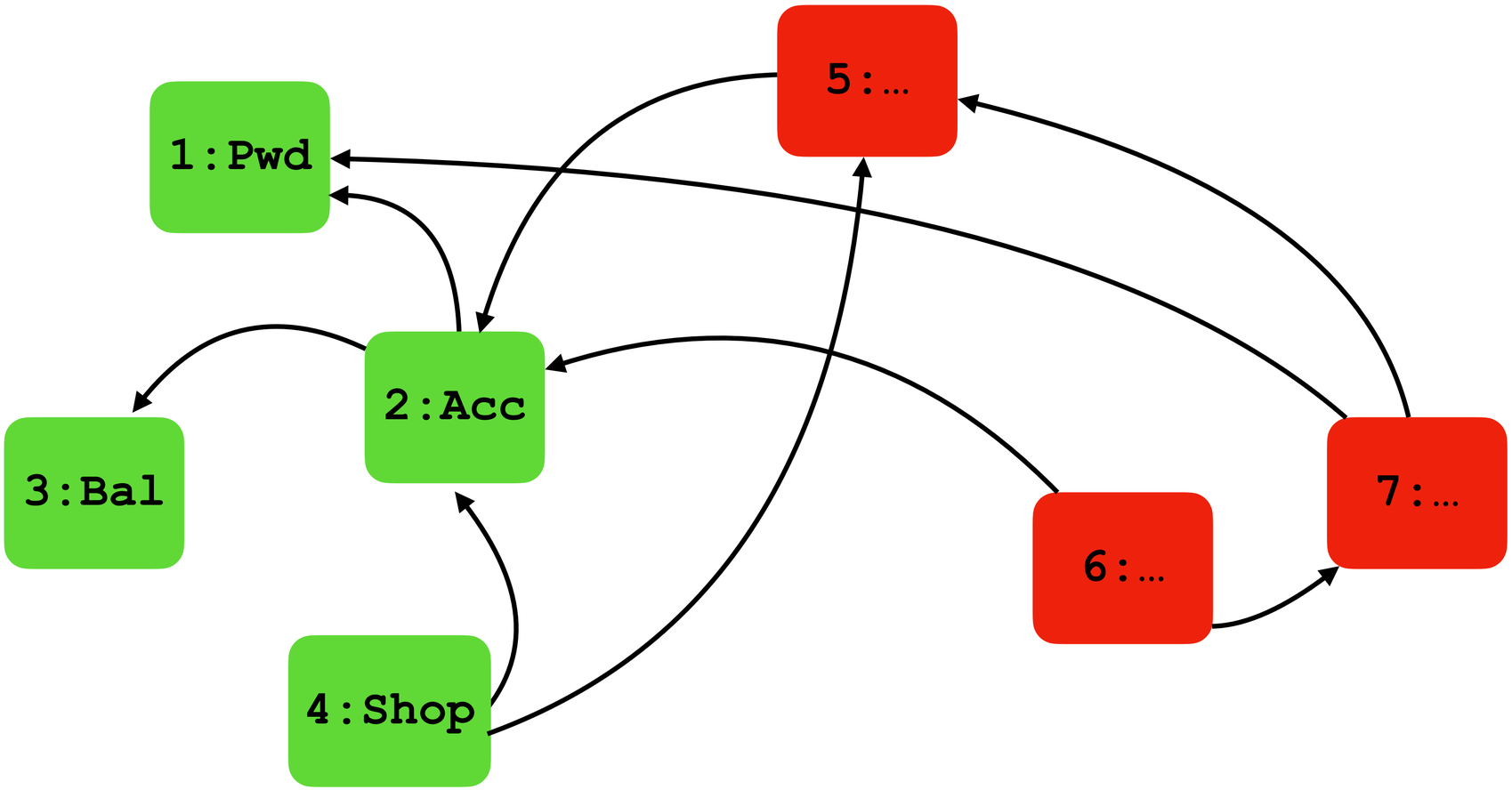
S2: without eventl. access to password no decrease in balance



Therefore, 5 calling 2 *cannot* reduce 2.balance.

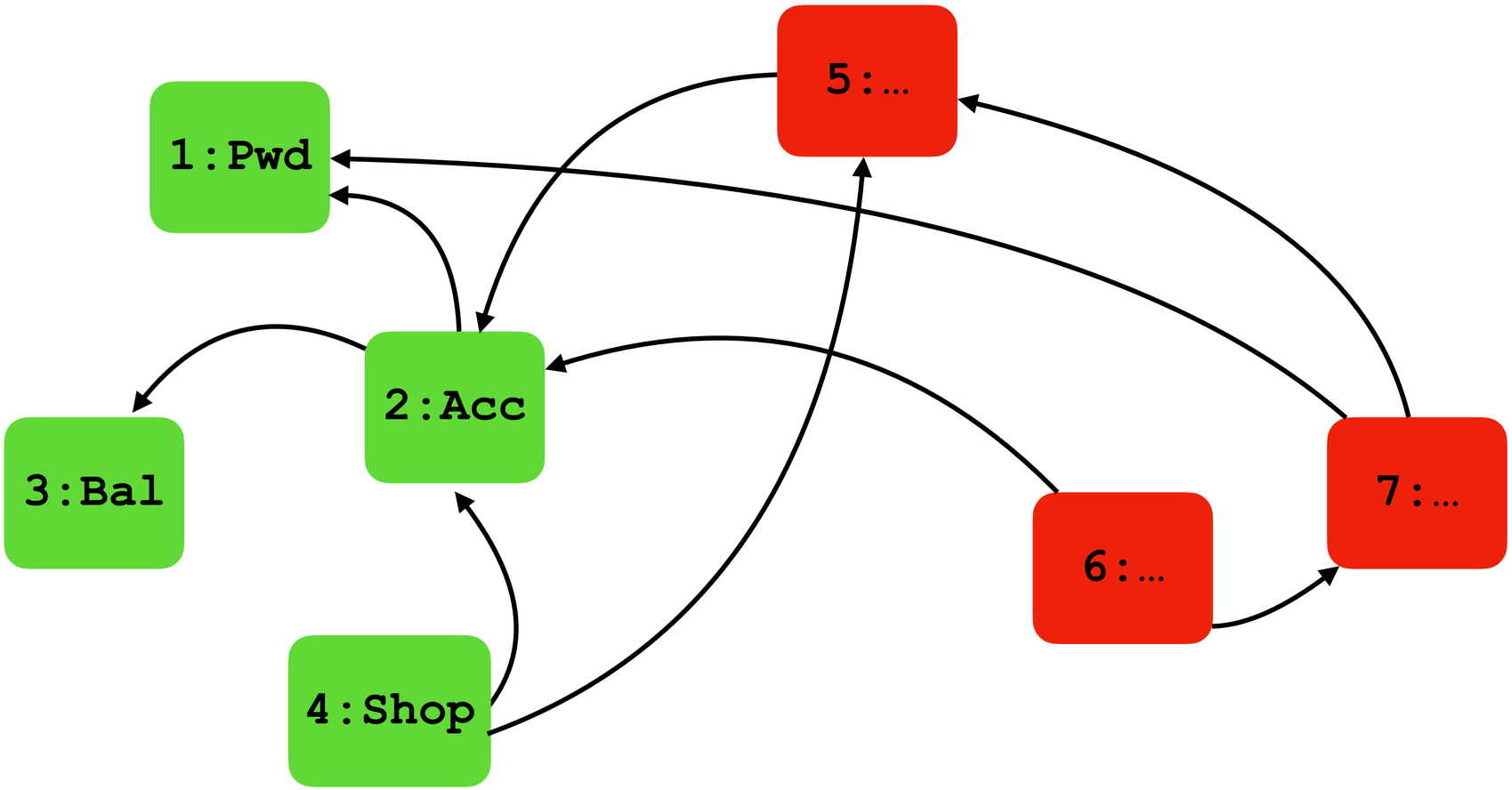
But, 6 calling 2 *may* reduce 2.balance.

Our remit:



Our remit:

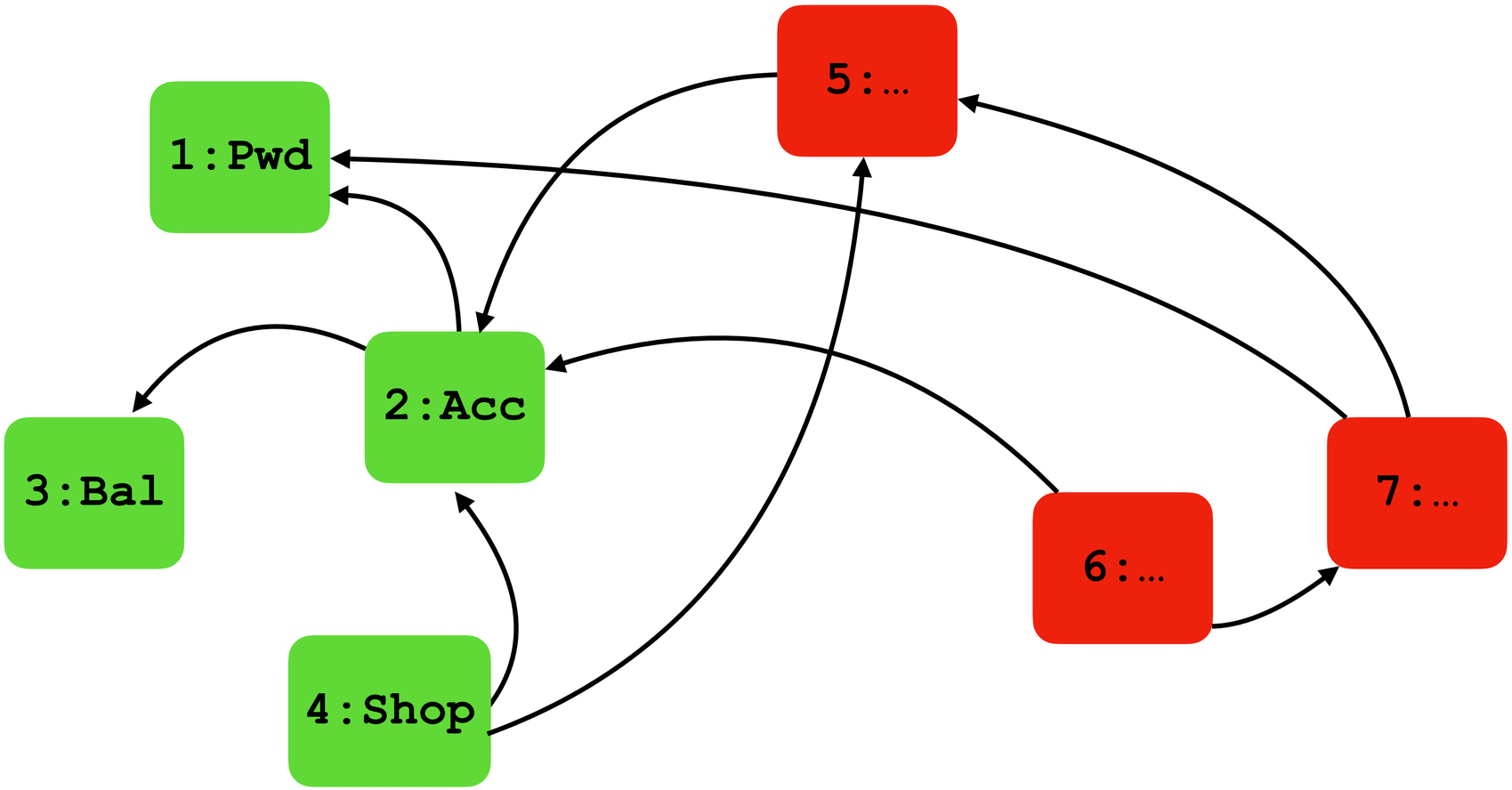
R2: Prove module's adherence to specification — later.



S2: without access to password no decrease in balance

Our remit:

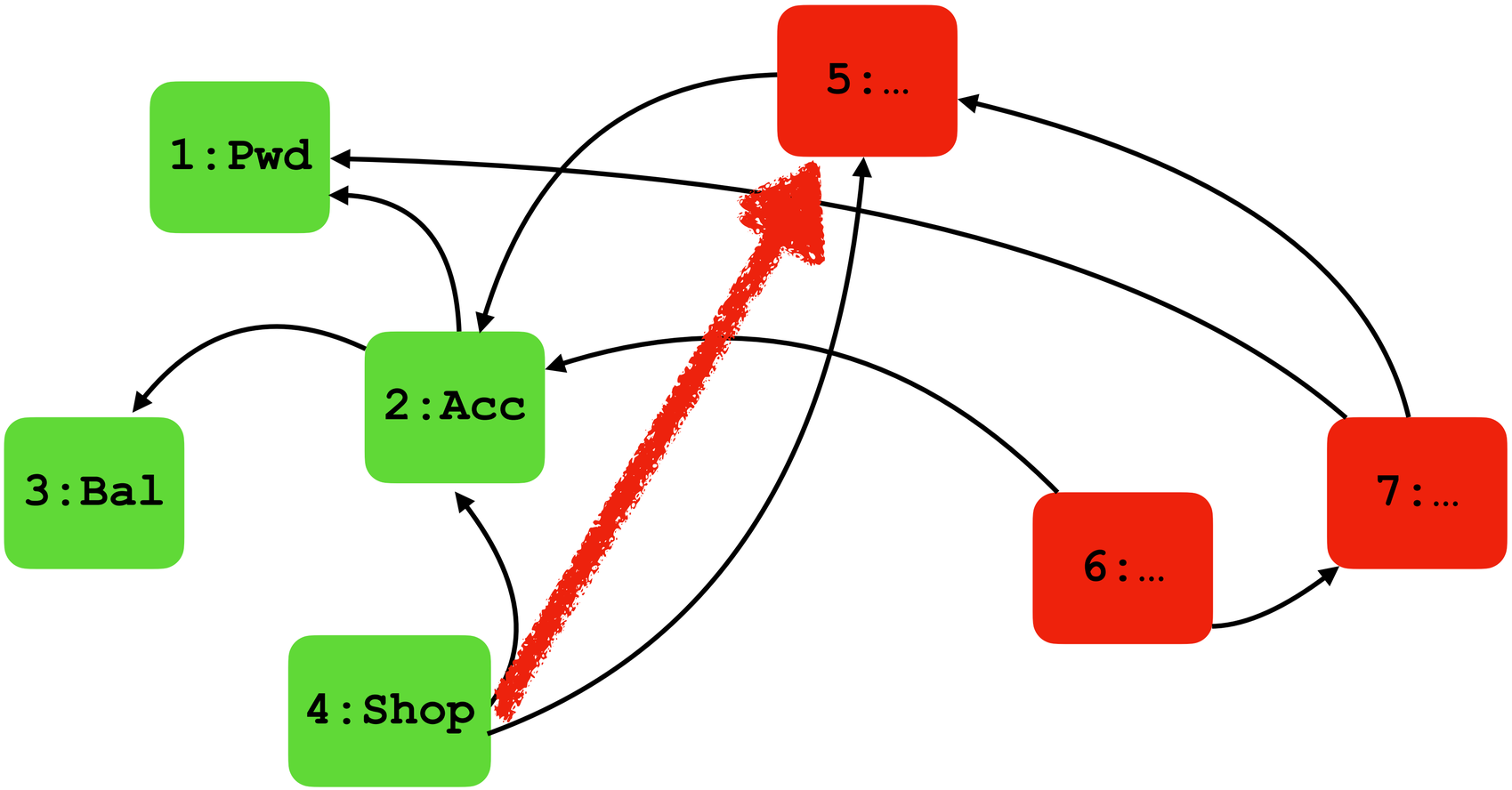
R3: Prove calls from internal to external object.



S2: without access to password no decrease in balance

Our remit:

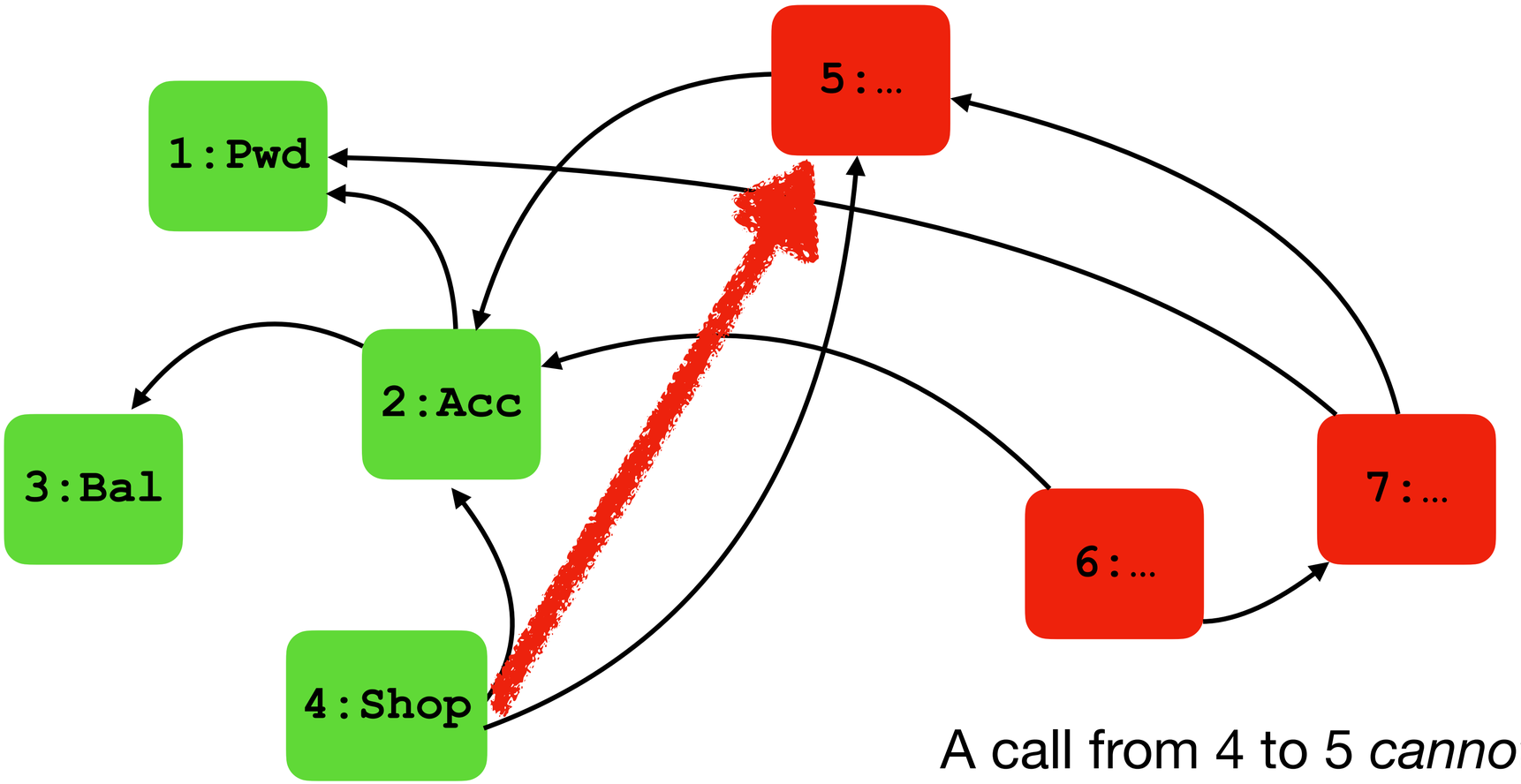
R3: Prove calls from internal to external object.



S2: without access to password no decrease in balance

Our remit:

R3: Prove calls from internal to external object.

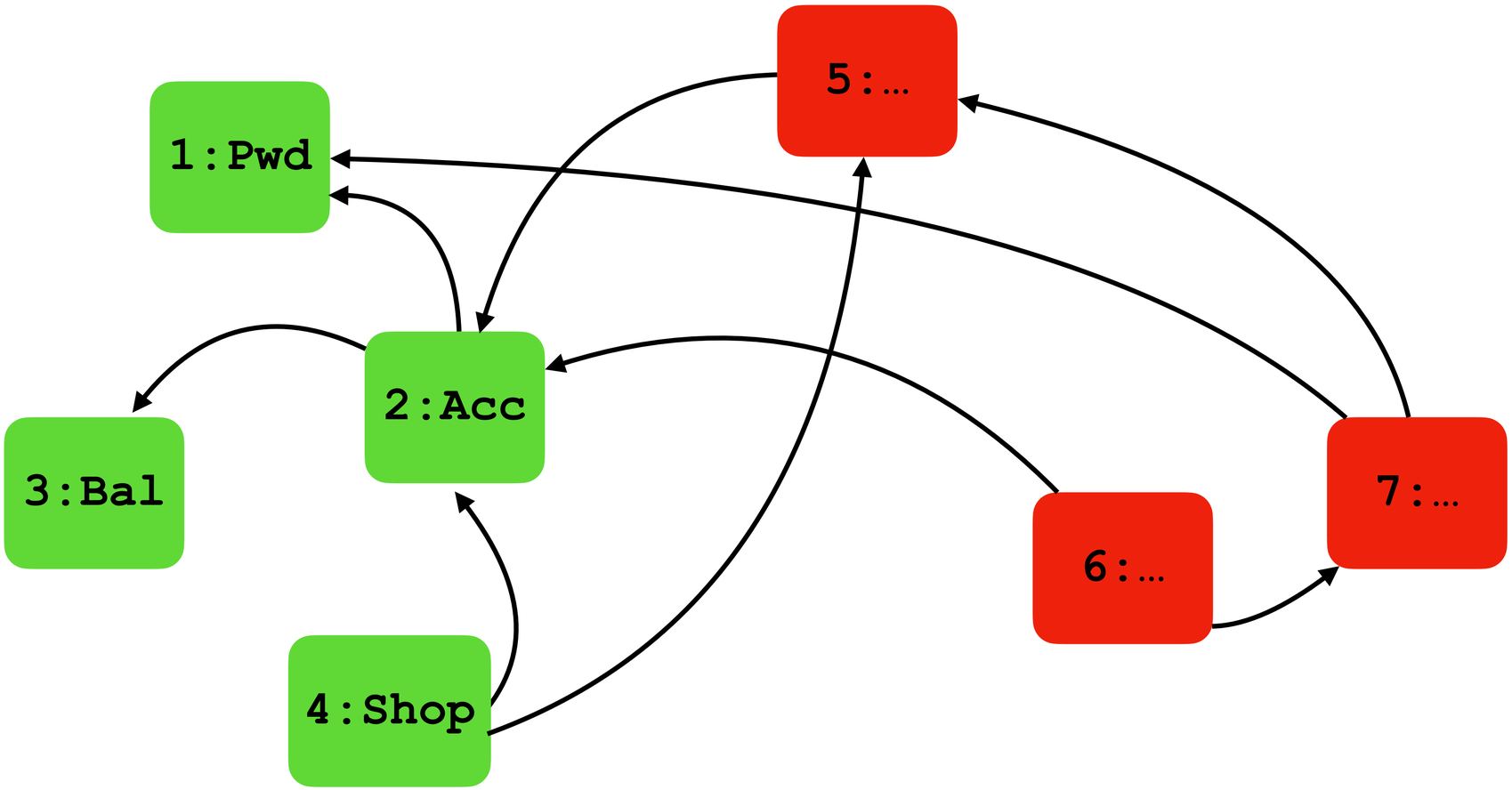


A call from 4 to 5 *cannot* reduce 2.balance

S2: without access to password no decrease in balance

Our remit:

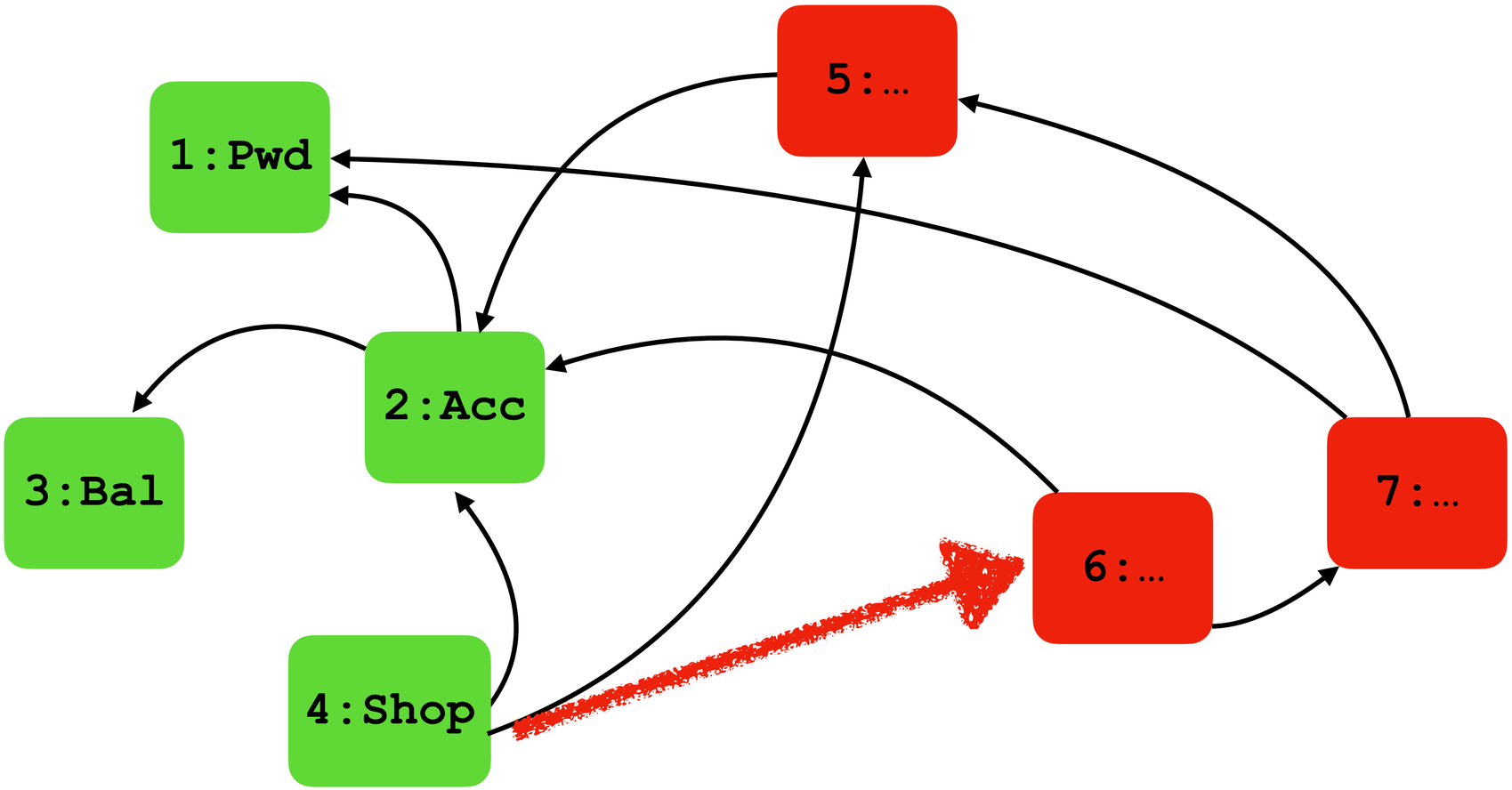
R3: Prove calls from internal to external object.



S2: without access to password no decrease in balance

Our remit:

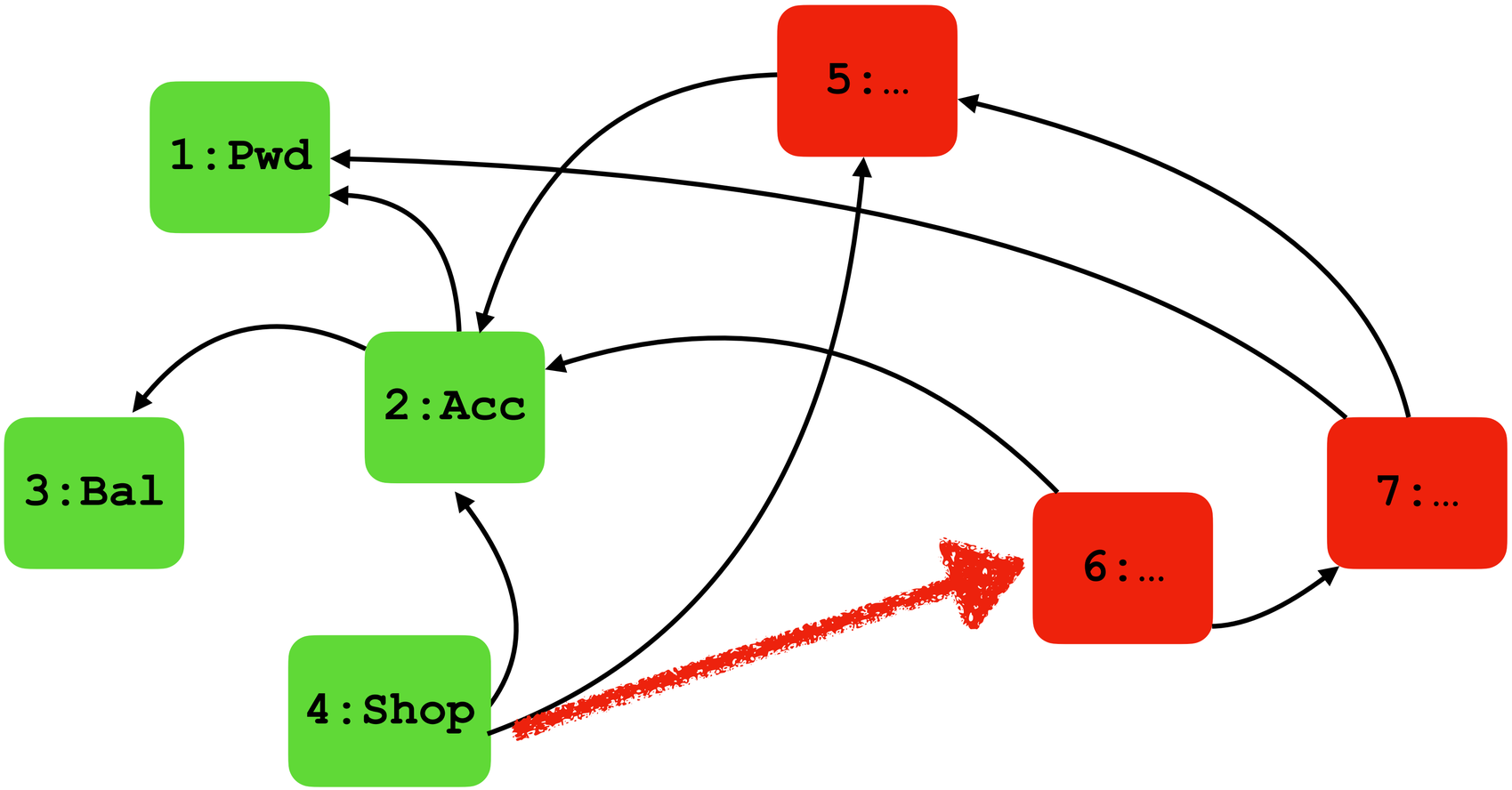
R3: Prove calls from internal to external object.



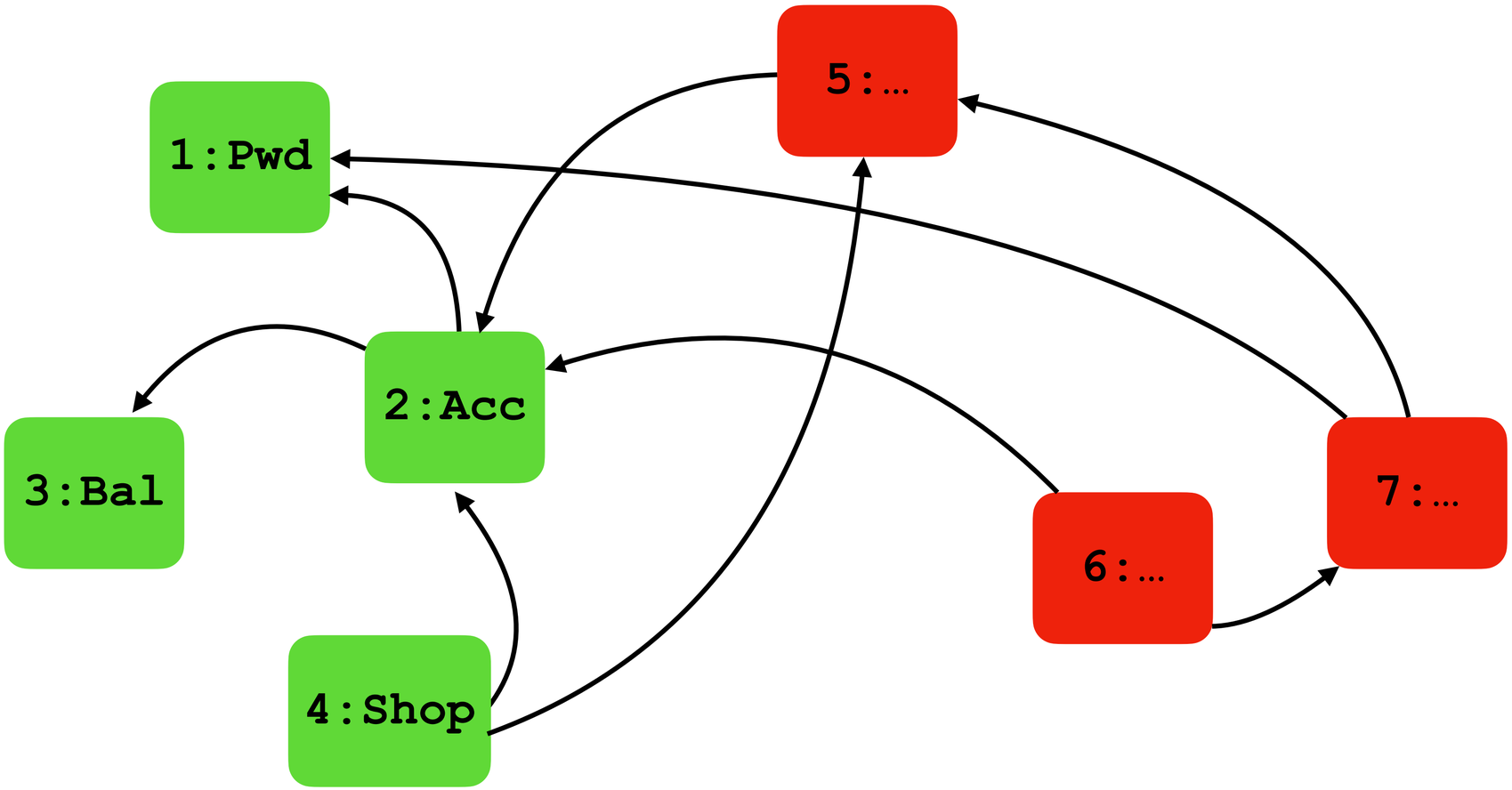
S2: without access to password no decrease in balance

Our remit:

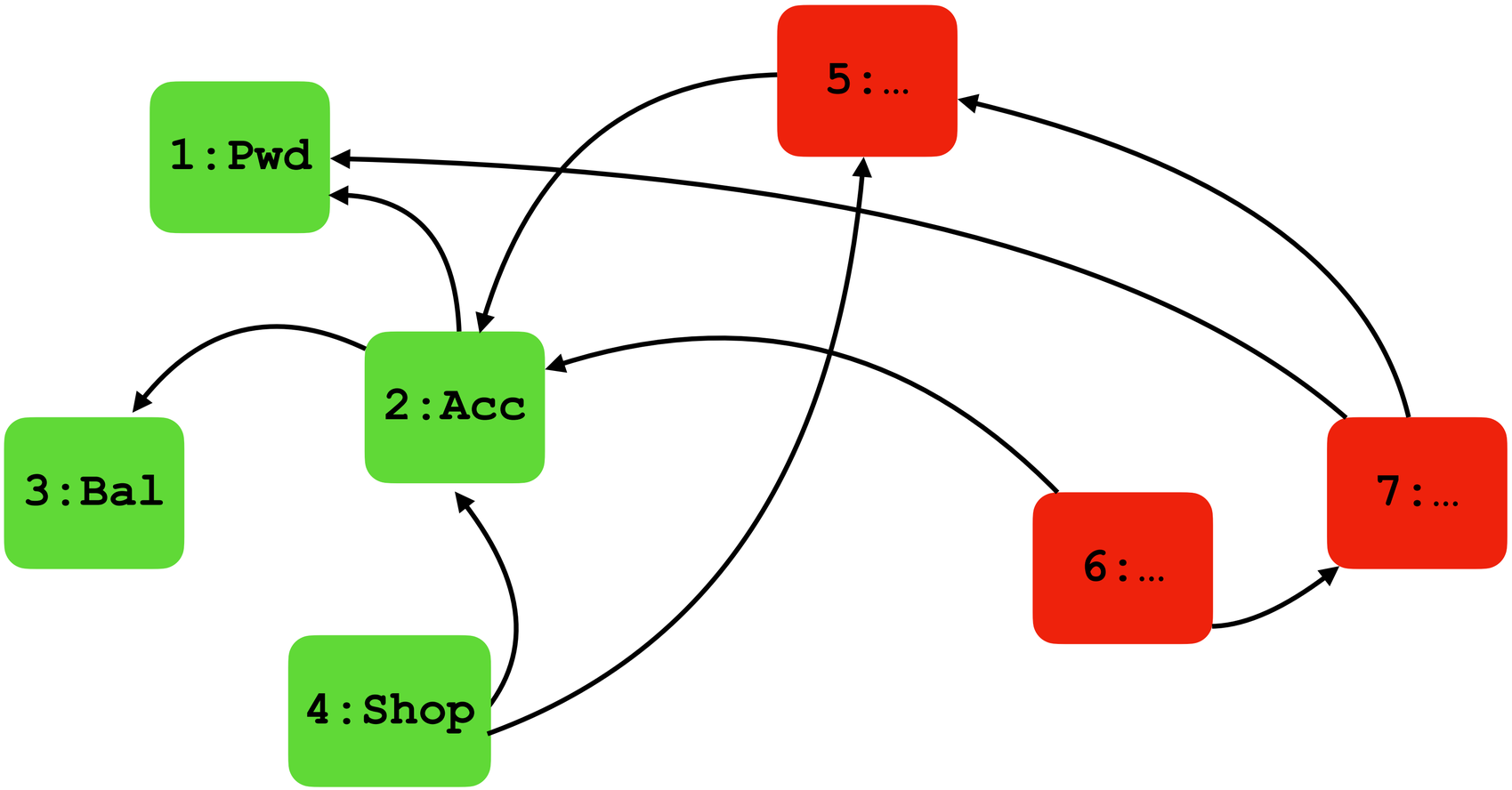
R3: Prove calls from internal to external object.



A call from 4 to 6 *might* reduce 2.balance


5 : …
1 : Pwd
2 : Acc
3 : Bal
7 : …
6 : …
4 : Shop

***Not* our remit:** Forbid external access to capabilty.



The Account example in Code

Three Modules

.. assuming all methods are public, and fields are private

```
module M_good
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(dest:Account, pwd':Password, amt:int) -> void
      if this.pwd==pwd'
        this.blnce-=amt
        dest.blnce+=amt
     public method set(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
```

Three Modules

.. assuming all methods are public, and fields are private

```
module M_good
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(dest:Account, pwd':Password, amt:int) -> void
      if this.pwd==pwd'
        this.blnce-=amt
        dest.blnce+=amt
     public method set(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
```

```
module M_bad
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..) ...
      ... as earlier ...
    public method set(pwd': Password)
      this.pwd=pwd'
```

Three Modules

.. assuming all methods are public, and fields are private

```
module M_good
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(dest:Account, pwd':Password, amt:int) -> void
      if this.pwd==pwd'
        this.blnce-=amt
        dest.blnce+=amt
     public method set(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
```

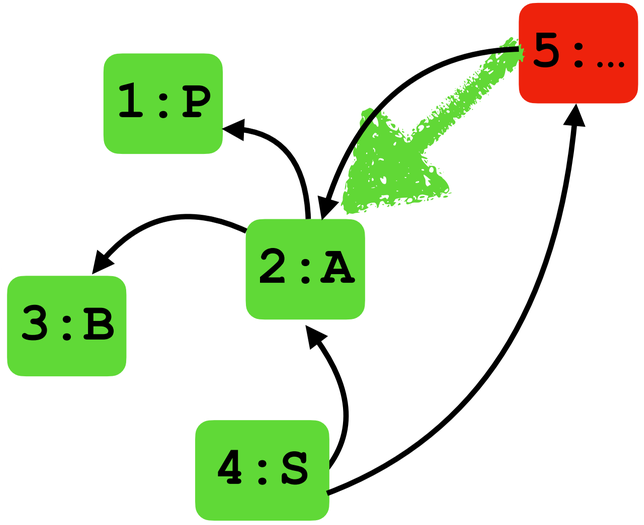
```
module M_bad
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..) ...
      ... as earlier ...
    public method set(pwd': Password)
      this.pwd=pwd'
```

```
module M_fine
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..)
      ... as earlier ...
    public method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mgood
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(dest:Account, pwd':Password, amt:int) -> void
      if this.pwd==pwd'
        this.blnce-=amt
        dest.blnce+=amt
     public method set(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
```




```
module Mbad
  class Password

    class Account
      field blnce:int
      field pwd: Password
      public method transfer(..) ...
        ... as earlier ...
      public method set(pwd': Password)
        this.pwd=pwd'
```

```
module Mfine
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..)
      ... as earlier ...
    public method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
```

Three Modules

.. assuming all methods are public, and fields are private

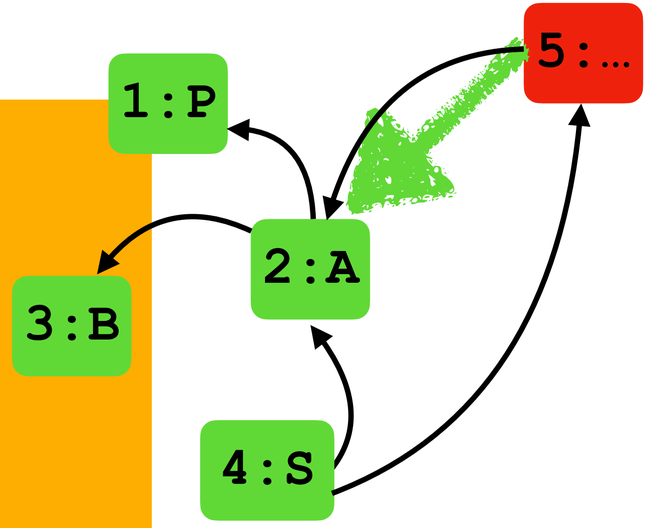
```
module Mgood
  class Password

  class Account
    field blnce:int
    field pwd: Passwor
    public method trans
      if this.pwd==pwd
        this.blnce-=amt
        dest.blnce+=amt
     public method set
      if this.pwd==nul
        this.pwd=pwd'
```

Remit_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{fine} \models S$



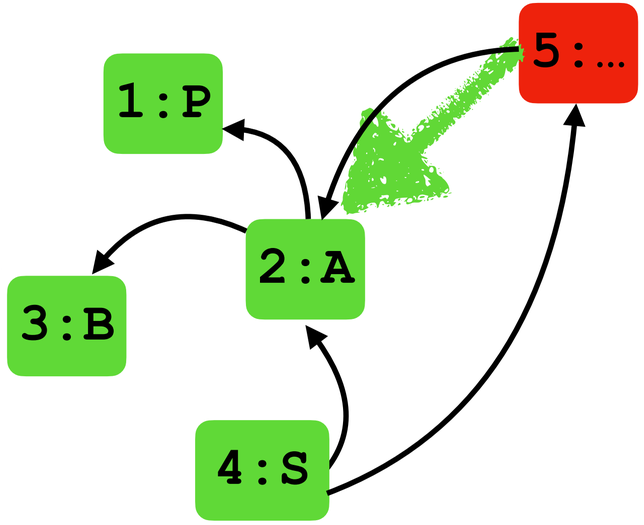

```
module Mbad
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..) ...
      ... as earlier ...
    public method set(pwd': Password)
      this.pwd=pwd'
```

```
module Mfine
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..)
      ... as earlier ...
    public method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mgood
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(dest:Account, pwd':Password, amt:int) -> void
      if this.pwd==pwd'
        this.blnce-=amt
        dest.blnce+=amt
     public method set(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
```




```
module Mbad
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..) ...
      ... as earlier ...
    public method set(pwd': Password)
      this.pwd=pwd'
```

```
module Mfine
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..)
      ... as earlier ...
    public method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
```

Three Modules

.. assuming all methods are public, and fields are private

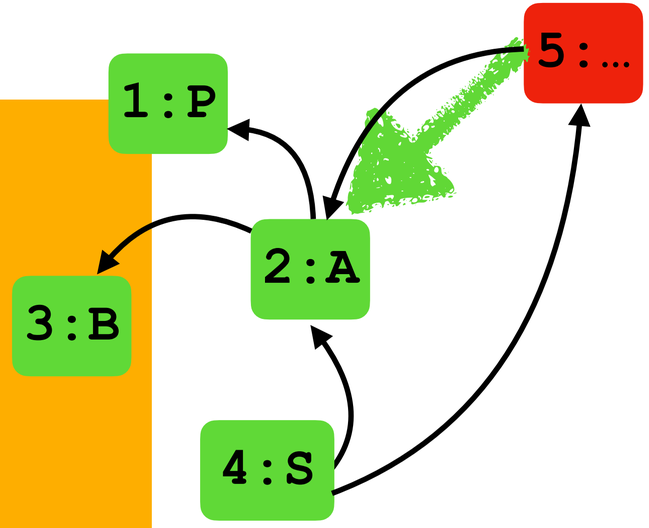
```
module Mgood
  class Password

  class Account
    field blnce:int
    field pwd: Passwor
    public method trans
      if this.pwd==pwd
        this.blnce-=amt
        dest.blnce+=amt
     public method set
      if this.pwd==nul
        this.pwd=pwd'
```

Remit_2: A module spec S, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{fine} \vdash S$



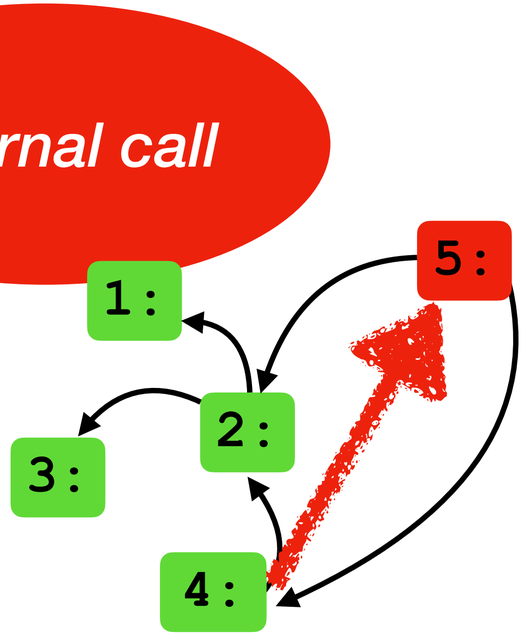

```
module Mbad
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..) ...
      ... as earlier ...
    public method set(pwd': Password)
      this.pwd=pwd'
```

```
module Mfine
  class Password

  class Account
    field blnce:int
    field pwd: Password
    public method transfer(..)
      ... as earlier ...
    public method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
```

```
1  class Shop
2    field accnt :Account, invntry: Inventory, clients: [external]
3
4    public method buy(buyer: external, anItem: Item) -> void
5      int price = anItem.price
6      int oldBlnce = this.accnt.blnce
7      buyer.payMe(this.accnt, price)
8      if (this.accnt.blnce == oldBlnce+price)
9         this.send(buyer,anItem)
10     else
11        buyer.tell("you have not paid me")
12     private method send(buyer: external anItem: Item) -> void
13        ...
```

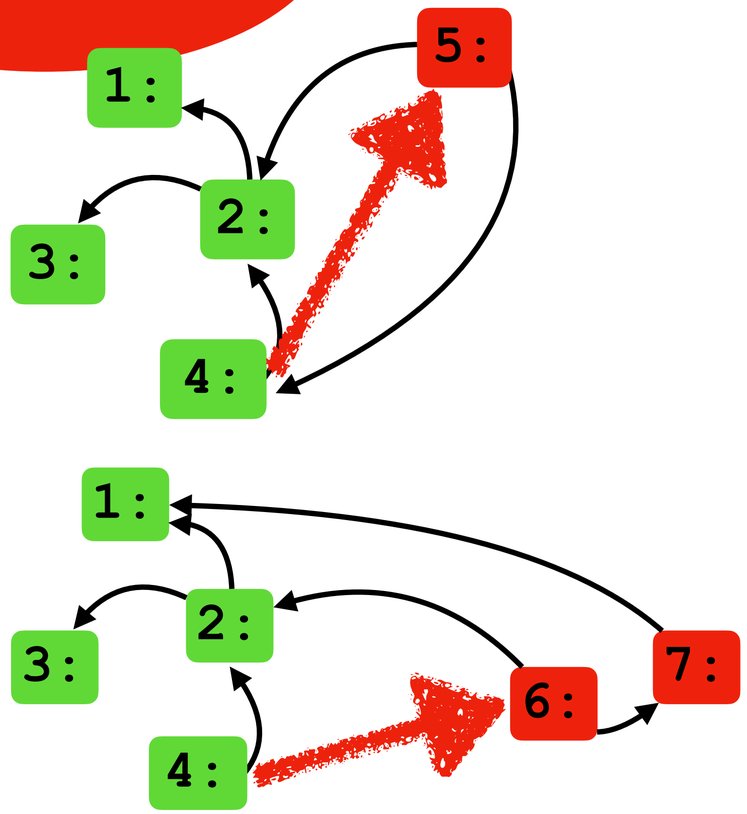
```
1  class Shop
2    field accnt :Account, invntry: Inventory, clients: [
3
4    public method buy(buyer: external, anItem: Item)
5      int price = anItem.price
6      int oldBlnce = this.accnt.blnce
7      buyer.payMe(this.accnt, price)
8      if (this.accnt.blnce == oldBlnce+price)
9         this.send(buyer,anItem)
10     else
11        buyer.tell("you have not paid me")
12     private method send(buyer: external anItem: Item) -> void
13        ...
```

```
1  class Shop
2    field accnt :Account, invntry: Inventory, clients: [
3
4    public method buy(buyer: external, anItem: Item)
5      int price = anItem.price
6      int oldBlnce = this.accnt.blnce
7      buyer.payMe(this.accnt, price)
8      if (this.accnt.blnce == oldBlnce+price)
9         this.send(buyer,anItem)
10     else
11        buyer.tell("you have not paid me")
12     private method send(buyer: external anItem: Item) -> void
13        ...
```



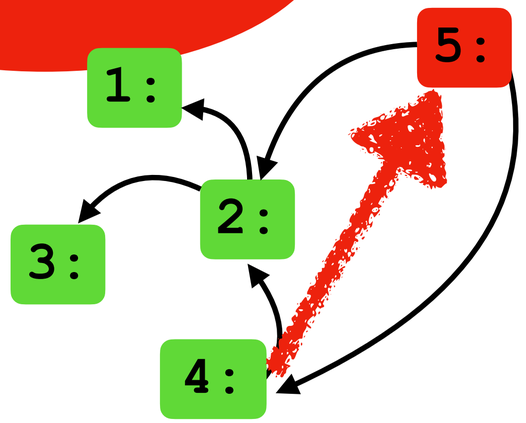
```
1  class Shop
2    field accnt :Account, invntry: Inventory, clients: [
3
4    public method buy(buyer: external, anItem: Item)
5      int price = anItem.price
6      int oldBlnce = this.accnt.blnce
7      buyer.payMe(this.accnt, price)
8      if (this.accnt.blnce == oldBlnce+price)
9         this.send(buyer,anItem)
10     else
11        buyer.tell("you have not paid me")
12     private method send(buyer: external anItem: Item) -> void
13        ...
```

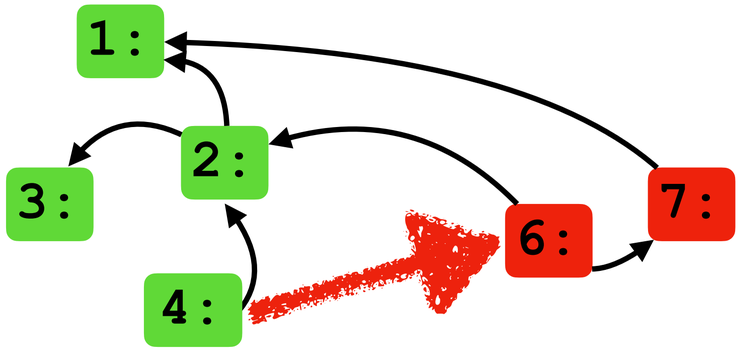
External call



```
1  class Shop
2    field accnt :Account, invntry: Inventory, clients: [
3
4    public method buy(buyer: external, anItem: Item)
5      int price = anItem.price
6      int oldBlnce = this.accnt.blnce
7      buyer.payMe(this.accnt, price)
8      if (this.accnt.blnce == oldBlnce+price)
9         this.send(buyer,anItem)
10     else
11        buyer.tell("you have not paid me")
12     private method send(buyer: external anItem: Item) -> void
13        ...
```

External call

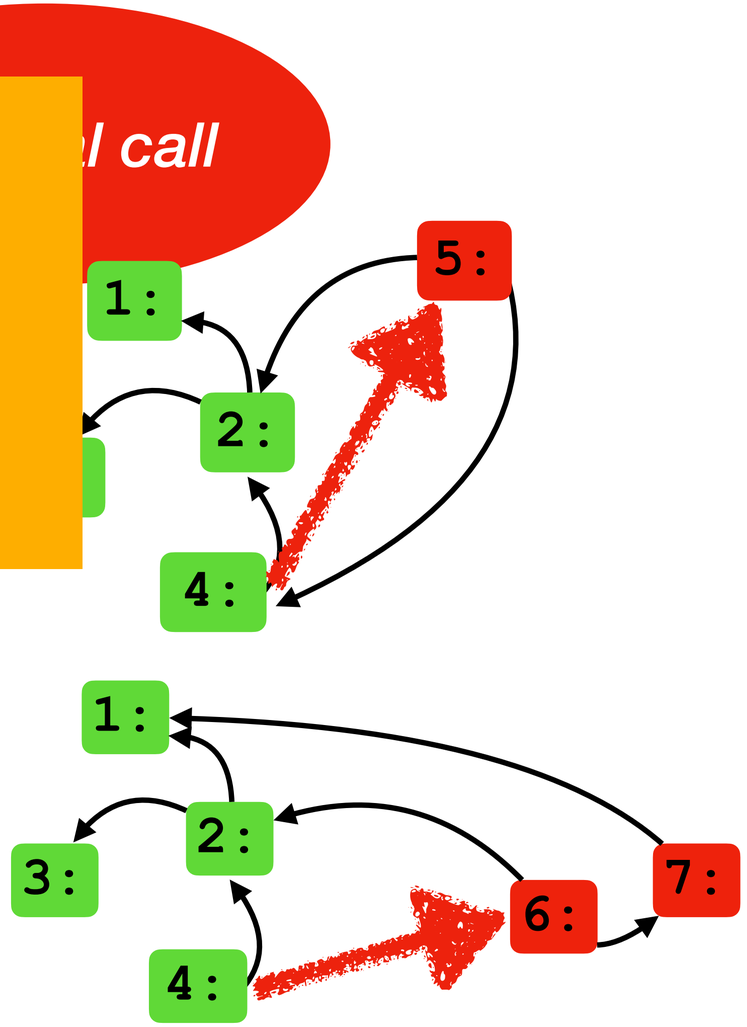




If `Account` comes from a “good” module,
and `buyer` has no unprotected access to `4.accnt.pwd`,
then
`4.accnt.blnce` will not decrease `4.accnt.blnce`,

```
1  class Shop
2    field accnt :A
3
4    public method
5      int price =
6      int oldBlnce
7      buyer.payMe(
8      if (this.acc
9         this.send
10     else
11        buyer.tell("you have not paid me")
12     private method send(buyer: external anItem: Item) -> void
13        ...
```

Remit_3: An inference system, such that we can prove **external** calls.



If `Account` comes from a “good” module,
and `buyer` has no unprotected access to `4.accnt.pwd`,
then
`4.accnt.blnce` will not decrease `4.accnt.blnce`,

Remit_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

Remit_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

Remit_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

So: “The password enables withdrawal from the account”?

Remit_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

Or: "Without the password call of withdraw will fail"?

Remit_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

~~***Or***: "Without the password call of withdraw will fail"?~~

Or: "Without eventual access to password no reduction of the balance of the account"?

Remit_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

~~***Or***: "Without the password call of withdraw will fail"?~~

Or: "Without eventual access to password no reduction of the balance of the account"?

So: $\forall$ a:Accnt,b:Num. { "without eventual external access to" a.pwd $\wedge$ a.balance$\geq$b}

Remit_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

R1: Specify that external access to capability necessary for effect.

~~***So***: "The password enables withdrawal from the account"?~~

~~***Or***: "Without the password call of withdraw will fail"?~~

Or: "Without eventual access to password no reduction of the balance of the account"?

So: $\forall$ a:Accnt,b:Num. { "without eventual external access to" a.pwd $\wedge$ a.balance$\geq$b}

Remit_1: A module spec S, such that …

Remit_1: A module spec S, such that …

In general: $\forall$ x1:C1,x2:C2…{ A }

Remit_1: A module spec S, such that …

In general: ∀ x1:C1,x2:C2…{ A }

Remit_1_a : Meaning of “without eventual external access to”

Remit_1: A module spec S, such that …

In general: ∀ x1:C1,x2:C2…{ A }

Remit_1_a : Meaning of “without eventual external access to”

Remit_1_b : Meaning of ∀ x1:C1,x2:C2… { A }

Remit_1: A module spec S, such that …

o: ∀ a:Accnt. ∀ b:Num. { “without eventual external access to” a.pwd ∧ a.balance≧b}

In general: ∀ x1:C1,x2:C2…{ A }

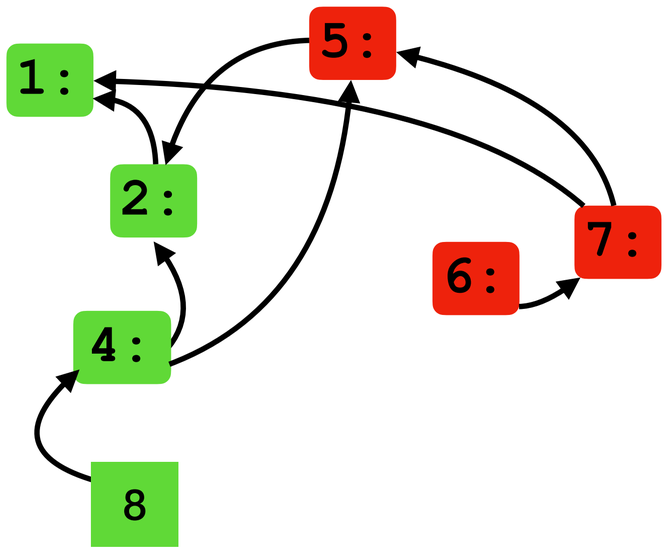
Remit_1_a : Meaning of “without eventual external access to”

Remit_1_b : Meaning of ∀ x1:C1,x2:C2… { A }

Remit_1_a : Meaning of “without eventual external access to”

… is about an external object eventually obtainining access.

Def: 《o》 ≜ ∀ o’ [o’ external and reachable from top of stack frame ⇒ 《o》↛o’]

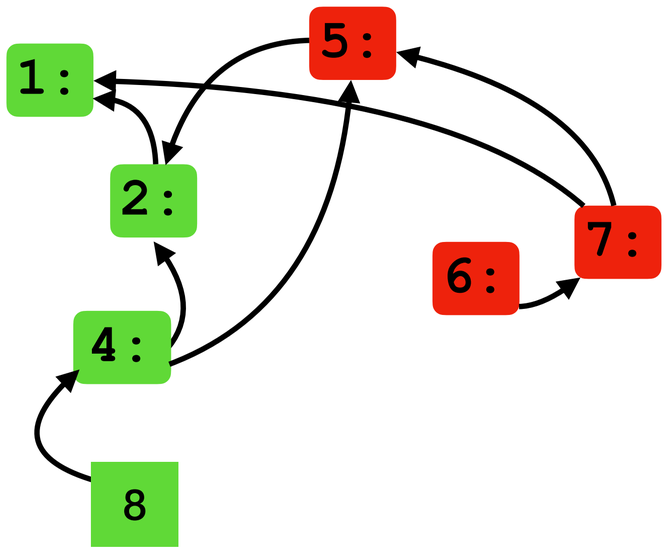


Remit_1_a : Meaning of “without eventual external access to”

… is about an external object eventually obtainining access.

Def: 《o》⊣o’ ≜ the penultimate object on any path from o’ to o is internal.

Def: 《o》 ≜ ∀ o’ [o’ external and reachable from top of stack frame ⇒ 《o》⊣o’]



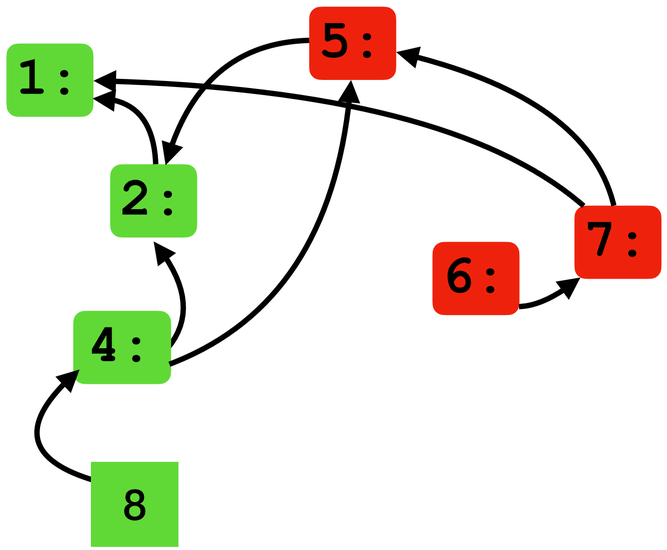
Remit_1_a : Meaning of “without eventual external access to”

… is about an external object eventually obtainining access.

Def: 《o》⊣o’ ≜ the penultimate object on any path from o’ to o is internal.

Def: 《o》 ≜ ∀ o’ [o’ external and reachable from top of stack frame ⇒ 《o》⊣o’]

For example:



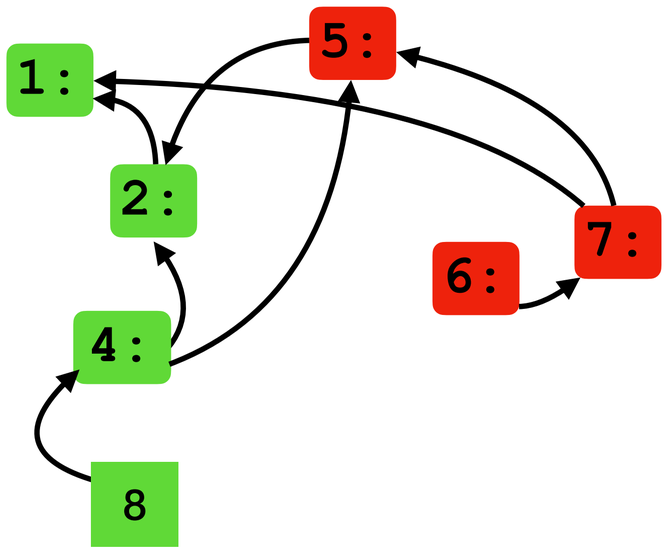
Remit_1_a : Meaning of “without eventual external access to”

… is about an external object eventually obtainining access.

Def: $\langle\!\langle o \rangle\!\rangle \dashv o'$ $\triangleq$ the penultimate object on any path from o’ to o is internal.

Def: $\langle\!\langle o \rangle\!\rangle$ $\triangleq$ $\forall$ o’ [o’ external and reachable from top of stack frame $\Rightarrow$ $\langle\!\langle o \rangle\!\rangle \dashv o'$]

For example:



$\ldots \models \langle\!\langle 1 \rangle\!\rangle \dashv 5$

$\ldots \not\models \langle\!\langle 1 \rangle\!\rangle \dashv 6$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 4$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 8$

Remit_1_a : Meaning of “without eventual external access to”

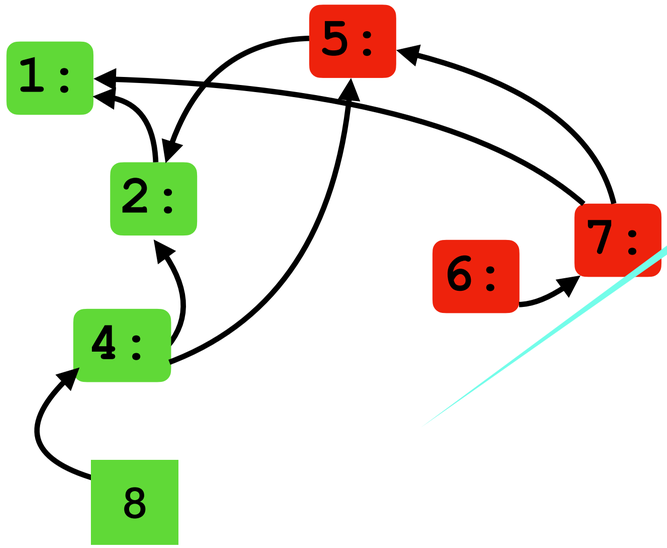
… is about an external object eventually obtainining access.

Def: 《o》+o’ ≜ the penultimate object on any path from o’ to o is internal.

Def: 《o》 ≜ ∀ o’ [o’ external frame ⇒ 《o》+o’]

Protection is “relative”
to an object

For example:



… ⊨ 《1》+ 5

… ⊭ 《1》+ 6

… ⊭ 《2》+ 4

… ⊭ 《2》+ 8

Remit_1_a : Meaning of "without eventual external access to"

Def: 《o》⫞o' ≜ the penultimate object on any path from o' to o is internal.

```
1  class Shop
2    field accnt :Account, invntry: Inventory, clients: [external]
3
4    public method buy(buyer: external, anItem: Item) -> void
5      int price = anItem.price
6      int oldBlnce = this.accnt.blnce
7      buyer.payMe(this.accnt, price)
8      if (this.accnt.blnce == oldBlnce+price)
9         this.send(buyer,anItem)
10     else
11        buyer.tell("you have not paid me")
12     private method send(buyer: external anItem: Item) -> void
13        ...
```

Remit_1_a : Meaning of "without eventual external access to"

Def: 《o》⊰o' ≜ the penultimate object on any path from o' to o is internal.

PRE: 《this.accnt.pwd》⊰buyer ∧ this.accnt.blnce == b

POST: this.accnt.blnce >= b

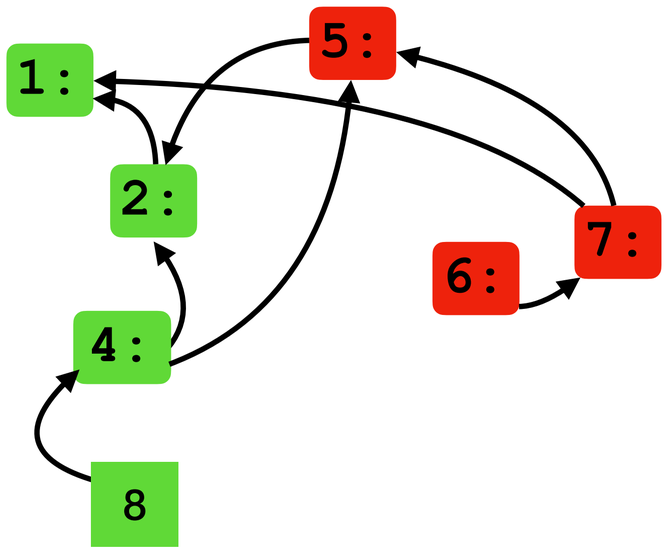
```
1  class Shop
2    field accnt :Account, invntry: Inventory, clients: [external]
3
4    public method buy(buyer: external, anItem: Item) -> void
5      int price = anItem.price
6      int oldBlnce = this.accnt.blnce
7      buyer.payMe(this.accnt, price)
8      if (this.accnt.blnce == oldBlnce+price)
9         this.send(buyer,anItem)
10     else
11        buyer.tell("you have not paid me")
12     private method send(buyer: external anItem: Item) -> void
13        ...
```

Remit_1_a : Meaning of “without eventual external access to”

… is about an external object eventually obtainining access.

Def: 《o》⊣o’ ≜ the penultimate object on any path from o’ to o is internal.

For example:



… ⊨ 《1》⊣ 5

… ⊭ 《1》⊣ 6

… ⊭ 《2》⊣ 4

… ⊭ 《2》⊣ 8

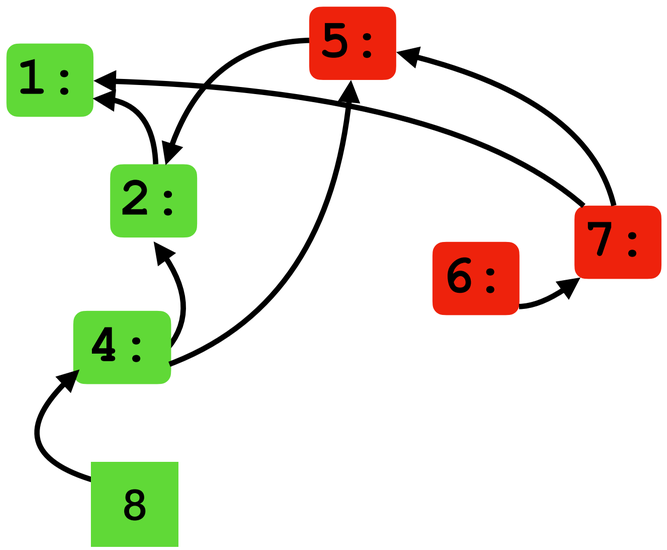
Remit_1_a : Meaning of “without eventual external access to”

… is about an external object eventually obtainining access.

Def: $\langle\!\langle o \rangle\!\rangle \dashv o'$ $\triangleq$ the penultimate object on any path from o’ to o is internal.

Def: $\langle\!\langle o \rangle\!\rangle$ $\triangleq$ $\forall$ o’ [o’ external and reachable from top of stack frame $\Rightarrow$ $\langle\!\langle o \rangle\!\rangle \dashv o'$]

For example:



$\ldots \models \langle\!\langle 1 \rangle\!\rangle \dashv 5$

$\ldots \not\models \langle\!\langle 1 \rangle\!\rangle \dashv 6$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 4$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 8$

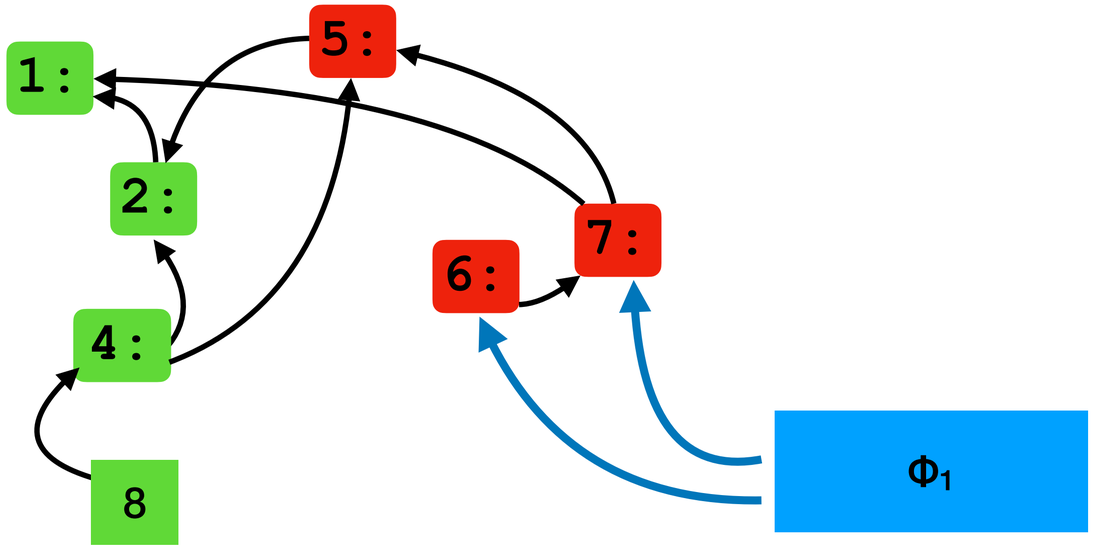
Remit_1_a : Meaning of “without eventual external access to”

… is about an external object eventually obtainining access.

Def: $\langle\!\langle o \rangle\!\rangle \dashv o'$ $\triangleq$ the penultimate object on any path from o’ to o is internal.

Def: $\langle\!\langle o \rangle\!\rangle$ $\triangleq$ $\forall$ o’ [o’ external and reachable from top of stack frame $\Rightarrow$ $\langle\!\langle o \rangle\!\rangle \dashv o'$]

For example:



$\ldots \models \langle\!\langle 1 \rangle\!\rangle \dashv 5$

$\ldots \not\models \langle\!\langle 1 \rangle\!\rangle \dashv 6$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 4$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 8$

$\Phi_1 \ldots \not\models \langle\!\langle 1 \rangle\!\rangle$

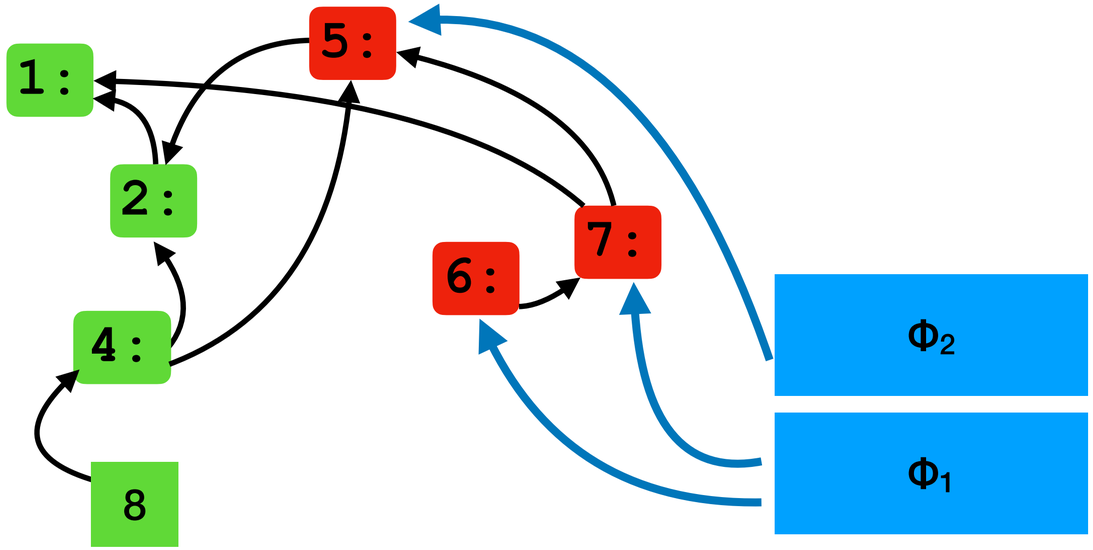
Remit_1_a : Meaning of “without eventual external access to”

… is about an external object eventually obtainining access.

Def: $\langle\!\langle o \rangle\!\rangle \dashv o'$ $\triangleq$ the penultimate object on any path from o’ to o is internal.

Def: $\langle\!\langle o \rangle\!\rangle$ $\triangleq$ $\forall$ o’ [o’ external and reachable from top of stack frame $\Rightarrow$ $\langle\!\langle o \rangle\!\rangle \dashv o'$]

For example:



$\ldots \models \langle\!\langle 1 \rangle\!\rangle \dashv 5$

$\ldots \not\models \langle\!\langle 1 \rangle\!\rangle \dashv 6$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 4$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 8$

$\Phi_1 \ldots \not\models \langle\!\langle 1 \rangle\!\rangle$

$\Phi_1 \Phi_2 \ldots \models \langle\!\langle 1 \rangle\!\rangle$

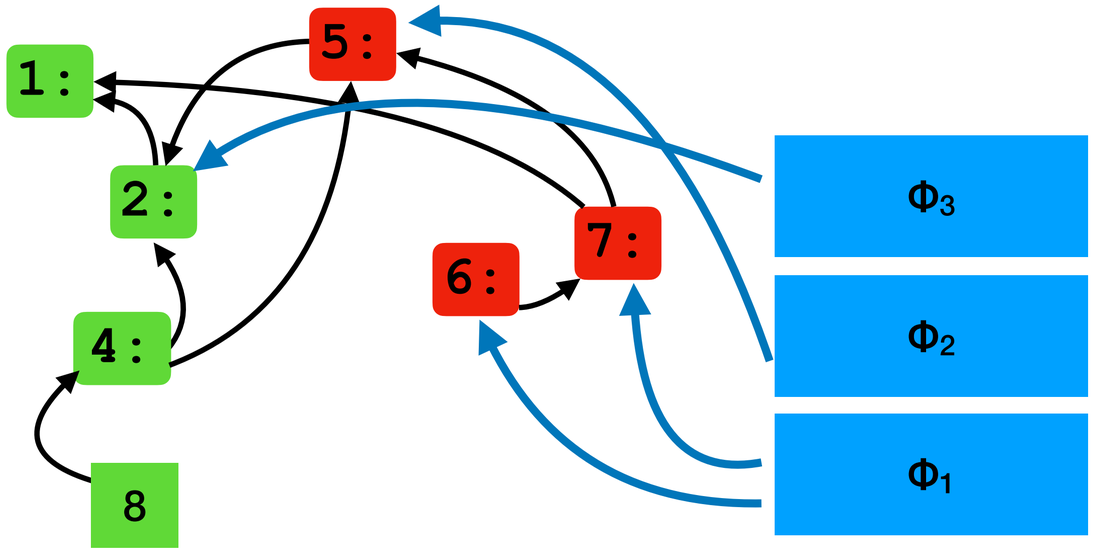
Remit_1_a : Meaning of “without eventual external access to”

… is about an external object eventually obtainining access.

Def: $\langle\!\langle o \rangle\!\rangle \dashv o'$ $\triangleq$ the penultimate object on any path from o’ to o is internal.

Def: $\langle\!\langle o \rangle\!\rangle$ $\triangleq$ $\forall$ o’ [o’ external and reachable from top of stack frame $\Rightarrow$ $\langle\!\langle o \rangle\!\rangle \dashv o'$]

For example:



$\ldots \models \langle\!\langle 1 \rangle\!\rangle \dashv 5$

$\ldots \not\models \langle\!\langle 1 \rangle\!\rangle \dashv 6$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 4$

$\ldots \not\models \langle\!\langle 2 \rangle\!\rangle \dashv 8$

$\Phi_1 \ldots \not\models \langle\!\langle 1 \rangle\!\rangle$

$\Phi_1\Phi_2 \ldots \models \langle\!\langle 1 \rangle\!\rangle$

$\Phi_1\Phi_2 \ldots \not\models \langle\!\langle 2 \rangle\!\rangle$

$\Phi_1\Phi_2\Phi_3 \ldots \models \langle\!\langle 1 \rangle\!\rangle$

$\Phi_1\Phi_2\Phi_3 \ldots \models \langle\!\langle 2 \rangle\!\rangle$

Remit_1_a : Meaning of “without eventual external access to”

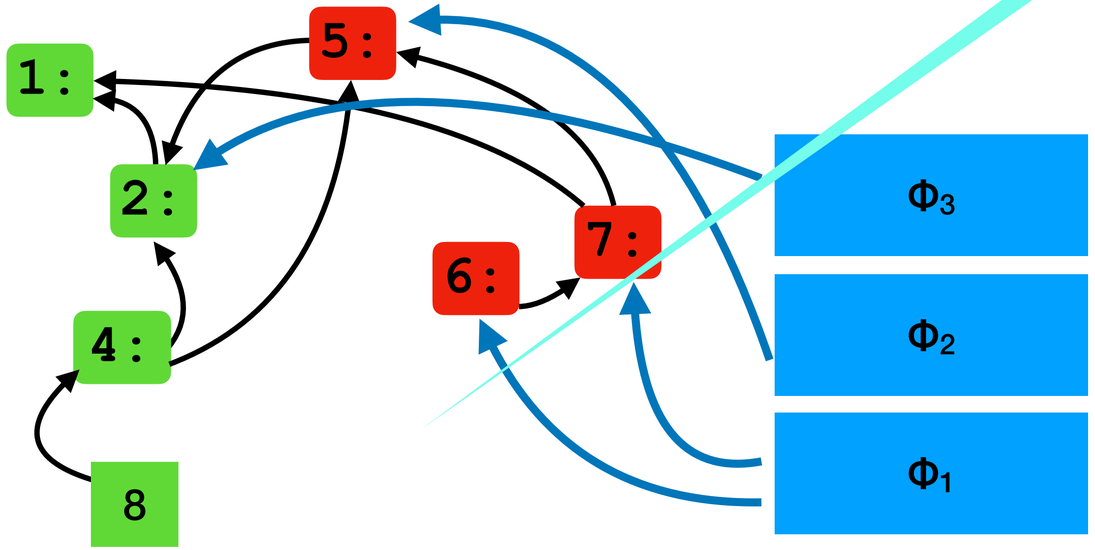
… is about an external object eventually obtainining access.

Def: 《o》⊣o’ ≜ the penultimate object on any path from o’ to o is internal.

Def: 《o》 ≜ ∀ o’ [o’ external frame ⇒ 《o》⊣o’]

Protection is “relative”
to an object,
or a frame

For example:



… ⊨ 《1》⊣ 5

… ⊭ 《1》⊣ 6

… ⊭ 《2》⊣ 4

… ⊭ 《2》⊣ 8

Φ1… ⊭ 《1》

Φ1Φ2… ⊨ 《1》

Φ1Φ2… ⊭ 《2》

Φ1Φ2Φ3… ⊨ 《1》

Φ1Φ2Φ3… ⊨ 《2》

Remit_1_b : Meaning of ∀x1:C1,x2:C2… { A }

Definition

$$M \models \forall x1:C1,\ldots xn:Cn\ \{ A \}$$

$$\triangleq$$

Remit_1_b : Meaning of ∀x1:C1,x2:C2… { A }

Definition

$$M \models \forall x1{:}C1,\ldots xn{:}Cn\ \{ A \}$$

$$\triangleq$$

$$\forall M', \forall \sigma, \sigma', \forall \alpha 1,\ldots \alpha n[$$

$$]$$

Remit_1_b : Meaning of $\forall$x1:C1,x2:C2… { A }

Definition

$$M \models \forall x1:C1,\ldots xn:Cn \{ A \}$$

$$\triangleq$$

$$\forall M', \forall \sigma, \sigma', \forall \alpha 1,\ldots \alpha n[$$

$$M, \sigma \models \texttt{this}: \text{ext} \wedge \alpha 1 :C1,\ldots \alpha n:Cn \wedge A[\alpha 1,\ldots \alpha n / x1,..xn]$$

$$]$$

Remit_1_b : Meaning of $\forall$x1:C1,x2:C2… { A }

Definition

$$M \models \forall x1{:}C1,\ldots xn{:}Cn\ \{ A \}$$

$$\triangleq$$

$$\forall M', \forall \sigma, \sigma', \forall \alpha 1,\ldots \alpha n[$$

$$M, \sigma \models \texttt{this}: \text{ext} \ \wedge\ \alpha 1 : C1,\ldots \alpha n{:}Cn \ \wedge A[\ \alpha 1,\ldots \alpha n \ / \ x1,..xn]$$

$$\wedge$$

$$M'^{*}M, \sigma \rightsquigarrow^{*} \sigma'$$

$$]$$

Remit_1_b : Meaning of $\forall$x1:C1,x2:C2… { A }

Definition

$$M \models \forall x1{:}C1,\ldots xn{:}Cn\ \{ A \}$$

$$\triangleq$$

$$\forall M', \forall \sigma, \sigma', \forall \alpha 1,\ldots \alpha n[$$

$$M, \sigma \models \texttt{this}: \text{ext} \ \wedge\ \alpha 1 : C1,\ldots \alpha n{:}Cn \ \wedge A[\ \alpha 1,\ldots \alpha n\ /\ x1,..xn]$$

$$\wedge$$

$$M'^{*}M, \sigma \rightsquigarrow^{*} \sigma'$$

$$\Rightarrow$$

$$M, \sigma' \models \texttt{this}: \text{ext} \longrightarrow A[\ \alpha 1,\ldots \alpha n\ /\ x1,..xn]$$

$$]$$

Remit_1_b : Meaning of ∀x1:C1,x2:C2… { A }

Definition

Scoped execution

$$M \models \forall x1{:}C1,\ldots xn{:}Cn\ \{ A \}$$

$$\triangleq$$

$$\forall M', \forall \sigma, \sigma', \forall \alpha 1,\ldots \alpha n[$$

$$M, \sigma \models \texttt{this}: \text{ext} \wedge \alpha 1 :C1,\ldots \alpha n{:}Cn \wedge A[\alpha 1,\ldots \alpha n / x1,..xn]$$

$$\wedge$$

$$M'{*}M, \sigma \rightsquigarrow^{*} \sigma'$$

$$\Rightarrow$$

$$M, \sigma' \models \texttt{this}: \text{ext} \longrightarrow A[\alpha 1,\ldots \alpha n / x1,..xn]$$

$$]$$

Remit_1_c : Meaning of $M'^{*}M,\ \sigma \rightsquigarrow^{*} \sigma'$

Definition

$$M'^{*}M,\ \sigma \rightsquigarrow^{*} \sigma' \triangleq$$

$\longrightarrow$

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

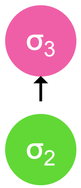
$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \to^* \sigma'$ "while not popping the top frame of σ"

$\longrightarrow$

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

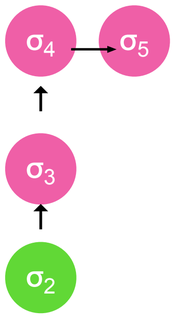
$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ -}>^* \sigma'$ "while not popping the top frame of σ"



Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

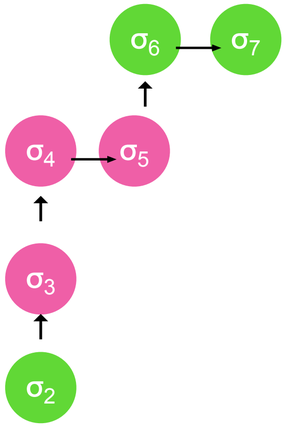
$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ -}>^* \sigma'$ "while not popping the top frame of σ"



Remit_1_c : Meaning of $M'^*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

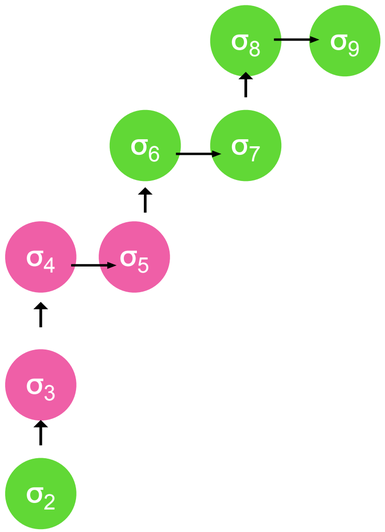
$M'^*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'^*M,\ \sigma \text{ -}>^* \sigma'$ “while not popping the top frame of σ”



Remit_1_c : Meaning of $M'^*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'^*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'^*M,\ \sigma \text{ -}>^* \sigma'$ “while not popping the top frame of σ”

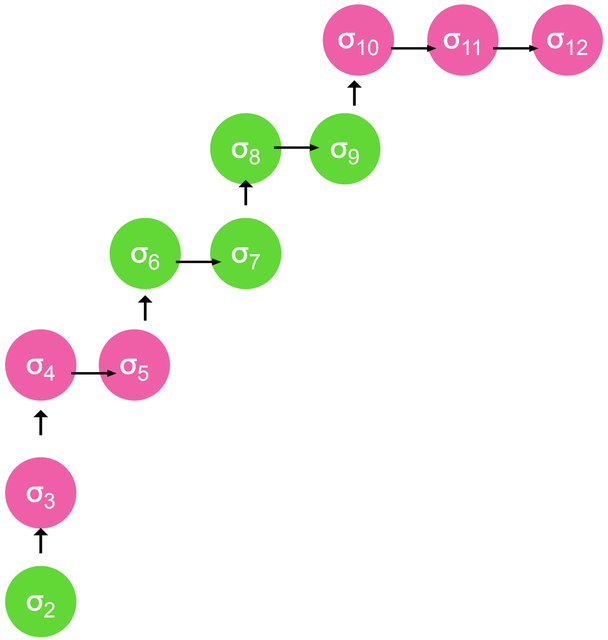


→

Remit_1_c : Meaning of $M'^*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'^*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'^*M,\ \sigma \text{ -}>^* \sigma'$ “while not popping the top frame of σ”

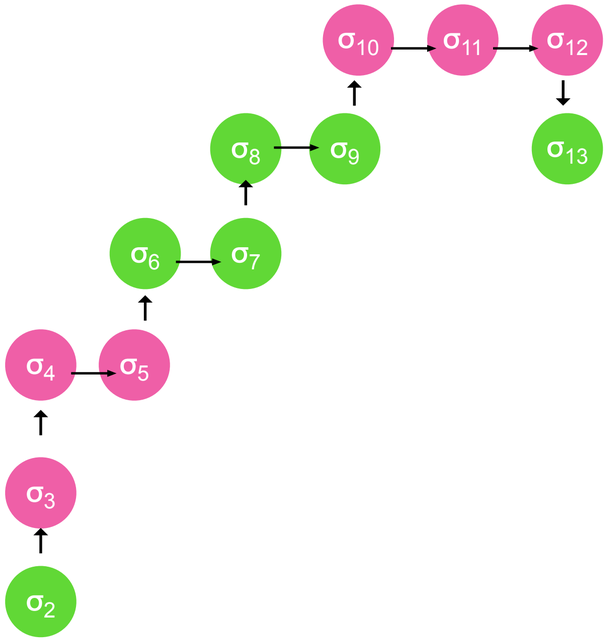


→

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ -}>^* \sigma'$ “while not popping the top frame of σ”

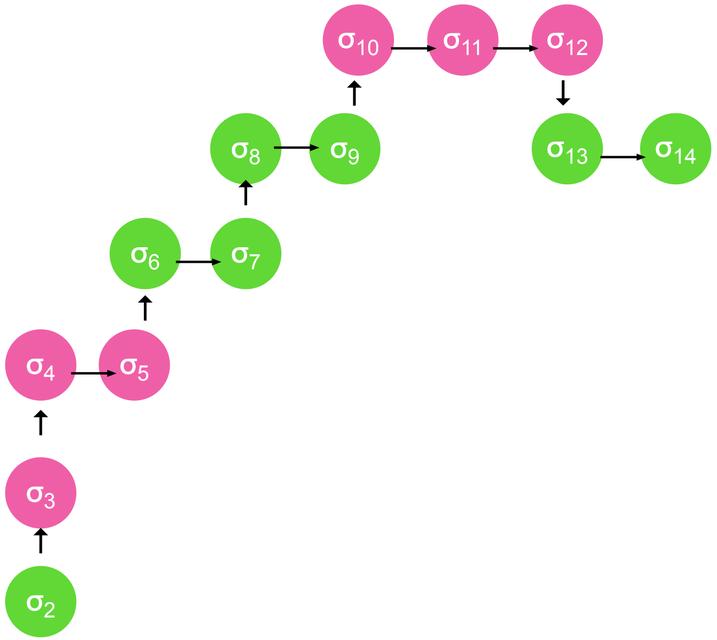


→

Remit_1_c : Meaning of $M'^*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'^*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'^*M,\ \sigma \text{ -}>^* \sigma'$ “while not popping the top frame of σ”

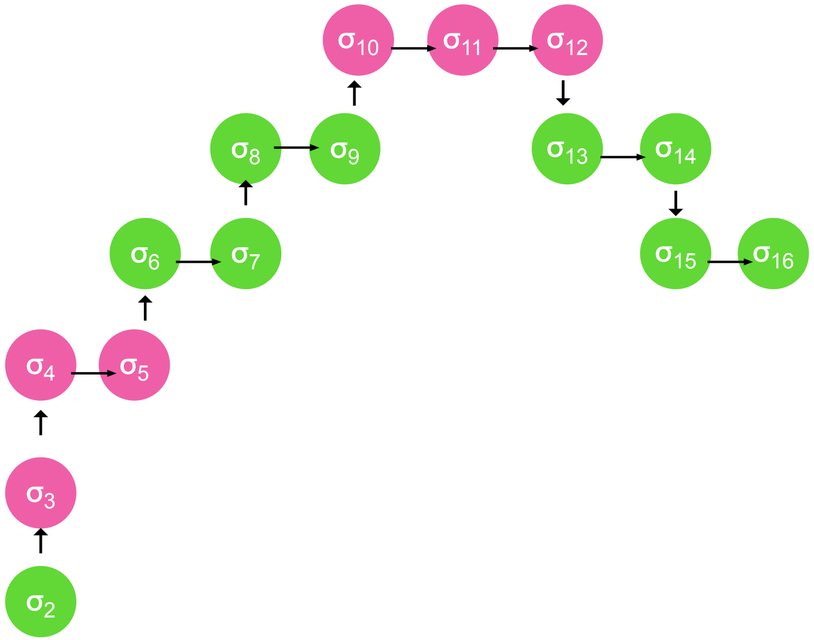


→

Remit_1_c : Meaning of $M'^*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'^*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'^*M,\ \sigma \text{ -}>^* \sigma'$ “while not popping the top frame of σ”

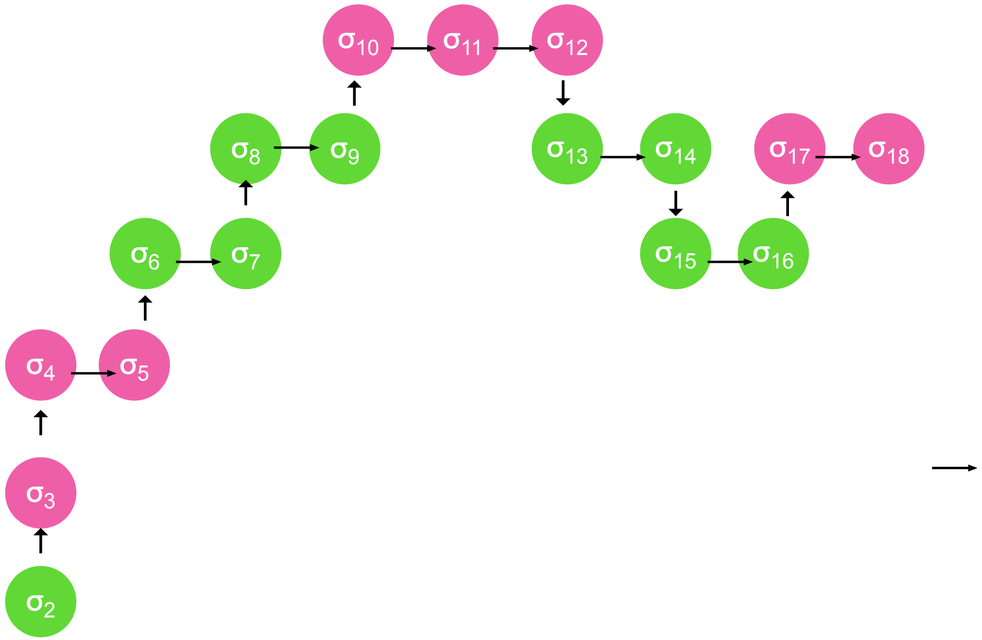


→

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

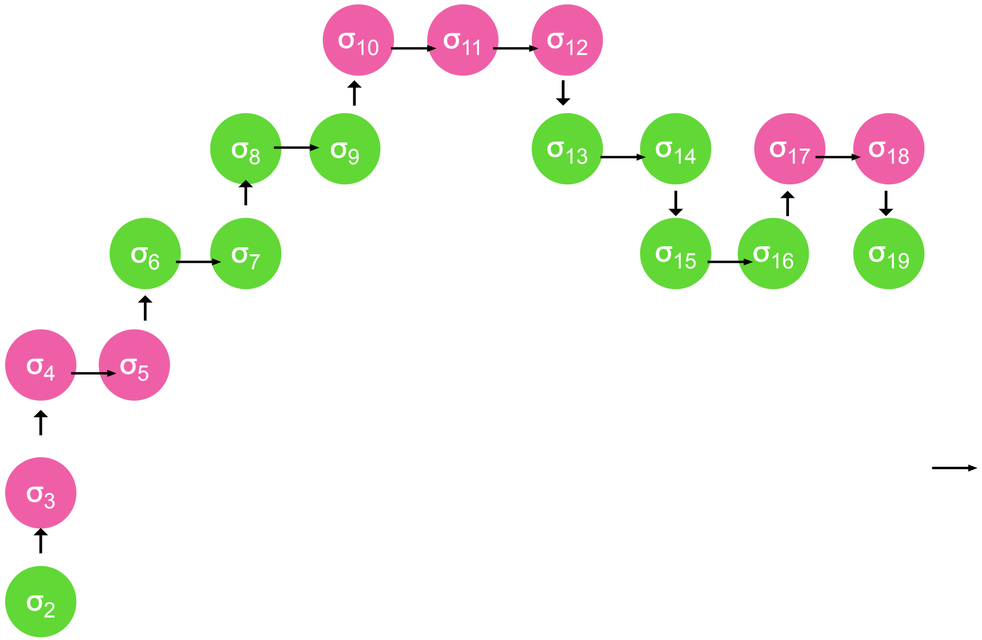
$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ -}>^* \sigma'$ “while not popping the top frame of σ”



Remit_1_c : Meaning of $M'^*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

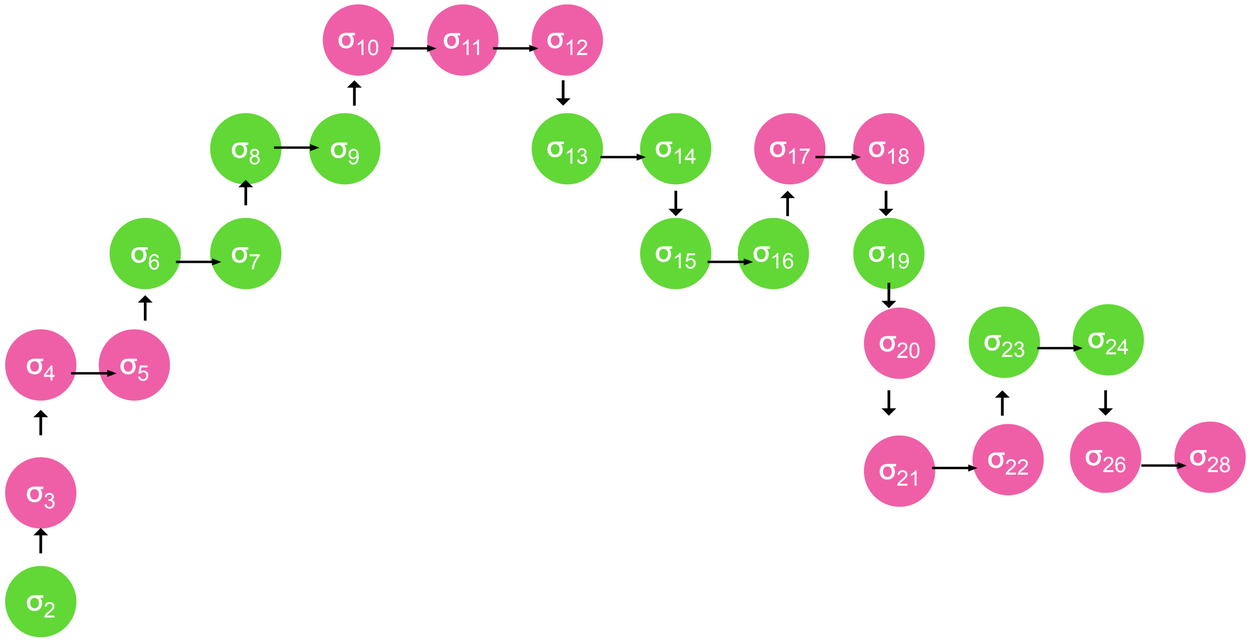
$M'^*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'^*M,\ \sigma \text{ -}>^* \sigma'$ “while not popping the top frame of σ”



Remit_1_c : Meaning of $M'^*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

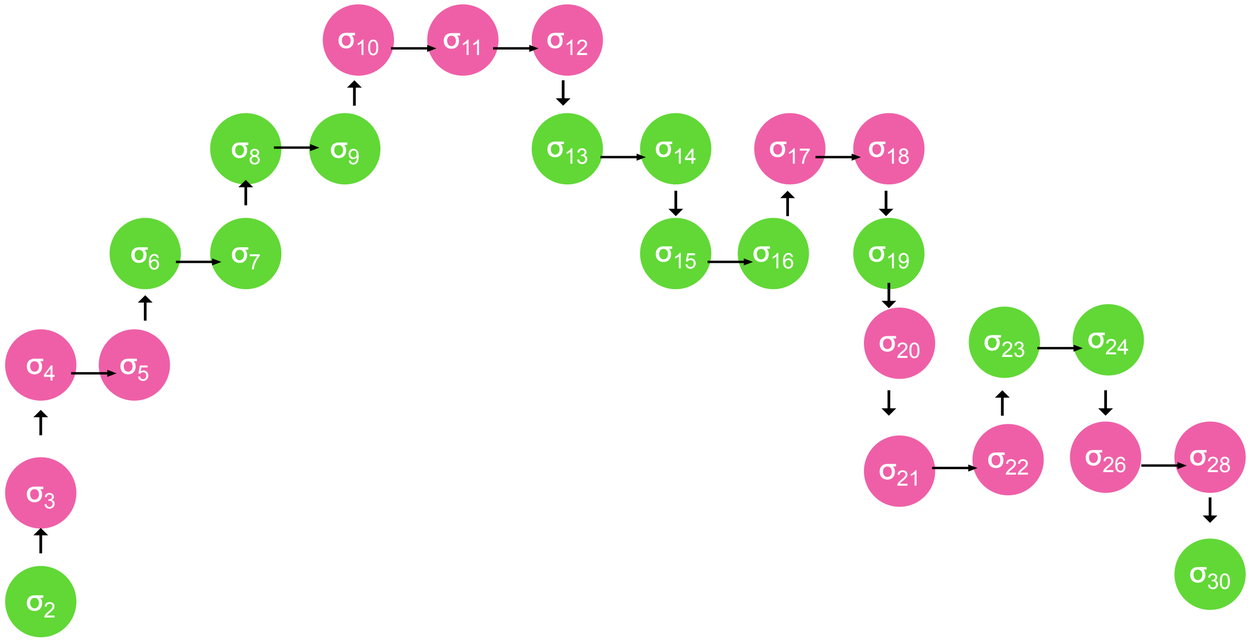
$M'^*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'^*M,\ \sigma \text{ -}>^* \sigma'$ “while not popping the top frame of σ”



Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

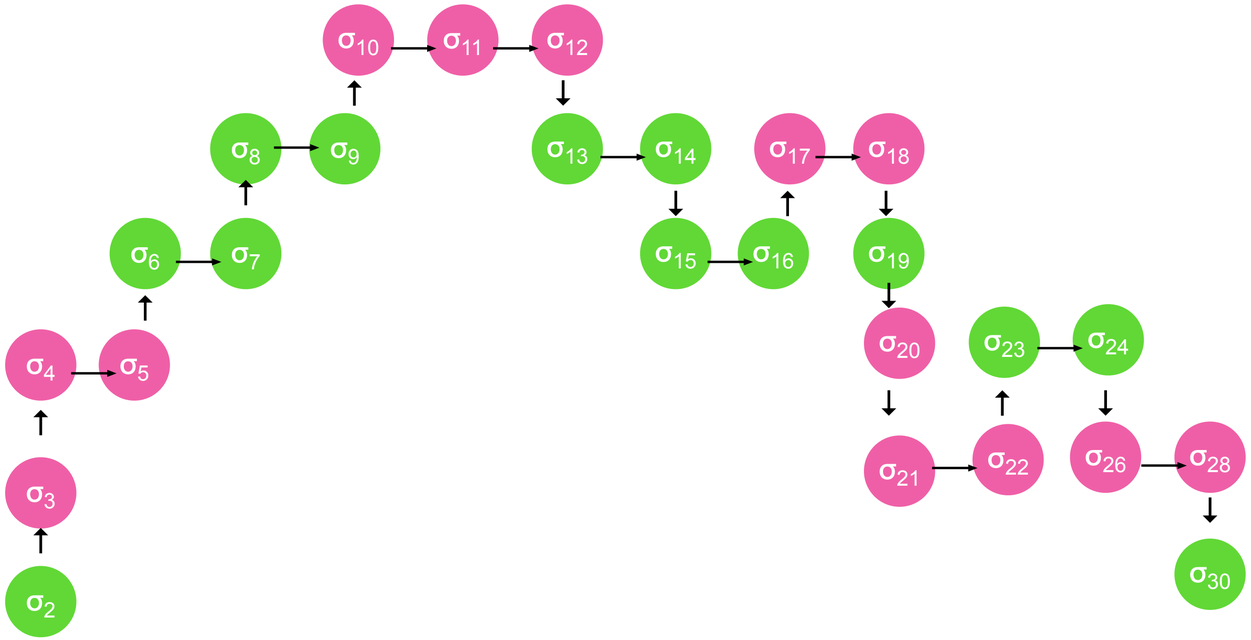
$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ -}>^* \sigma'$ "while not popping the top frame of σ"



Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ ->}^* \sigma'$ "while not popping the top frame of σ"



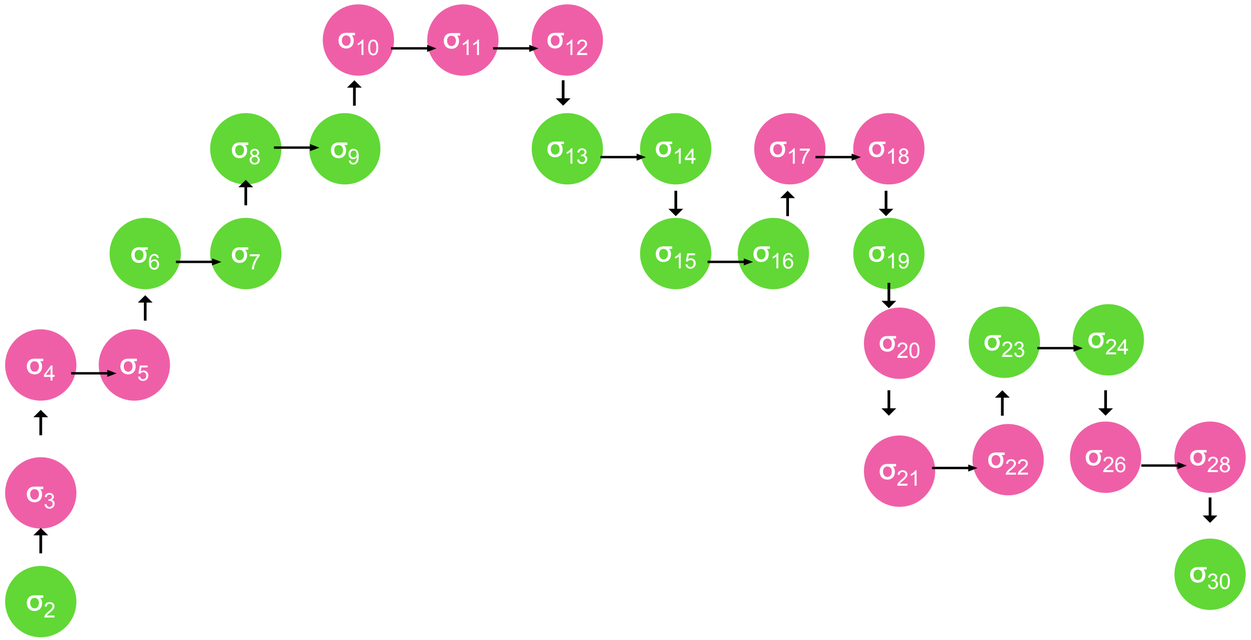
$\sigma_4 \text{ ->}^* \sigma_{18}$

$\sigma_4 \rightsquigarrow^* \sigma_{18}$

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \to^* \sigma'$ "while not popping the top frame of σ"



$\sigma_4 \to^* \sigma_{18}$

$\sigma_4 \rightsquigarrow^* \sigma_{18}$

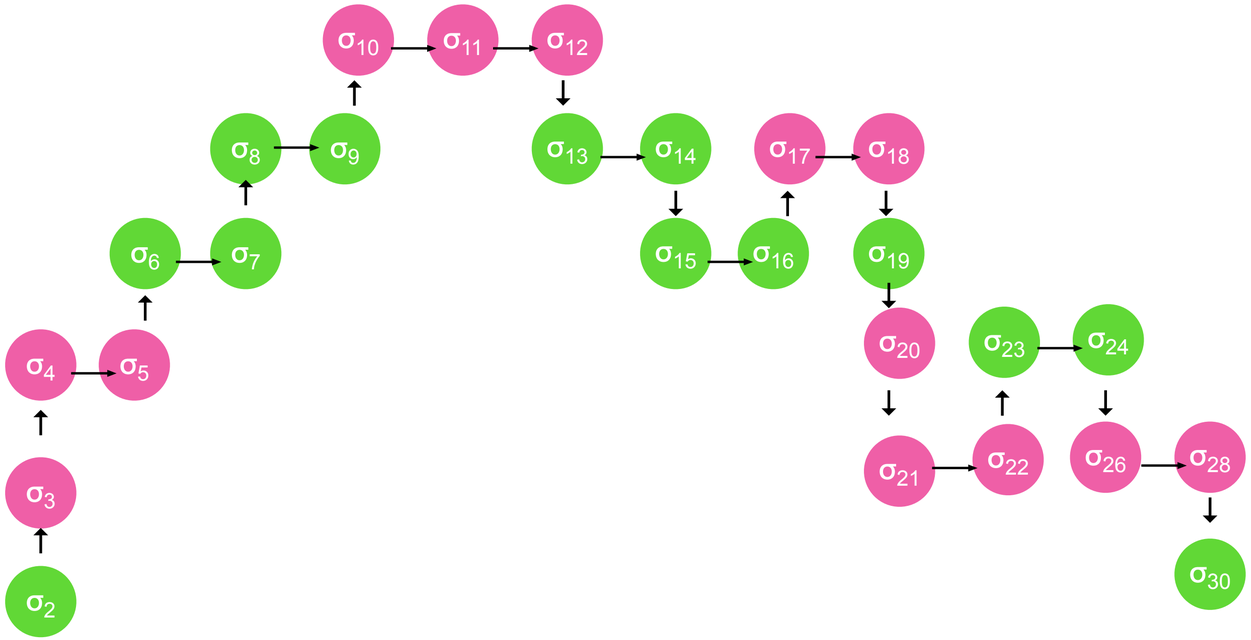
$\sigma_4 \to^* \sigma_{20}$

$\sigma_4 \rightsquigarrow^* \sigma_{20}$

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ -}>^* \sigma'$ "while not popping the top frame of σ"



$\sigma_4 \text{ -}>^* \sigma_{18}$

$\sigma_4 \rightsquigarrow^* \sigma_{18}$

$\sigma_4 \text{ -}>^* \sigma_{20}$

$\sigma_4 \rightsquigarrow^* \sigma_{20}$

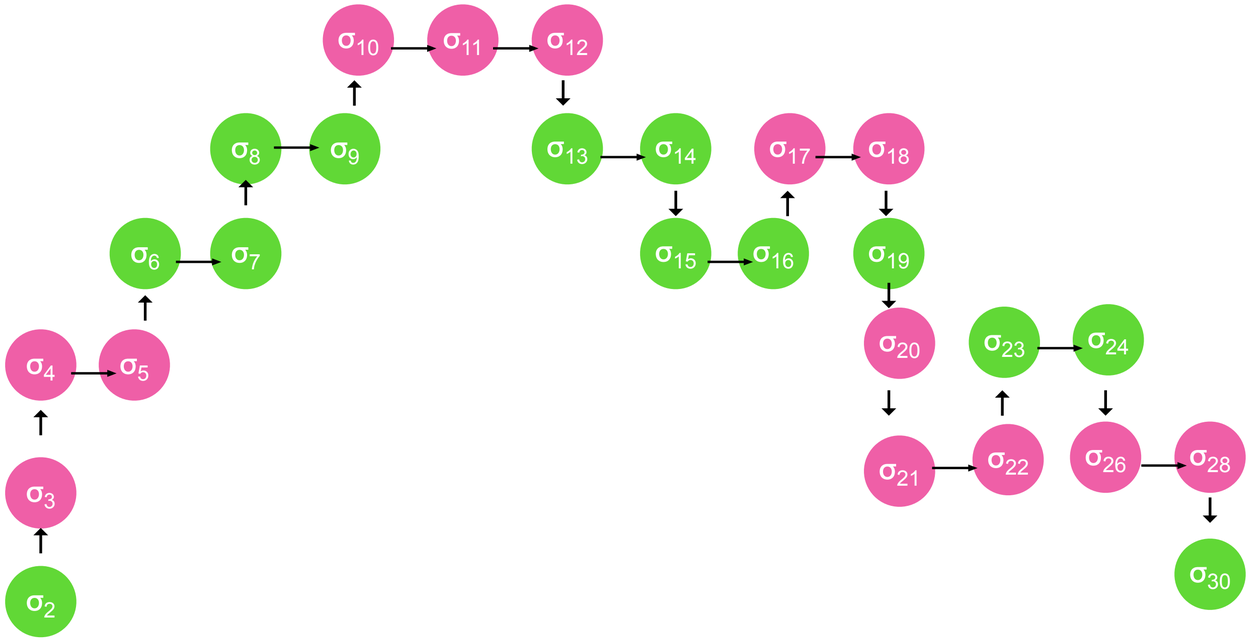
$\sigma_4 \text{ -}>^* \sigma_{21}$

$\neg(\sigma_4 \rightsquigarrow^* \sigma_{21})$

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ -}>^* \sigma'$ "while not popping the top frame of σ"



$\sigma_4 \text{ -}>^* \sigma_{18}$

$\sigma_4 \rightsquigarrow^* \sigma_{18}$

$\sigma_4 \text{ -}>^* \sigma_{20}$

$\sigma_4 \rightsquigarrow^* \sigma_{20}$

$\sigma_4 \text{ -}>^* \sigma_{21}$

$\neg(\sigma_4 \rightsquigarrow^* \sigma_{21})$

$\sigma_4 \text{ -}>^* \sigma_{24}$

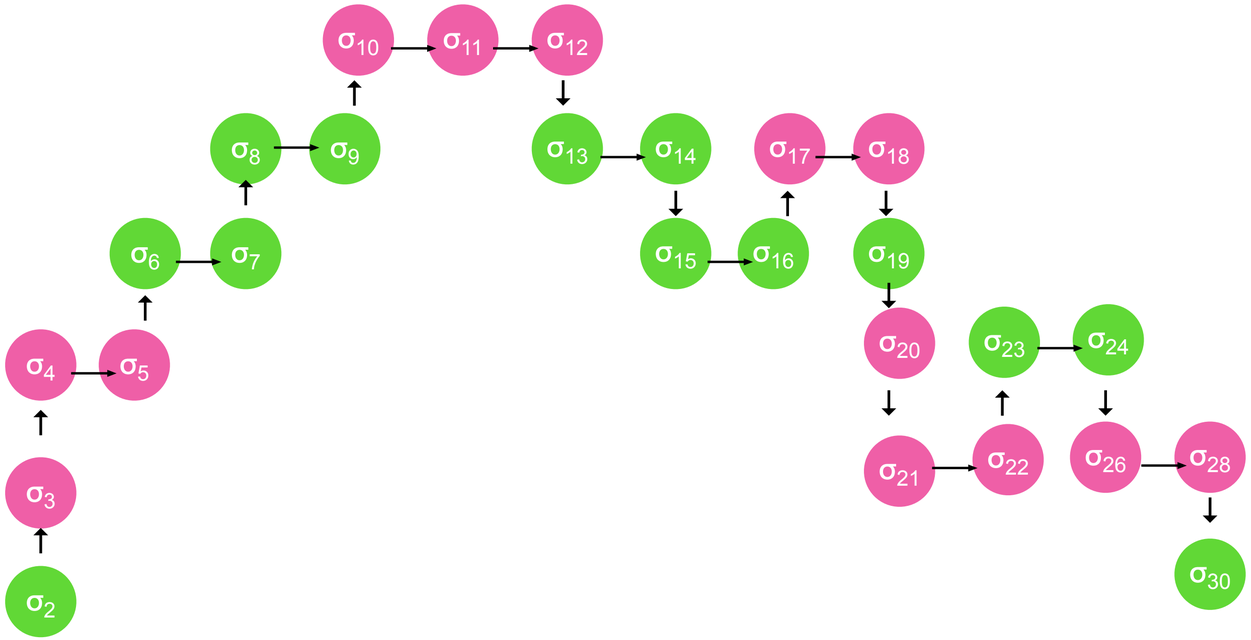
$\neg(\sigma_4 \rightsquigarrow^* \sigma_{24})$

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ ->}^* \sigma'$ “while not popping the top frame of σ”

Therefore, $\forall$x1:C1,…xn:Cn { A } maens that



σ_4 -> * σ_{18}

$\sigma_4 \rightsquigarrow^* \sigma_{18}$

σ_4 -> * σ_{20}

$\sigma_4 \rightsquigarrow^* \sigma_{20}$

σ_4 -> * σ_{21}

$\neg(\sigma_4 \rightsquigarrow^* \sigma_{21})$

σ_4 -> * σ_{24}

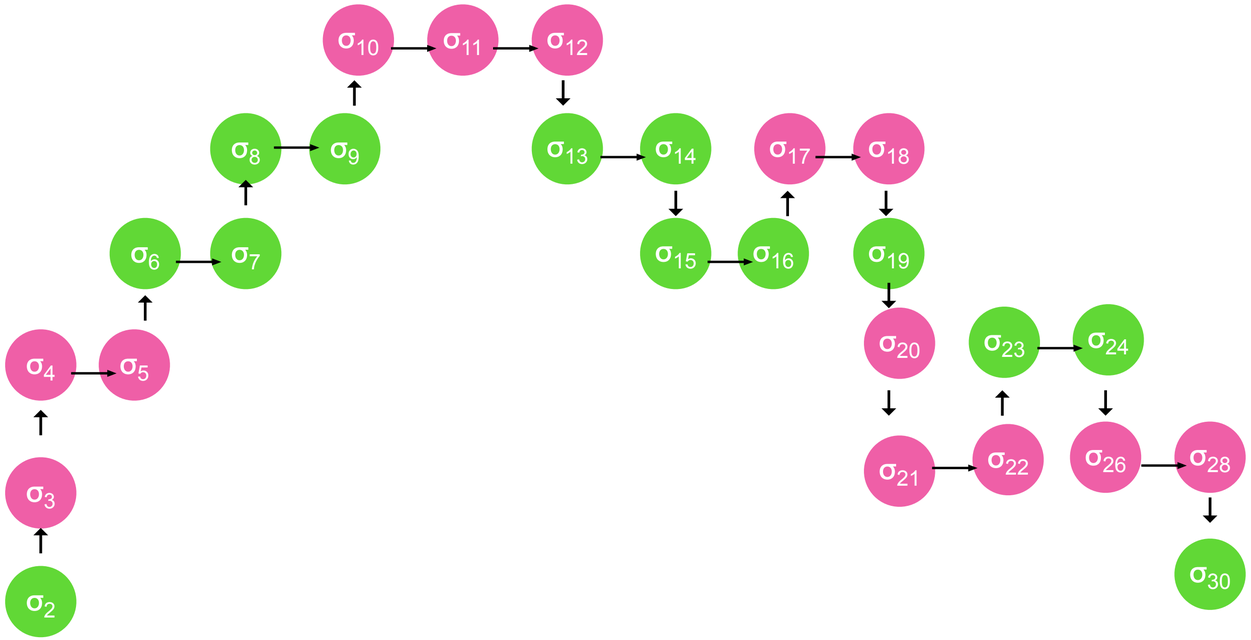
$\neg(\sigma_4 \rightsquigarrow^* \sigma_{24})$

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ ->}^* \sigma'$ “while not popping the top frame of σ”

Therefore, $\forall$x1:C1,…xn:Cn { A } maens that



σ_4 ->* σ_{18}

$\sigma_4 \rightsquigarrow^* \sigma_{18}$

σ_4 ->* σ_{20}

$\sigma_4 \rightsquigarrow^* \sigma_{20}$ A preserved from σ_4 to σ_{20}

σ_4 ->* σ_{21}

$\neg(\sigma_4 \rightsquigarrow^* \sigma_{21})$

σ_4 ->* σ_{24}

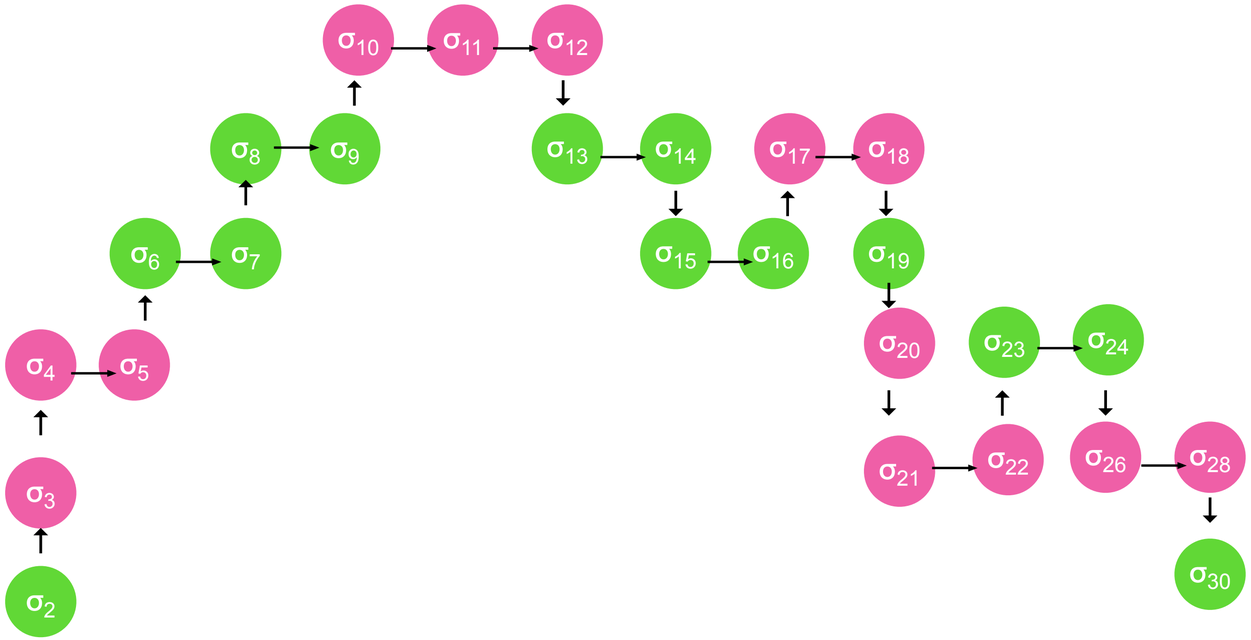
$\neg(\sigma_4 \rightsquigarrow^* \sigma_{24})$

Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ ->}^* \sigma'$ “while not popping the top frame of σ”

Therefore, $\forall$x1:C1,…xn:Cn { A } maens that



σ_4 ->* σ_{18}

$\sigma_4 \rightsquigarrow^* \sigma_{18}$ A preserved from σ_4 to σ_{18}

σ_4 ->* σ_{20}

$\sigma_4 \rightsquigarrow^* \sigma_{20}$ A preserved from σ_4 to σ_{20}

σ_4 ->* σ_{21}

$\neg(\sigma_4 \rightsquigarrow^* \sigma_{21})$

σ_4 ->* σ_{24}

$\neg(\sigma_4 \rightsquigarrow^* \sigma_{24})$

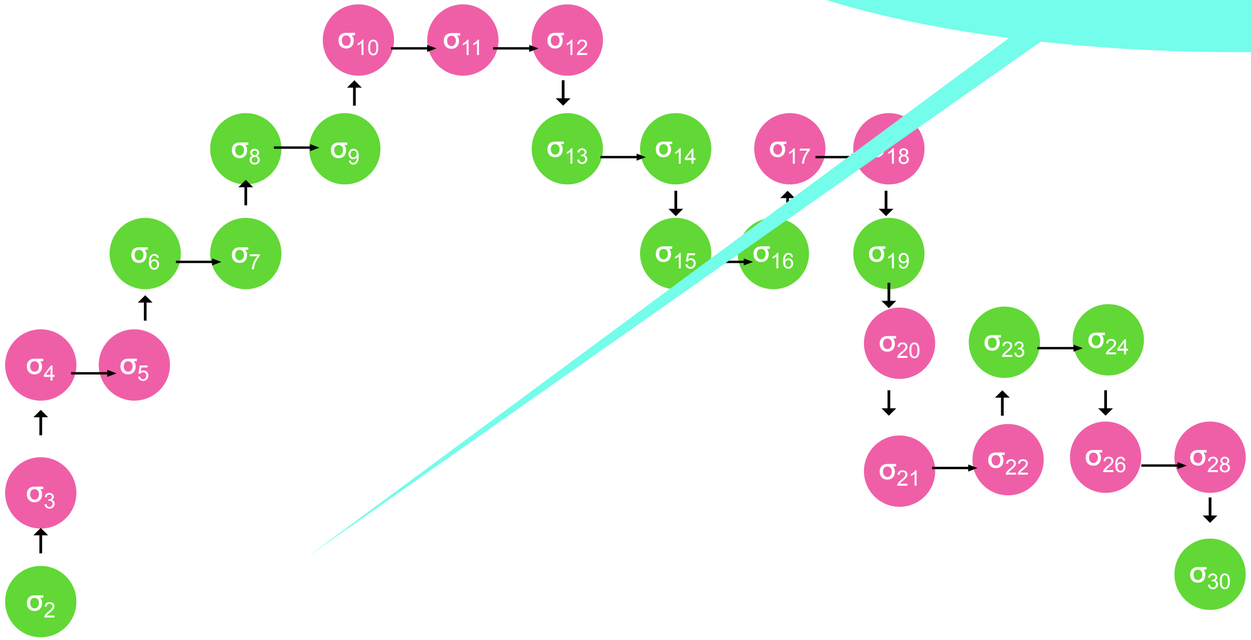
Remit_1_c : Meaning of $M'*M,\ \sigma \rightsquigarrow^* \sigma'$

Definition

$M'*M,\ \sigma \rightsquigarrow^* \sigma' \quad \triangleq \quad M'*M,\ \sigma \text{ ->}^* \sigma'$ "while not popping the top frame of σ"

{ A } maens that

Execution is "relative" to a state



σ_{18} A preserved from σ_4 to σ_{18}

σ_4 ->* σ_{20}

$\sigma_4 \rightsquigarrow^* \sigma_{20}$ A preserved from σ_4 to σ_{20}

σ_4 ->* σ_{21}

$\neg(\sigma_4 \rightsquigarrow^* \sigma_{21})$

σ_4 ->* σ_{24}

$\neg(\sigma_4 \rightsquigarrow^* \sigma_{24})$

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

S1 $\triangleq$ $\forall$ a:Account. { ⟪a⟫ }

S2 $\triangleq$ $\forall$ a:Account. { ⟪a.pwd ⟫ }

S3 $\triangleq$ $\forall$ a:Account,b:Num. { ⟪a⟫ $\wedge$ a.blnce = b }

S4 $\triangleq$ $\forall$ a:Account, b:Num. { ⟪a.pwd ⟫ $\wedge$ a.blnce $\geq$ b }

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

API - agnostic:
a.blnce, a.pwd can be ghost

Talk about effects

Talk about
emergent behaviour

$S1 \triangleq \forall\ a{:}Account.\ \{ \langle\!\langle a \rangle\!\rangle \}$

$S2 \triangleq \forall\ a{:}Account.\ \{ \langle\!\langle a.pwd \rangle\!\rangle \}$

$S3 \triangleq \forall\ a{:}Account, b{:}Num.\ \{ \langle\!\langle a \rangle\!\rangle \wedge a.blnce = b \}$

$S4 \triangleq \forall\ a{:}Account, b{:}Num.\ \{ \langle\!\langle a.pwd \rangle\!\rangle \wedge a.blnce \geq b \}$

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

API - agnostic:
a.blnce, a.pwd can be ghost

Talk about effects

Talk about
emergent behaviour

$S1 \triangleq \forall\ a{:}Account.\ \{ \langle\!\langle a \rangle\!\rangle \}$

$S2 \triangleq \forall\ a{:}Account.\ \{ \langle\!\langle a.pwd \rangle\!\rangle \}$

$S3 \triangleq \forall\ a{:}Account, b{:}Num.\ \{ \langle\!\langle a \rangle\!\rangle \wedge a.blnce = b \}$

$S4 \triangleq \forall\ a{:}Account, b{:}Num.\ \{ \langle\!\langle a.pwd \rangle\!\rangle \wedge a.blnce \geq b \}$

$M_{bad} \not\models S2$ $M_{bad} \not\models S4$

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

API - agnostic:
a.blnce, a.pwd can be ghost

Talk about effects

Talk about
emergent behaviour

$S1 \triangleq \forall\ a{:}Account.\ \{ \langle\!\langle a \rangle\!\rangle \}$

$S2 \triangleq \forall\ a{:}Account.\ \{ \langle\!\langle a.pwd \rangle\!\rangle \}$

$S3 \triangleq \forall\ a{:}Account, b{:}Num.\ \{ \langle\!\langle a \rangle\!\rangle \wedge a.blnce = b \}$

$S4 \triangleq \forall\ a{:}Account, b{:}Num.\ \{ \langle\!\langle a.pwd \rangle\!\rangle \wedge a.blnce \geq b \}$

$M_{bad} \not\models S2$ $M_{bad} \not\models S4$ $M_{fine} \models S2$ $M_{fine} \models S4$

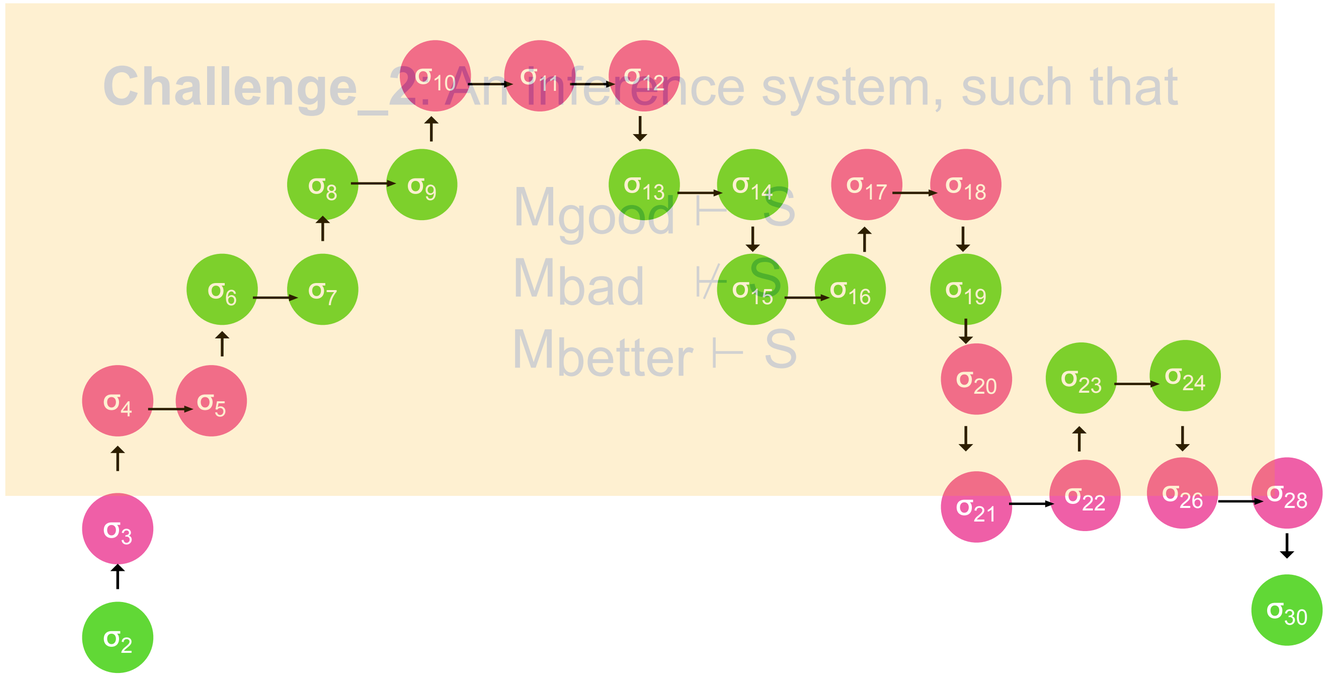
Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

In the context of arbitrary, unlimited calls from internal to external,
and arbitrary, unlimited calls from external to internal,



Challenge_2: An inference system, such that …

An assertion A is **encapsulated** by module M, if
it can only be invalidated through calls to methods from M.

Challenge_2: An inference system, such that …

An assertion A is **encapsulated** by module M, if
it can only be invalidated through calls to methods from M.

For example:

$$\mathtt{Mod}_{\mathtt{bad}} \vDash \mathit{Encaps}(\mathtt{a:Account} \wedge \mathtt{a.balance} = \mathtt{bal})$$
$$\mathtt{Mod}_{\mathtt{better}} \vDash \mathit{Encaps}(\mathtt{a:Account} \wedge \mathtt{a.balance} = \mathtt{bal})$$

Challenge_2: An inference system, such that …

An assertion A is **encapsulated** by module M, if
it can only be invalidated through calls to methods from M.

For example:

$$\texttt{Mod}_{\texttt{bad}} \vDash \mathit{Encaps}(\texttt{a : Account} \wedge \texttt{a.balance} = \texttt{bal})$$
$$\texttt{Mod}_{\texttt{better}} \vDash \mathit{Encaps}(\texttt{a : Account} \wedge \texttt{a.balance} = \texttt{bal})$$

Assume two further modules, Mod_{ul} and Mod_{pl}, which use ledgers to keep a map between accounts and their balances, which export functions that allow the update of this map. In Mod_{ul} the ledger is *not* protected, while in Mod_{pl} the ledger *is* protected.

$$Mod_{ul} \nvDash \mathit{Encaps}(\texttt{a : Account} \wedge \texttt{a.balance} = \texttt{bal})$$
$$Mod_{pl} \vDash \mathit{Encaps}(\texttt{a : Account} \wedge \texttt{a.balance} = \texttt{bal})$$

Challenge_2: An inference system, such that …

Three Stages

Assume an ulderlying Hoare logic of triples with usual meaning

$$M \vdash_{ul} \{ A \}\, s \,\{ A' \}$$

1st stage Expand it to Hoare logic of triples with usual meaning

$$M \vdash \{A\}\, s \,\{A'\},$$

satisfying the precon

2nd Stage Expand triples to quadruples

$$M \vdash \{A\}\, s \,\{A'\} \parallel \{A''\};$$

Which promises that

- termination of *s* leads to a state satisfying *A'*
- intermediate external states satisfy *A"*

3rd Stage Rules for module satisfying a specification

$$M \vdash S$$

Challenge_2: An inference system, such that …

1st stage

We expand underlying Hoare logic to Hoare logic of triples with usual meaning

$$\textbf{EXTEND}\quad \frac{M \vdash_{ul} \{ A \}\, s\, \{ A' \} \quad s \text{ contains no method call}}{M \vdash \{A\}\, s\, \{A'\}}$$

$$\textbf{TYPES-1}\quad \frac{s \text{ contains no method call}}{M \vdash \{x : C\}\, s\, \{x : C\}}$$

Challenge_2: An inference system, such that …

2nd stage

We expand triples to quadruples

$$\textbf{INV}\quad \frac{M \vdash \{A\}\, s\, \{A'\}}{M \vdash \{A\}\, s\, \{A'\} \parallel \{A''\}}$$

$$\textbf{TYPES-2}\quad \frac{M \vdash \{A\}\, s\, \{A'\} \parallel \{A''\}}{M \vdash \{x : C \wedge A\}\, s\, \{x : C \wedge A'\} \parallel \{A''\}}$$

$$\textbf{COMBINE}\quad \frac{M \vdash \{A_1\}\, s\, \{A_2\} \parallel \{A\} \qquad M \vdash \{A_3\}\, s\, \{A_4\} \parallel \{A\}}{M \vdash \{A_1 \wedge A_3\}\, s\, \{A_2 \wedge A_4\} \parallel \{A\}}$$

$$\textbf{SEQU}\quad \frac{M \vdash \{A_1\}\, s_1\, \{A_2\} \parallel \{A\} \qquad M \vdash \{A_2\}\, s_2\, \{A_3\} A}{M \vdash \{A_1\}\, s_1; s_2\, \{A_3\} \parallel \{A\}}$$

$$\textbf{CONSEQU}\quad \frac{M \vdash \{A_2\}\, s\, \{A_3\} \parallel \{A_4\} \qquad M \vdash A_1 \rightarrow A_2 \qquad M \vdash A_3 \rightarrow A_5 \qquad M \vdash A_4 \rightarrow A_6}{M \vdash \{A_2\}\, s\, \{A_5\} \parallel \{A_6\}}$$

Challenge_2: An inference system, such that …

3rd stage

$$\frac{M \vdash \mathscr{S}pec(M)}{\vdash M}\ \textsc{WellFrm_Mod} \qquad \frac{M \vdash S_1 \qquad M \vdash S_2}{M \vdash S_1 \wedge S_2}\ \textsc{Comb_Spec}$$

Challenge_2: An inference system, such that …

3rd stage

$$\frac{???}{M \vdash \forall \overline{x : C}\{A\}}\ \textsc{invariant}$$

Challenge_2: An inference system, such that …

$$\frac{M \vdash Encps(\overline{x : C} \wedge A)}{M \vdash \forall \overline{x : C}\{A\}}\ \textbf{INVARIANT}$$

Challenge_2: An inference system, such that …

$$\frac{M \vdash Encps(\overline{x : C} \wedge A) \qquad \forall D, m : \quad \texttt{mBody}(m, D, M) = \texttt{public}\ (\overline{y : C})\{\ stmt\ \} \implies}{M \vdash \forall \overline{x : C}\{A\}}\ \textbf{INVARIANT}$$

Challenge_2: An inference system, such that …

$$\frac{\begin{array}{l} M \vdash Encps(\overline{x : C} \wedge A) \\ \forall D, m : \quad \texttt{mBody}(m, D, M) = \texttt{public}\ (\overline{y : C})\{\ stmt\ \} \implies \\ \qquad M \vdash \{\ \texttt{this} : \texttt{D}, \overline{y : D}, \overline{x : C} \wedge PRE(A)\ \}\ stmt\ \{\ POST(A)\ \} \parallel \{\ POST(A)\ \} \end{array}}{M \vdash \forall \overline{x : C}\{A\}}\ \textbf{INVARIANT}$$

Remit_3: An inference system, such that we can prove external calls

Challenge_4: An inference system, such we can prove external calls

$$\frac{}{M \vdash \{\ y_0 : \texttt{ext} \quad , \quad \} \ u := y_0.m(y_1, ..y_n)\ \{\ ???\ \} \parallel \{\ ??\ \}}\ [\textsc{Call\_Ext}]$$

Challenge_4: An inference system, such we can prove external calls

$$\frac{\vdash \; M \; : \; \forall \overline{x : D}\{A\}}{M \vdash \{ \; y_0 : \texttt{ext} \quad \overline{\quad\quad} \quad , \quad\quad\quad\quad \} \; u := y_0.m(y_1, ..y_n) \; \{ \quad ??? \quad \} \; \| \; \{ \; ?? \; \}} \; [\textsc{Call\_Ext}]$$

Challenge_4: An inference system, such we can prove external calls

$$\frac{\vdash \; M \; : \; \forall \overline{x : D}\{A\}}{M \vdash \{ \; y_0 : \texttt{ext} \wedge \overline{x : D} \; , \qquad \} \; u := y_0.m(y_1, ..y_n) \; \{ \quad ??? \quad \} \parallel \{ \; ?? \; \}} \; [\textsc{Call\_Ext}]$$

Challenge_4: An inference system, such we can prove external calls

$$\frac{\vdash \; M \; : \; \forall \overline{x : D}\{A\}}{M \vdash \{ \; y_0 : \texttt{ext} \wedge \overline{x : D} \wedge A \multimap \forall \overline{y} \; \} \, u := y_0.m(y_1, ..y_n) \, \{ \quad ??? \quad \} \parallel \{ \; ?? \; \}} \; [\textsc{Call\_Ext}]$$

Challenge_4: An inference system, such we can prove external calls

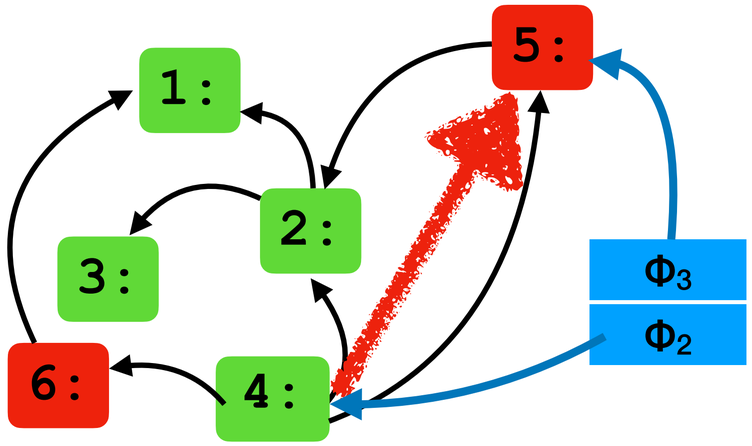
$$\frac{\vdash \; M \; : \; \forall \overline{x : D}\{A\}}{M \vdash \{ \; y_0 : \texttt{ext} \wedge \overline{x : D} \wedge A \multimap \forall \overline{y} \; \} \; u := y_0.m(y_1, ..y_n) \; \{ \; A \multimap \forall \overline{y} \; \} \; \| \; \{ \; ?? \; \}} \; [\textsc{Call\_Ext}]$$

Challenge_4: An inference system, such we can prove external calls

$$\frac{\vdash \; M \; : \; \forall \overline{x : D}\{A\}}{M \vdash \{ \; y_0 : \texttt{ext} \wedge \overline{x : D} \wedge A \multimap \forall \overline{y} \; \} \, u := y_0.m(y_1, ..y_n) \, \{ \; A \multimap \forall \overline{y} \; \} \; \| \; \{ \; A \; \}} \; [\textsc{Call\_Ext}]$$

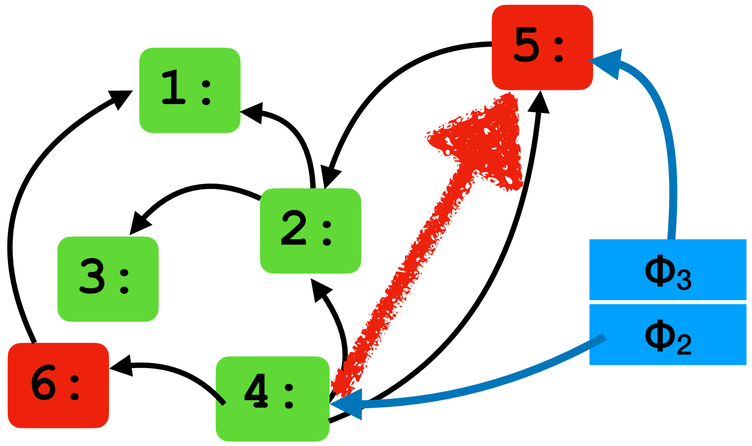
Challenge_4: An inference system, such we can prove external calls

$$\frac{\vdash \; M \; : \; \forall \overline{x : D}\{A\}}{M \vdash \{ \; y_0 : \texttt{ext} \wedge \overline{x : D} \wedge A \multimap \forall \overline{y} \; \} \, u := y_0.m(y_1, ..y_n) \, \{ \; A \multimap \forall \overline{y} \; \} \; \| \; \{ \; A \; \}} \; [\textsc{Call\_Ext}]$$



Challenge_4: An inference system, such we can prove external calls

$$\frac{\vdash \; M \; : \; \forall \overline{x : D}\{A\}}{M \vdash \{\; y_0 : \texttt{ext} \wedge \overline{x : D} \wedge A \multimap \nabla \overline{y} \;\}\; u := y_0.m(y_1, ..y_n) \;\{\; A \multimap \nabla \overline{y} \;\} \;\|\; \{\; A \;\}} \; [\textsc{Call\_Ext}]$$

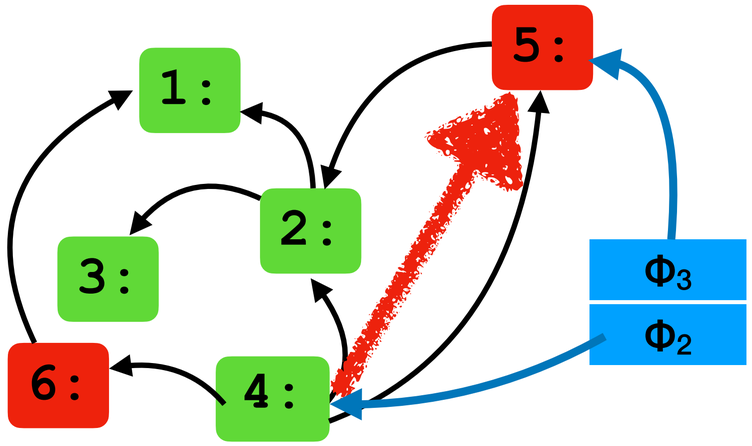


Definition 5.7. [The $\multimap\nabla$ operator] is defined below

$$\begin{array}{rcl@{\qquad}rcl}
(\langle e \rangle) \multimap \nabla \overline{y} & \triangleq & \langle e \rangle \leftrightarrowtail \overline{y} & (A_1 \wedge A_2) \multimap \nabla \overline{y} & \triangleq & (A_1 \multimap \nabla \overline{y}) \wedge (A_2 \multimap \nabla \overline{y}) \\
(\langle e \rangle \leftrightarrowtail \overline{u}) \multimap \nabla \overline{y} & \triangleq & \langle e \rangle \leftrightarrowtail \overline{u} & (\forall x : C.A) \multimap \nabla \overline{y} & \triangleq & \forall x : C.(A \multimap \nabla \overline{y}) \\
(\,\mathbf{intl}\, e\,) \multimap \nabla \overline{y} & \triangleq & \mathbf{intl}\, e & (\neg A) \multimap \nabla \overline{y} & \triangleq & \neg (A \multimap \nabla \overline{y}) \\
e \multimap \nabla \overline{y} & \triangleq & e & (e : C) \multimap \nabla \overline{y} & \triangleq & e : C
\end{array}$$

Challenge_4: An inference system, such we can prove external calls

$$\frac{\vdash \; M \; : \; \forall \overline{x : D}\{A\}}{M \vdash \{\; y_0 : \texttt{ext} \wedge \overline{x : D} \wedge A \multimap \nabla \overline{y} \;\}\; u := y_0.m(y_1, ..y_n) \;\{\; A \multimap \nabla \overline{y} \;\} \;\|\; \{\; A \;\}} \; [\textsc{Call\_Ext}]$$



Definition 5.7. [The $\multimap\nabla$ operator] is defined below

$$\begin{array}{rcl@{\qquad}rcl}
(\langle e \rangle) \multimap \nabla \overline{y} & \triangleq & \langle e \rangle \leftrightarrowtail \overline{y} & (A_1 \wedge A_2) \multimap \nabla \overline{y} & \triangleq & (A_1 \multimap \nabla \overline{y}) \wedge (A_2 \multimap \nabla \overline{y}) \\
(\langle e \rangle \leftrightarrowtail \overline{u}) \multimap \nabla \overline{y} & \triangleq & \langle e \rangle \leftrightarrowtail \overline{u} & (\forall x : C.A) \multimap \nabla \overline{y} & \triangleq & \forall x : C.(A \multimap \nabla \overline{y}) \\
(\,\mathbf{intl}\, e\,) \multimap \nabla \overline{y} & \triangleq & \mathbf{intl}\, e & (\neg A) \multimap \nabla \overline{y} & \triangleq & \neg (A \multimap \nabla \overline{y}) \\
e \multimap \nabla \overline{y} & \triangleq & e & (e : C) \multimap \nabla \overline{y} & \triangleq & e : C
\end{array}$$

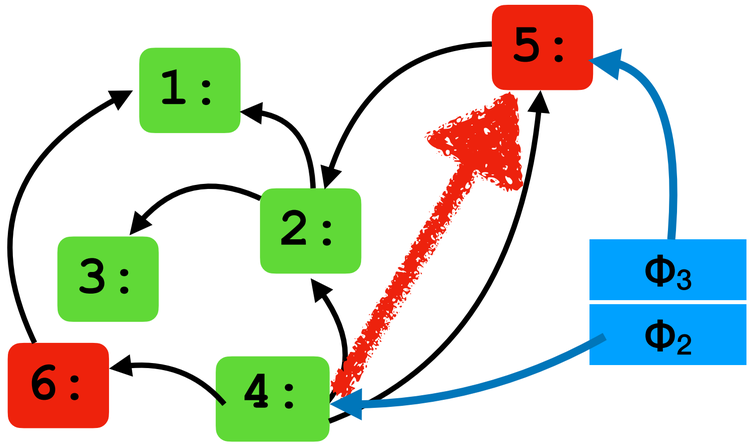
Lemma 5.8. For any state σ, assertion A, and variables $\overline{y}, \overline{z}$, disjoint with one another: If $fv(A) = \emptyset$, then

$$\begin{array}{lcl}
(1)\;\; M, \sigma \models A \multimap \nabla \overline{y} & \Longrightarrow & M,\; \sigma \nabla \overline{y} \models A \\
(2)\;\; M, \sigma \nabla (\overline{y}, \overline{z}) \models A & \Longrightarrow & M, \sigma \models A \multimap \nabla \overline{y}
\end{array}$$

Challenge_4: An inference system, such we can prove external calls

Protection is
"relative" to a frame;
Our $-\!\!\triangledown$ operator
helps us switch to callee's view

$$M \vdash \{\ y_0 : \texttt{ext} \wedge \overline{x : D} \wedge A \mathrel{-\!\!\triangledown} \overline{y}\ \}\ u := y_0 \ldots$$



Definition 5.7. [The $-\!\!\triangledown$ operator] is defined below

$$\begin{array}{rcl@{\qquad}rcl}
(\langle e \rangle) \mathrel{-\!\!\triangledown} \overline{y} & \triangleq & \langle e \rangle \leftrightarrow\!\!\!\!\times\ \overline{y} & (A_1 \wedge A_2) \mathrel{-\!\!\triangledown} \overline{y} & \triangleq & (A_1 \mathrel{-\!\!\triangledown} \overline{y}) \wedge (A_2 \mathrel{-\!\!\triangledown} \overline{y}) \\
(\langle e \rangle \leftrightarrow\!\!\!\!\times\ \overline{u}) \mathrel{-\!\!\triangledown} \overline{y} & \triangleq & \langle e \rangle \leftrightarrow\!\!\!\!\times\ \overline{u} & (\forall x : C.A) \mathrel{-\!\!\triangledown} \overline{y} & \triangleq & \forall x : C.(A \mathrel{-\!\!\triangledown} \overline{y}) \\
(\ \mathbf{intl}\ e\ ) \mathrel{-\!\!\triangledown} \overline{y} & \triangleq & \mathbf{intl}\ e & (\neg A) \mathrel{-\!\!\triangledown} \overline{y} & \triangleq & \neg (A \mathrel{-\!\!\triangledown} \overline{y}) \\
e \mathrel{-\!\!\triangledown} \overline{y} & \triangleq & e & (e : C) \mathrel{-\!\!\triangledown} \overline{y} & \triangleq & e : C
\end{array}$$

Lemma 5.8. For any state σ, assertion A, and variables $\overline{y}, \overline{z}$, disjoint with one another: If $fv(A) = \emptyset$, then

$$\begin{array}{lcl}
(1)\ M, \sigma \models A \mathrel{-\!\!\triangledown} \overline{y} & \Longrightarrow & M,\ \sigma \triangledown \overline{y} \models A \\
(2)\ M, \sigma \triangledown (\overline{y}, \overline{z}) \models\ A & \Longrightarrow & M, \sigma \models A \mathrel{-\!\!\triangledown} \overline{y}
\end{array}$$

- Distinction between external/internal objects

- ∀ x: … { A } Two state invariants for external states / relative execution

- Specifications talk about necessary conditions for effect:

  ∀ x: … { ⟪e ⟫ ∧ A }

  means that **capability** e is needed in order to invalidate A

- ⟪e ⟫: capability e is protected from reachable external objects

Summary

- API-agrnositc spec,
- “Algorithmic” inference system system,
- Reason with open calls

- Protection, ⟪e ⟫, is relative to state σ. Use —▽ to switch view
- Execution, M’*M, σ ↝* σ’ , is relative to state σ

- Surpises:
  - Use sufficient conditions to talk about necessary conditions
  - from temporal operators/logics to invariants/Hoare logics