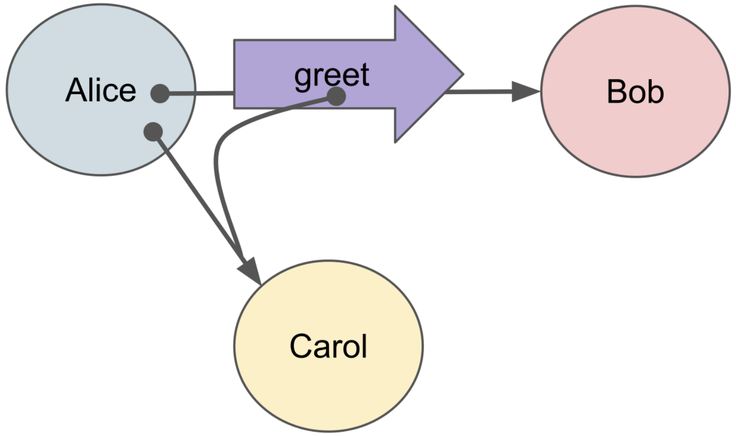
Reasoning about External Calls using Object Capabilities

Sophia Drossopoulou, Imperial College London

WG 2.3 Athens, 19th May 2025



James Noble

Susan Eisenbach

Julian Mackay

The Problem

External and Internal Code is tightly intertwined.

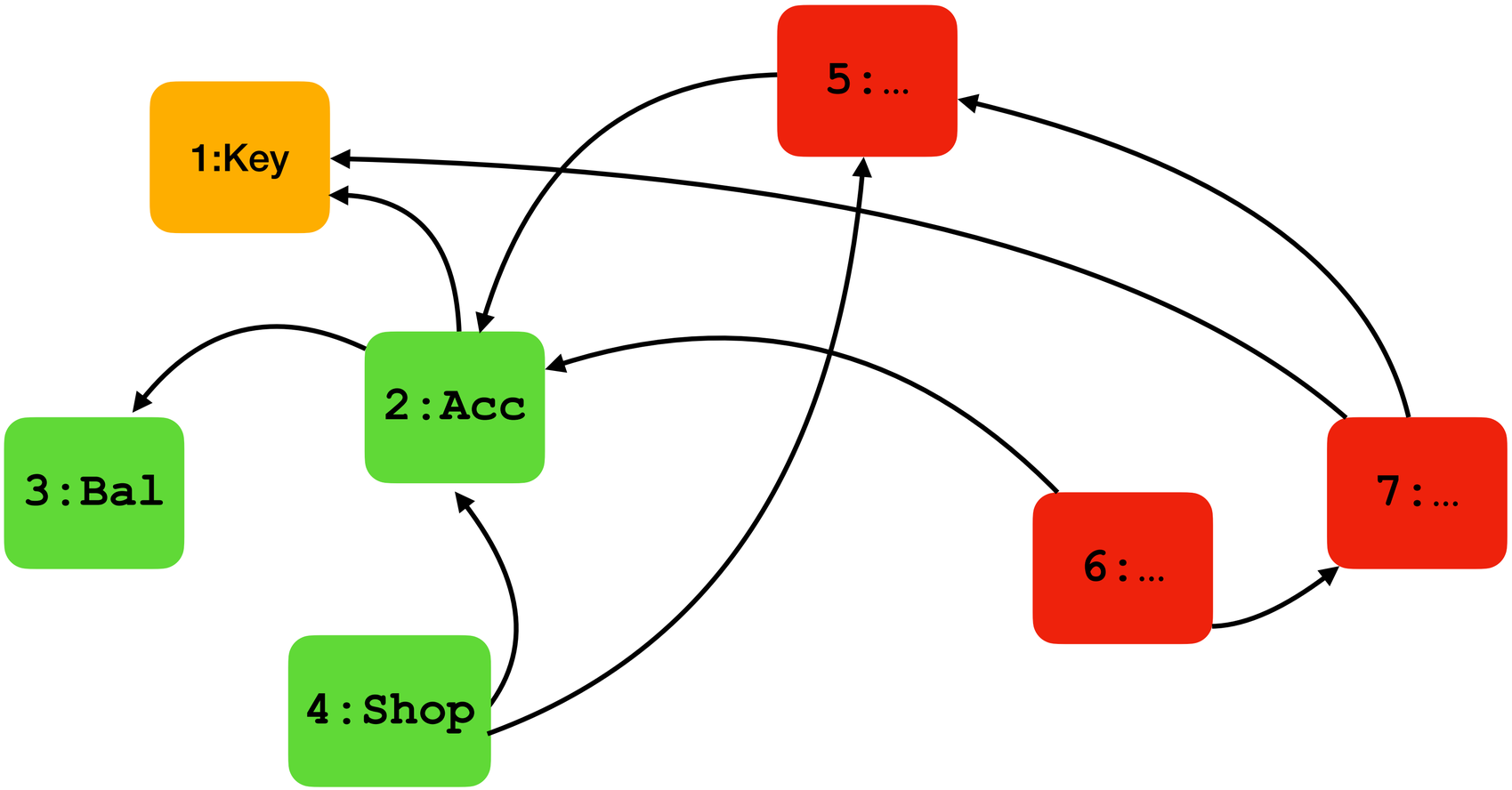
Object Capabilities used to control External Effects

Our Remit

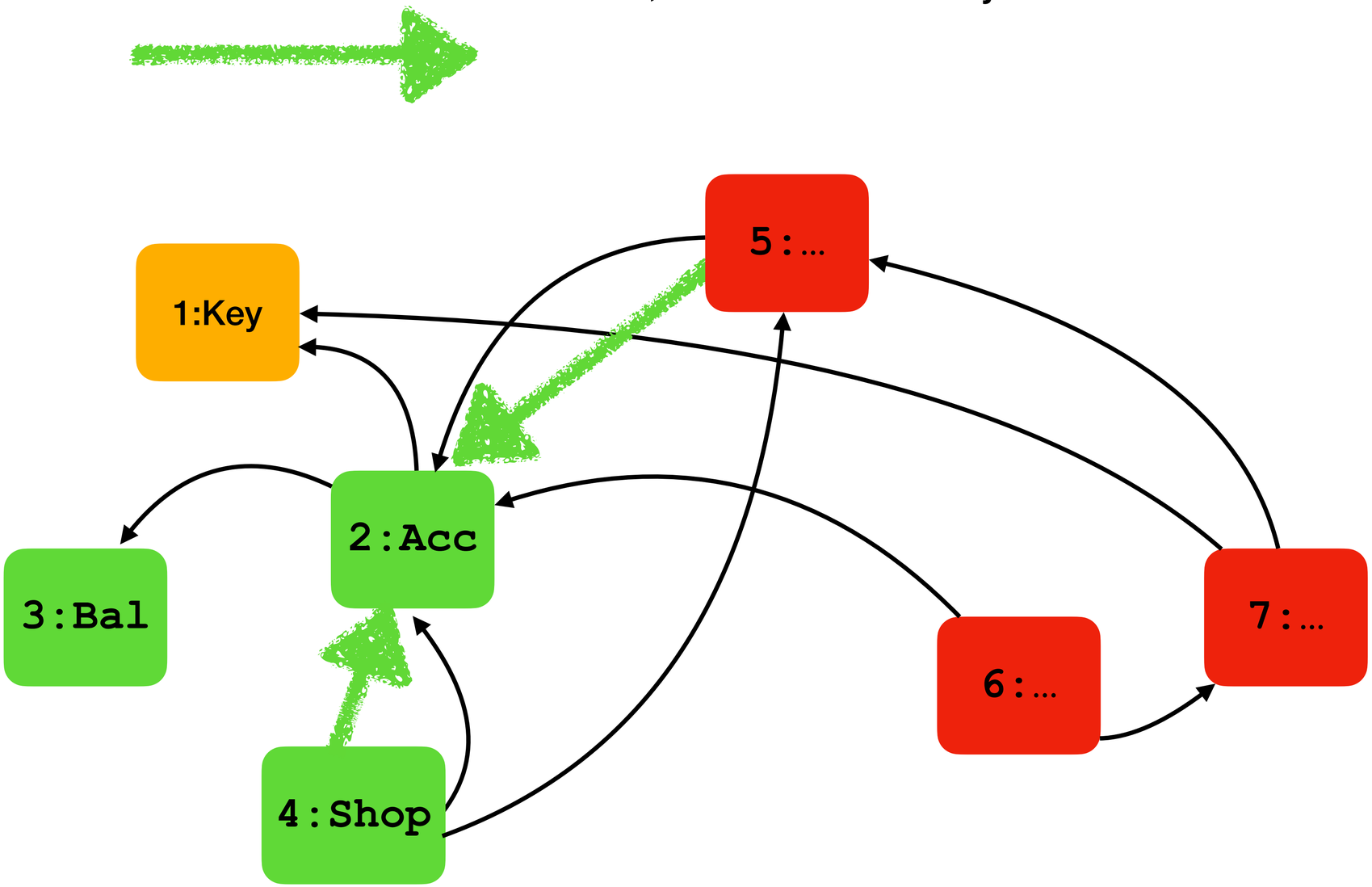
Reason about Object Capabilities controling Effects:
1) Specification, 2) Verification

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

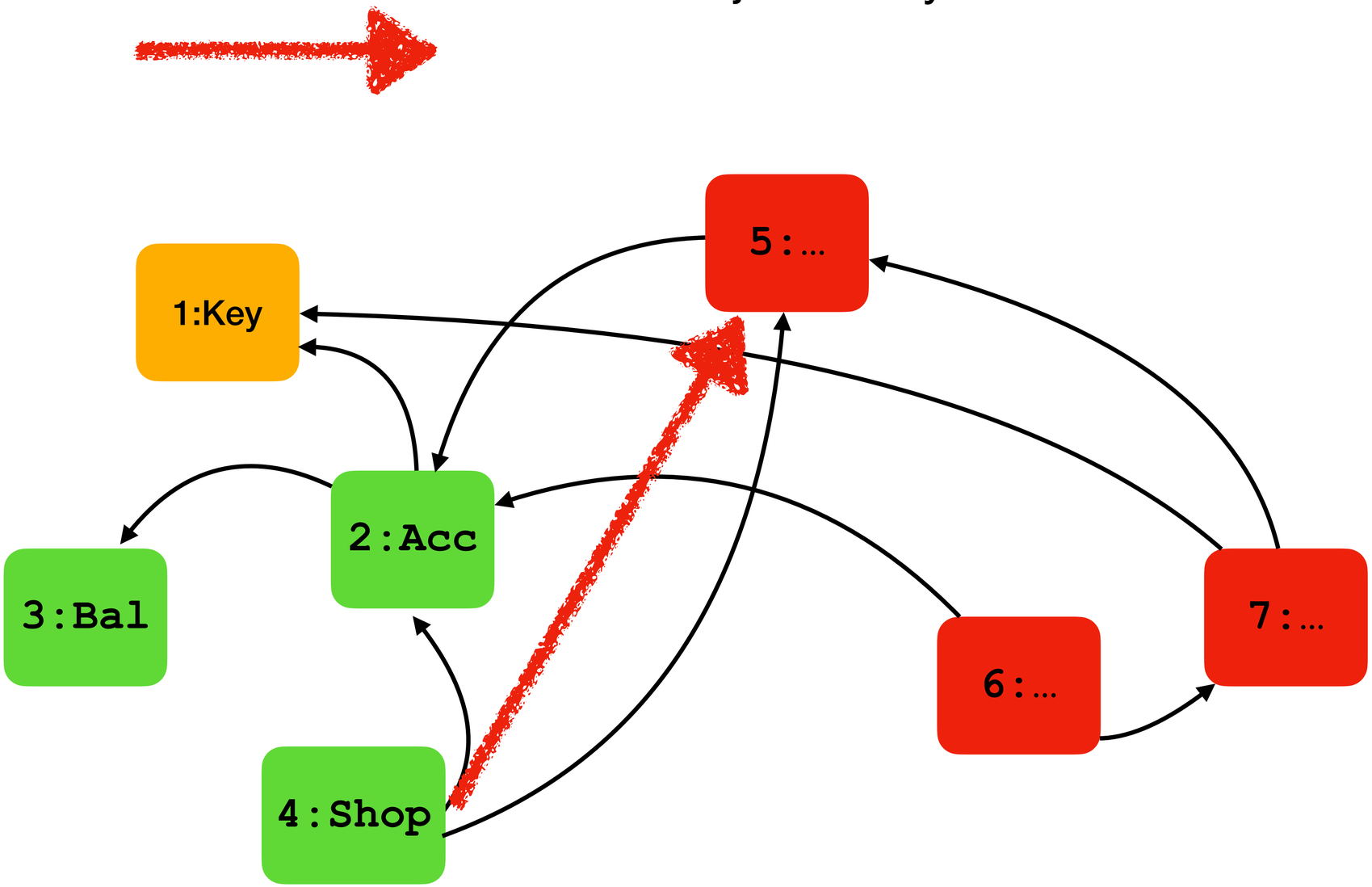
Capability objects (ocap) **are necessary** for certain effects.

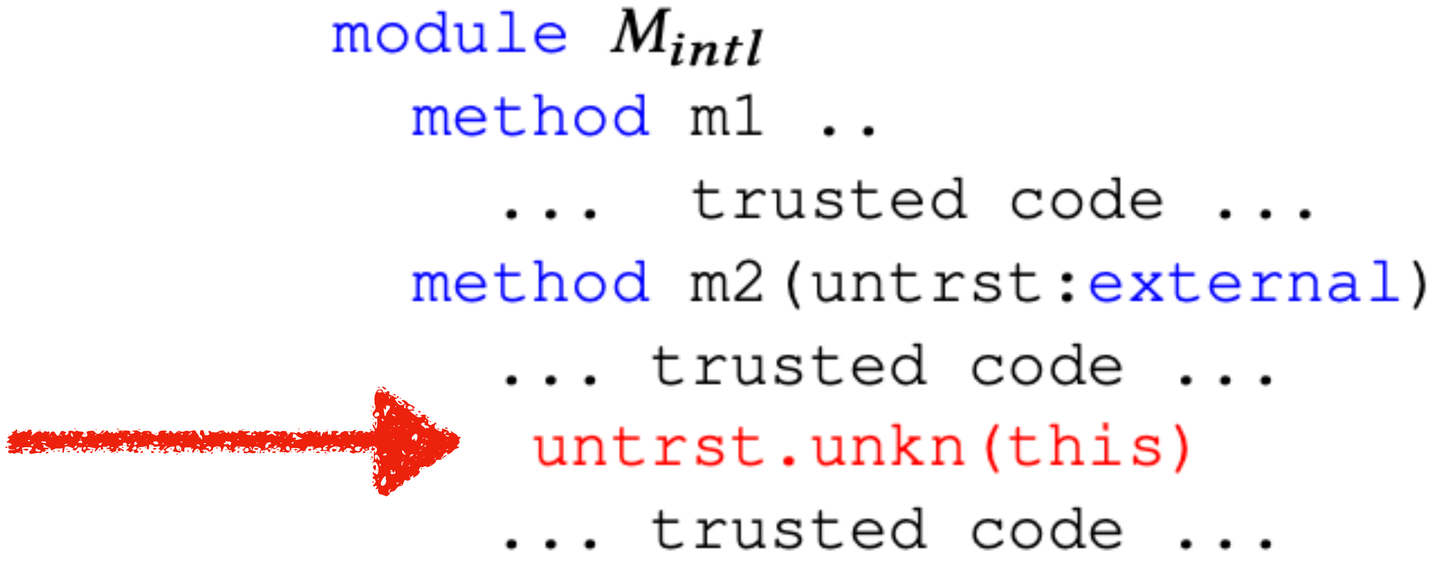


Internal Calls: Internal, or external objects call methods on internal objects.



External Calls: Internal objects may call methods on external objects.





What are the possible effects at the external call?

1) Some effects will **never** happen cf. object invariants

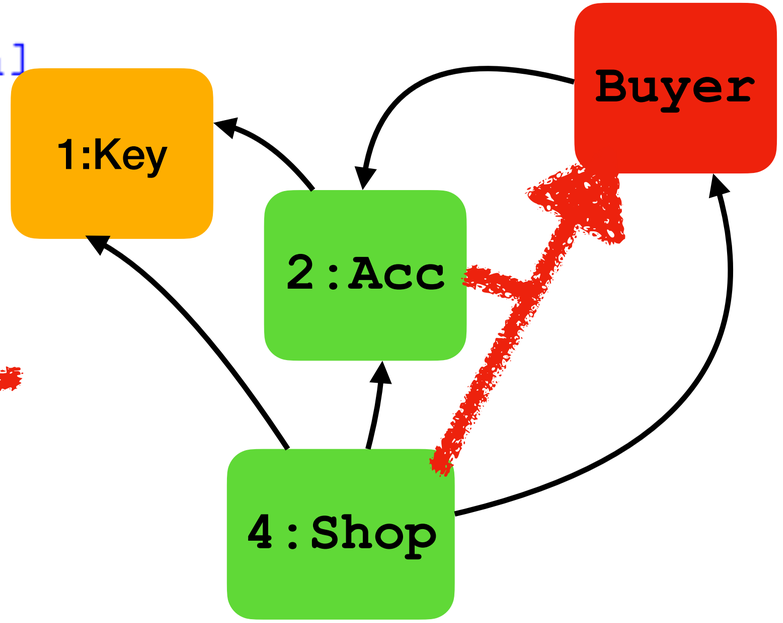
2) Some effects **may** happen — *but only if* external access to certain capabilities.

Our Work

```
class Shop

  field accnt:Account, invntry:Inventory, clients:external

  public method buy(buyer:external, anItem:Item)
     int price = anItem.price
     int oldBlnce = this.accnt.blnce
     buyer.pay(this.accnt, price)
     if (this.accnt.blnce == oldBlnce+price)
        this.send(buyer,anItem)
     else
        buyer.tell("you have not paid me")
```



What are the possible effects at the external call?

Q Can buyer steal money from the shop's account?

A No, if a) Money not reduced unless external access to account's key

b) Buyer has no access to account's key

c) Module does not leak the account's key

Some effects **may** happen — *but only if* some condition/access to ocap

1st Attempt

A ::= … | A->A | **will**<A> | **was**<A> | **ext**<e> | **acc**<e,e'>

2nd Attempt

No logic

A ::= … | A->A | **ext**<e> | **acc**<e,e'>

S ::= **from** A_{pre} **to** Apost **onlyIf** A_{nec} |
from A_{pre} **to** Apost **onlyThrough** A_{nec}

Logic,
only internal calls

Current Attempt

A ::= … | **ext**<e> | **prot**<e> | **prot**<e,e'>

S ::= ¬ A ∧ ¬ A_{nec} "is invariant"

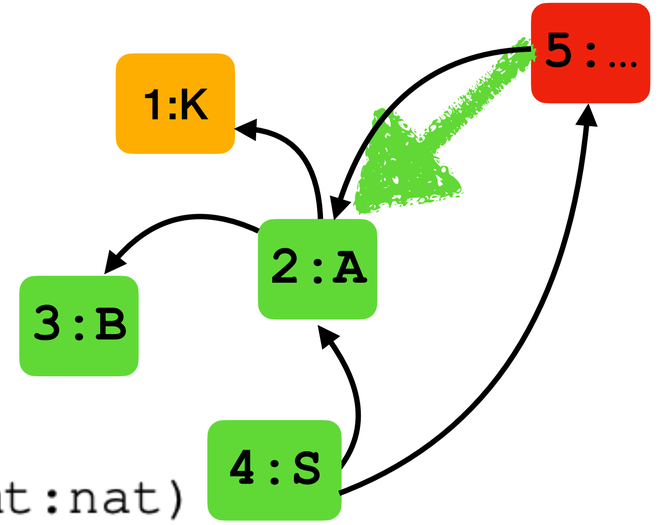
Logic,
external calls

The Account example in Code

(fields are private)

Three Modules

```
module M_good
  class Shop    ... as earlier ...
  class Account
    field blnce:int
    field key:Key
    public method transfer(dest:Account, key':Key, amt:nat)
       if (this.key==key')  this.blnce-=amt; dest.blnce+=amt
    public method set(key':Key)
       if (this.key==null)  this.key=key'
```




```
module M_bad
  ... as earlier ...
  public method set(key':Key)
    this.key=key'
```

```
module M_fine
  ... as earlier ...
  public method set(key',key'':Key)
    if (this.key==key') this.key=key''
```

The trouble is … the emergent behaviour

M_{bad}

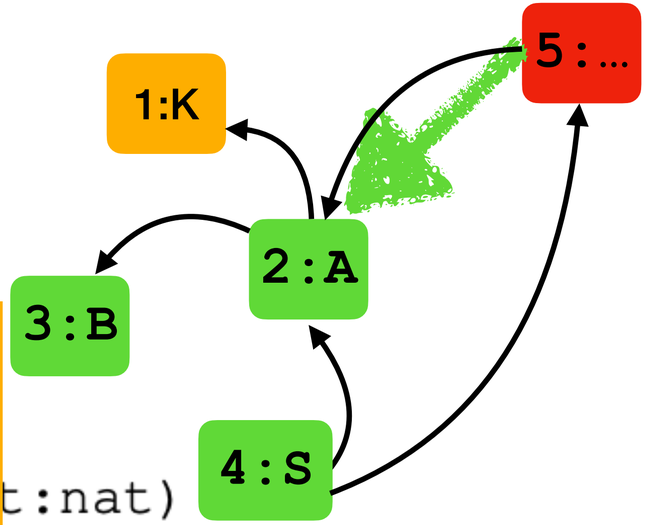
1) Attacker calls `set(…)` and changes the password
2) Attacker calls `transfer(…)` and uses password to withdraw money

Parity MultiSig Wallet Hack (150,000 ETH (~30M USD) 2017)

1) Attacker calls `initWallet(…)` and makes themselves the single owner
2) Attacker calls `execute(…)` and withdrawns all funds

Three Modules

```
module Mgood
  class Shop    ... as earlier ...
  class Account
    field blnc
    field key:
    public met                                    t:nat)
       if (thi                                    ce+=amt
    public met
       if (thi
```



Remit_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

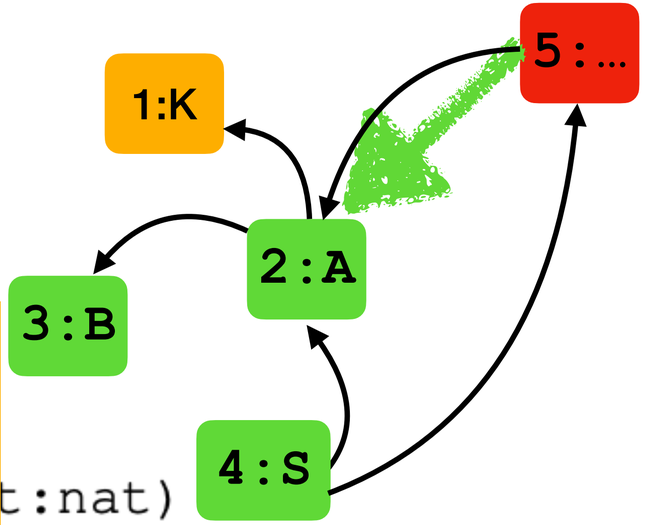
$M_{fine} \models S$

```
module Mbad
  ... as earlier ...
  public method set(key':Key)
    this.key=key'
```

```
module Mfine
  ... as earlier ...
  public method set(key',key'':Key)
    if (this.key==key') this.key=key''
```

Three Modules

```
module Mgood
  class Shop    ... as earlier ...
  class Account
    field blnc
    field key:
    public met                                    t:nat)
       if (thi                                    ce+=amt
    public met
       if (thi
```



Remit_2: An inference system with which to prove

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{fine} \vdash S$

```
module Mbad
  ... as earlier ...
  public method set(key':Key)
    this.key=key'
```

```
module Mfine
  ... as earlier ...
  public method set(key',key'':Key)
    if (this.key==key') this.key=key''
```

Three Modules

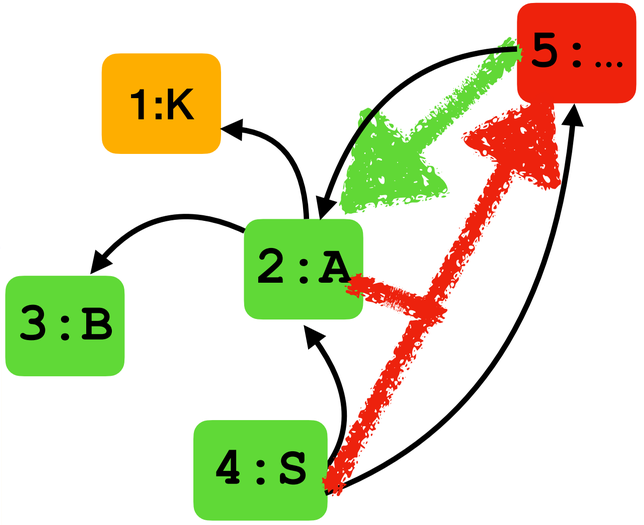
```
class Shop

  field accnt:Account, invntry:Inventory, clients:external

  public method
    int price =
    int oldBlnc
    buyer.pay(t
    if (this.ac
       this.sen
    else
       buyer.te
```

Remit_3: An inference system with which to prove

external calls



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

In Summary

External Objects

- may have access to internal objects
- may execute arbitrary code
- may invoke any public internal method
- may collude with one another
- may *not* directly read/write internal fields

Our Specifications

- guarantee that certain effects happen only under certain condtions
- In genereal do *not* preclude these conditions

Remit_1: A specification language,

and module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{fine} \models S$

We want to give formal meaning to

(*) Effect (*E*) can be caused
- **only** by external objects calling methods on internal objects, and
- **only** if the causing object has access to capability.

Assume that effect *E* invalidates assertion *A*.

Then, we could formalize **(*)** through

A ∧ “no external access to OCAP” is “invariant”

We need to determine

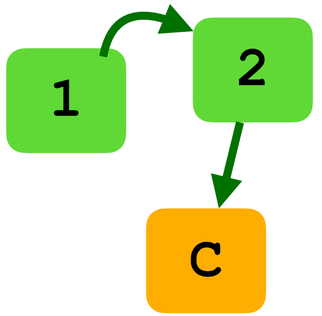
- “no external access to OCAP”
- “invariant”

We now determine

- “no external access to OCAP”
- “invariant”

1st Answer

- No external objects exist.



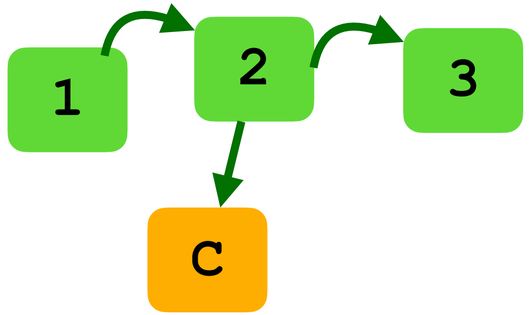
***Unsound*!**

We now determine

- “no external access to OCAP”
- “invariant”

2nd Answer

- No external objects exist.
- No external objects created.



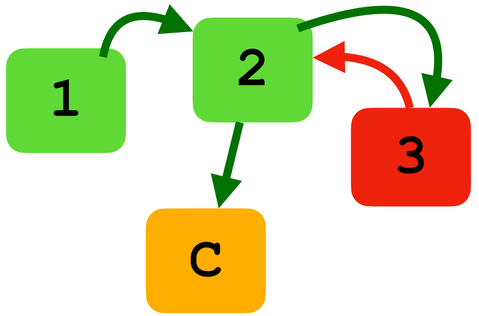
Too Strong!

We now determine

- “no external access to OCAP”
- “invariant”

3rd Answer

- No external object has direct access to OCAP



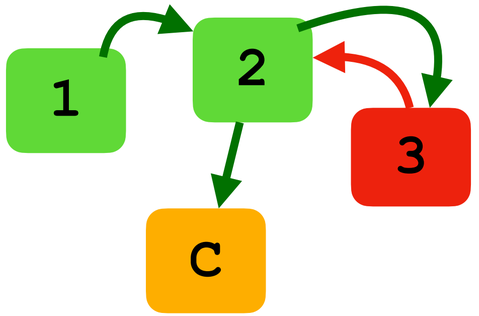
***Unsound*!**

We now determine

- “no external access to OCAP”
- “invariant”

4th Answer

- No external object has direct access to OCAP.
- No inrernal objects leak capability to OCAP.



Too strong!

We now determine

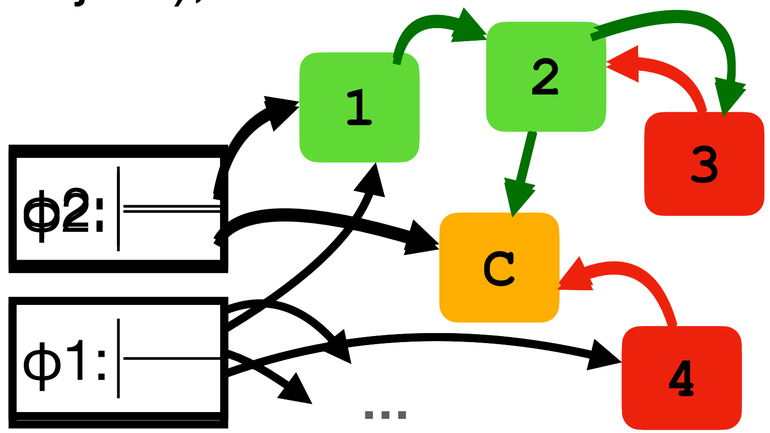
- “no external access to OCAP”
- “invariant”

5th Answer

- No *currently accessible* external object has direct access to OCAP.
- No internal objects leak access to OCAP.

invariant ≙ preserved in external states (`this` is external object),
during execution of current call

Our Approach!

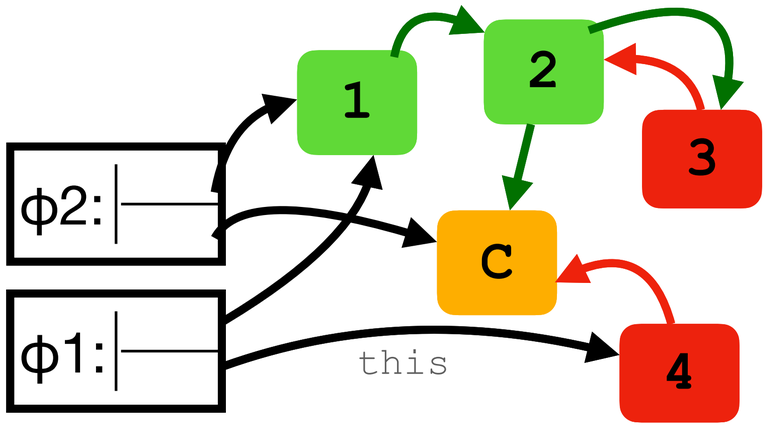


Remit_1_a : Express/Meaning of: No *currentlly accessible* external object has direct accees to o

Protection increases as we push frames

Def: o *protected*

$$\langle\!\langle o \rangle\!\rangle \triangleq \forall\, o'.[\ o' \text{ extl} \wedge o' \text{ reachble from top frame} \Rightarrow \forall f.[\ o''.f \neq o\ ]\ ]$$
$$\wedge\ [\ \texttt{this}\ \text{extl} \Rightarrow o \text{ not an arg}\ ]$$



$\Phi_1 \not\models \langle\!\langle C \rangle\!\rangle$
$\Phi_1 \not\models \langle\!\langle 2 \rangle\!\rangle$
$\Phi_1 \not\models \langle\!\langle 1 \rangle\!\rangle$

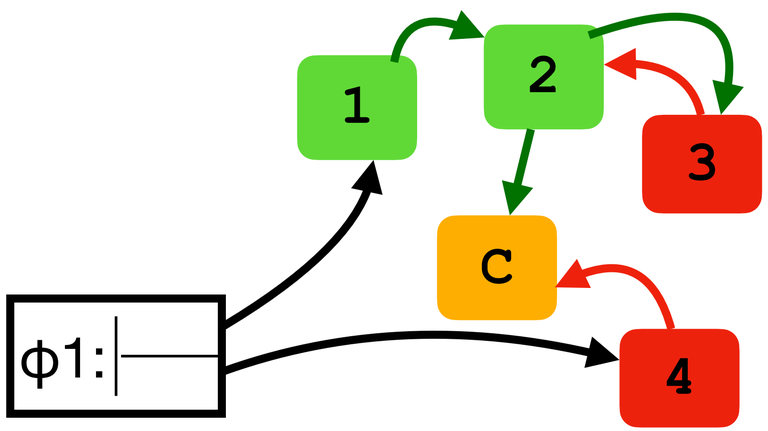
$\Phi_1\Phi_2 \models \langle\!\langle C \rangle\!\rangle$
$\Phi_1\Phi_2 \not\models \langle\!\langle 2 \rangle\!\rangle$
$\Phi_1\Phi_2 \models \langle\!\langle 1 \rangle\!\rangle$

Protection is “relative” to top frame

We also define …

Def: o *protected from* o'

$$\langle\!\langle o \rangle\!\rangle \not\leftarrow o' \quad \triangleq \quad \forall o''.[\ \ o'' \text{ extl} \wedge o'' \text{ reachble from } o' \ \Rightarrow \ \ \forall f.[\ o''.f \neq o\ ]\ \ ]$$



… $\models \langle\!\langle 2 \rangle\!\rangle \not\leftarrow 1$

… $\not\models \langle\!\langle 2 \rangle\!\rangle \not\leftarrow 3$

… $\models \langle\!\langle C \rangle\!\rangle \not\leftarrow 1$

… $\models \langle\!\langle C \rangle\!\rangle \not\leftarrow 3$

… $\not\models \langle\!\langle C \rangle\!\rangle \not\leftarrow 2$

Assertions may talk of protection and externals

$$A \quad ::= \quad e \mid e : C \mid \neg A \mid A \wedge A \mid \forall x : C.A \mid e : \texttt{extl} \mid \langle\!\langle e \rangle\!\rangle \nrightarrow e \mid \langle e \rangle$$

Remit_1_b : Express/meaning "invariant"

We propose "*scoped* invariants", of the form $\forall x_1{:}C_1,\ldots x_n{:}C_n\ \{\ A\ \}$

Eg $\forall$ `s:Shop.` { `s.account` $\neq$ `null` $\rightarrow$ « `s.account` key »

Any number of execution steps,
before returning from current frame

Definition

$$M \vDash \forall x_1{:}C_1,\ldots x_n{:}C_n\ \{\ A\ \} \triangleq \forall M'.\ \forall \sigma, \sigma'.\ \forall \alpha_1,\ldots\alpha_n.[$$

$$M, \sigma \vDash \texttt{this}\texttt{:}\ \text{ext} \wedge \alpha_1 :C_1,\ldots\alpha_n{:}Cn \wedge A[\alpha_1,\ldots\alpha_n / x_1,..x_n]$$

$$\wedge \quad M' \bullet M, \sigma \rightsquigarrow^* \sigma'$$

$$\wedge \quad M, \sigma' \vDash \texttt{this}\texttt{:}\ \text{ext}$$

$$\Rightarrow$$

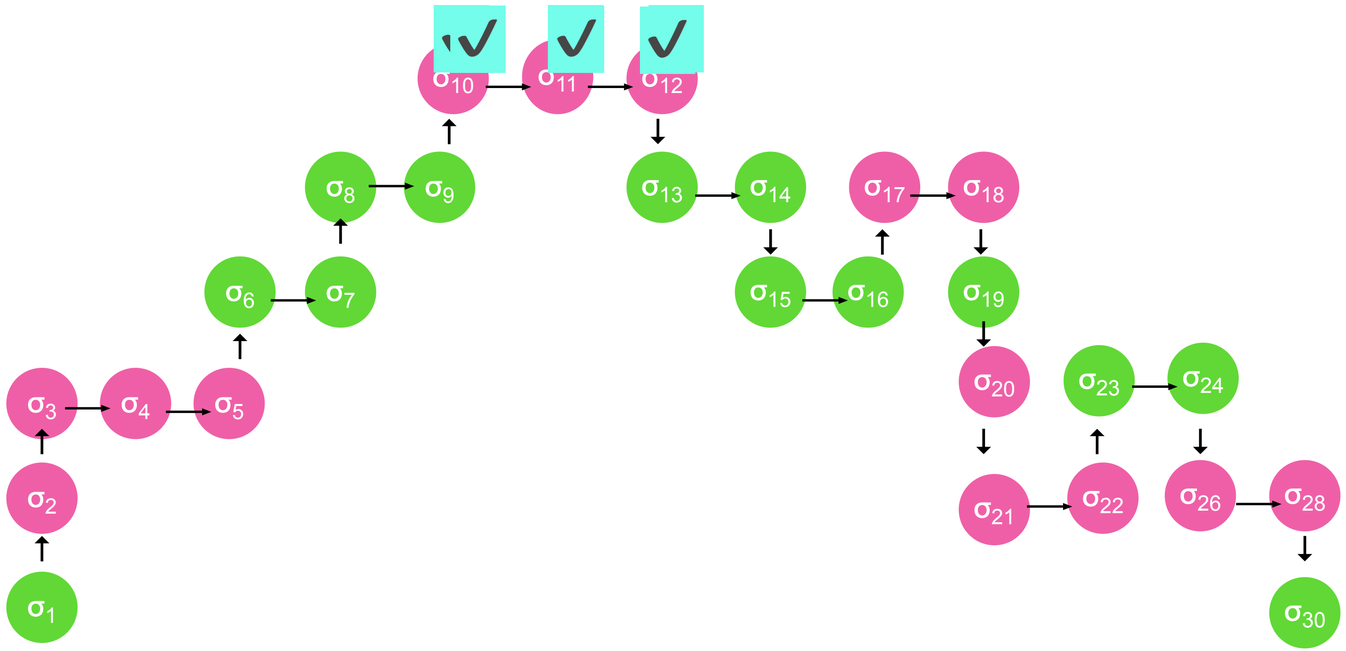
$$M, \sigma' \vDash A[\alpha_1,\ldots\alpha_n / x_1,..x_n]$$

$$]$$

An example:

Consider call graph below, with green disks for internal states (eg σ_2),
and pink disks for external states (eg σ_{28}).

$M \models \forall x{:}C.\ \{\ A\ \}$ means that $M, \sigma_{10} \models \alpha{:}C \wedge A[\alpha/x]$ implies $M, \sigma_{11} \models A[\alpha/x]$
$M, \sigma_{12} \models A[\alpha/x]$

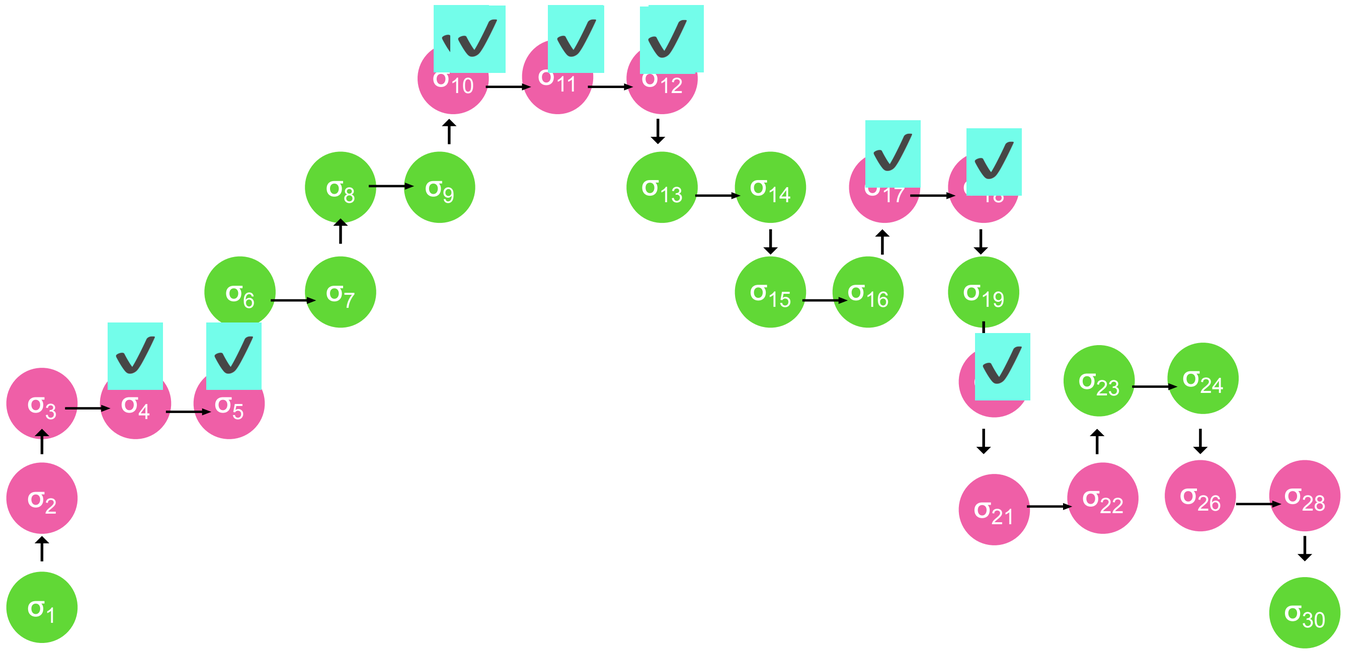


An example:

Consider call graph below, with green disks for internal states (eg σ_2),
and pink disks for external states (eg σ_{28}).

$M \models \forall x{:}C.\ \{\ A\ \}$ means that $M, \sigma_4 \models \alpha{:}C \wedge A[\alpha/x]$ implies

$M, \sigma_5 \models A[\alpha/x]$
$M, \sigma_6 \models A[\alpha/x]$
$M, \sigma_{10} \models A[\alpha/x]$
$M, \sigma_{10} \models A[\alpha/x]$
$M, \sigma_{11} \models A[\alpha/x]$
$M, \sigma_{17} \models A[\alpha/x]$
$M, \sigma_{18} \models A[\alpha/x]$
$M, \sigma_{20} \models A[\alpha/x]$



Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{fine} \models S$

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

Talk about effects

Talk about emergent behaviour

S1 ≜ ∀ a:Account. { ⟪a⟫ }

S2 ≜ ∀ a:Account. { ⟪a.key ⟫ }

S4 ≜ ∀ a:Account, b:Num. { ⟪a.key ⟫ ∧ a.blnce ≥ b }

S5 ≜ { ⟪this.accnt.key ⟫ ↩ buyer ∧ this.accnt.blnce =b }
Shop::buy(buyer:external, item:Item)
{ this.accnt.blnce ≥ b }

$M_{bad} \not\models S2$

$M_{bad} \not\models S4$

$M_{bad} \not\models S1$

$M_{good} \models S2 \wedge S4 \wedge S5$

$M_{good} \not\models S1$

$M_{fine} \models S2 \wedge S4 \wedge S5$

$M_{fine} \not\models S1$

Remember Parity MultiSig Wallet Hack

Parity MultiSig Wallet Hack (150,000 ETH (~30M USD) 2017)

1) Attacker calls `initWallet(…)` and makes themselves the single owner
2) Attacker calls `execute(…)` and withdrawns all funds

An implementation that satisfies the below avoids the hack (assuming [illegible] is governed by the votes of the owners}

S10 ≜ ∀ msig:Multisig, o:Address.{ o ∈ msig.owner }

S11 ≜ ∀ msig:Multisig, o:Address { o ∈ msig.owner ∧ ⟪o⟫ }

S12 ≜ ∀ msig:Multisig, o:Address, v:Vote { o ∈ msig.owner ∧ ⟪o⟫ ∧ msig.

S13 ≜ ∀ msig:Multisig, o:Address, f:Funds

{ o ∈ msig.owner ∧ ⟪o⟫ ∧ msig.votes(o)=NO ∧ msig.funds = f }

The owner set only grows

No owner is leaked

Nobody can vote on behalf of others

No change of funds unless all owners agreed

Challenge_2: An inference system, such that

$$M_{good} \vdash S$$
$$M_{bad} \nvdash S$$
$$M_{fine} \vdash S$$

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 …

Three Stages

Assume an underlying 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 $M \vdash \{A\}\, s \,\{A'\}$

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 extend some underlying Hoare logic to a Hoare logic of triples with usual meaning

$$\textsc{extend}\ \frac{M \vdash_{ul} \{ A \}\ s\ \{ A' \} \quad s \text{ contains no method call}}{M \vdash \{A\}\ s\ \{A'\}}$$

$$\textsc{types-1}\ \frac{s \text{ contains no method call}}{M \vdash \{x : C\}\ s\ \{x : C\}}$$

Challenge_2: An inference system, such that …

2nd stage

We expand triples to quadruples

INV

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

TYPES-2

$$\frac{M \vdash \{A\}\, s\, \{A'\} \parallel \{A''\}}{M \vdash \{x : C \wedge A\}\, s\, \{x : C \wedge A'\} \parallel \{A''\}}$$

COMBINE

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

SEQU

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

CONSEQU

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

Challenge_2: An inference system, such that …

2nd stage

We introduce triples for protection

$$\frac{u \overset{\text{txt}}{\neq} x}{M \vdash \{\, true \,\}\ u = \texttt{new}\ C\ \{\, \langle u \rangle \ \wedge \langle u \rangle \leftrightarrow\!\!\!\!\times\ x \,\}} \quad [\textsc{prot-new}]$$

$$\frac{M \vdash \{\, e = z \,\}\ stmt\ \{\, e = z \,\}}{M \vdash \{\ \langle e \rangle\ \}\ stmt\ \{\ \langle e \rangle\ \}} \quad [\textsc{prot-1}]$$

$stmt$ is free of method cals, or assignment to z

$$\frac{M \vdash \{\ z = e \wedge z' = e'\ \}\ stmt\{\ z = e \wedge z' = e'\ \}}{M \vdash \{\ \langle e \rangle \leftrightarrow\!\!\!\!\times\ e'\ \}\ stmt\ \{\ \langle e \rangle \leftrightarrow\!\!\!\!\times\ e'\ \}} \quad [\textsc{prot-2}]$$

$stmt$ is either $x := y$ or $x := y.f$, and $z, z' \overset{\text{txt}}{\neq} x$

$$\frac{x \overset{\text{txt}}{\neq} z}{M \vdash \{\ \langle y.f \rangle \leftrightarrow\!\!\!\!\times\ z\ \}\ x = y.f\ \{\ \langle x \rangle \leftrightarrow\!\!\!\!\times\ z\ \}} \quad [\textsc{prot-3}]$$

$$\frac{}{M \vdash \{\ \langle x \rangle \leftrightarrow\!\!\!\!\times\ z\ \wedge\ \langle x \rangle \leftrightarrow\!\!\!\!\times\ y'\ \}\ y.f = y'\ \{\ \langle x \rangle \leftrightarrow\!\!\!\!\times\ z\ \}} \quad [\textsc{prot-4}]$$

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 …

Protection is “relative” to a frame;
Our $-\!\!\triangledown$ operator

helps us switch to callee’s view

INVARIANT

$$\frac{M \vdash Encps(\overline{x : C} \wedge A) \quad \forall D, m : \texttt{mBody}(m, D, M) = \texttt{public}\ (\overline{y : D})\{\ stmt\ \} \implies M \vdash \{\ \texttt{this} : \texttt{D}, \overline{y : D},\ \overline{x : C} \wedge A \wedge A \text{-}\!\!\triangledown (\texttt{this}, \overline{y})\ \}\ stmt\ \{\ A \wedge A \text{-}\!\!\triangledown \texttt{res}\ \} \parallel \{\ A\ \}}{M \vdash \forall \overline{x : C}.\{A\}}$$

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

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

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

Challenge_4a: From Caller to Callee

We consider `Shop`'s method pay, and want to prove the external call, ie

$$\{\ \texttt{buyer:extl} \wedge \langle\!\langle \texttt{this.accnt.key} \rangle\!\rangle\!\!\nleftrightarrow \texttt{buyer} \wedge \texttt{this.accnt.blnce} = b\ \}$$

```
buyer.pay(this.accnt,price)
```

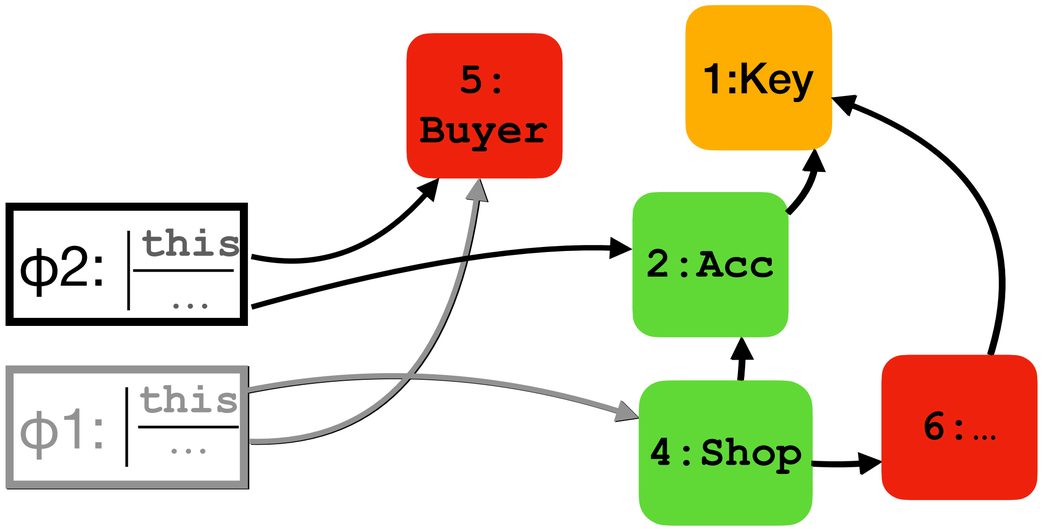
$\{\ \texttt{this.accnt.blnce} \geq b\ \}$ ...

We want to use S4, ie $\forall$ a:Account, b:Num. { ⟪a.key ⟫ ∧ a.blnce ≥ b }

BUT, $\phi_1 \not\models \langle\!\langle 1 \rangle\!\rangle$ 🤦

AHA!! to use S4, we only need ϕ_1,**callee** $\models \langle\!\langle 1 \rangle\!\rangle$ 😅

Indeed, $\phi_1\,\phi_2 \models \langle\!\langle 1 \rangle\!\rangle$ 😁



Therefore, we need an operator which mediates asserions between viewpoint of the callee and viewpoint of caller.

Challenge_4a: From Caller to Callee — The $-\triangledown$ operator

$\triangledown$ translates an assertion from the view of the callee to that of the caller.

Definition 6.1. [The $-\triangledown$ operator]

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

Example: ⟪this.accnt.key⟫ $-\triangledown$ buyer = ⟪this.accnt.key⟫ $+$ buyer

Lemma 6.2. For states σ, assertions A, so that $Stb^{+}(A)$ and $Fv(A) = \emptyset$, frame ϕ, variables $y_0, \overline{y}$:

$$
\begin{array}{lcl}
(2)\ M, \sigma \models A -\triangledown Rng(\phi) & \Longrightarrow & M, \sigma \triangledown \phi \models A \\
(3)\ M, \sigma \triangledown \phi \models A \wedge \texttt{extl} & \Longrightarrow & M, \sigma \models A -\triangledown Rng(\phi)
\end{array}
$$

Challenge_4a: From Caller to Callee — The $-\triangledown$ operator

$\triangledown$ translates

Protection is "relative" to a frame;

$\triangledown$ operator switches assertion to callee's viewpoint

$$(e : \texttt{extl})-\triangledown\overline{y} \triangleq e : \texttt{extl} \qquad (\neg A)-\triangledown\overline{y} \triangleq \neg(A-\triangledown\overline{y})$$
$$e-\triangledown\overline{y} \triangleq e \qquad (e : C)-\triangledown\overline{y} \triangleq e : C$$

Example: ⟪this.accnt.key⟫ $-\triangledown$ buyer = ⟪this.accnt.key⟫ $+$ buyer

Lemma 6.2. For states σ, assertions A, so that $Stb^{+}(A)$ and $Fv(A) = \emptyset$, frame ϕ, variables $y_0, \overline{y}$:

(2) $M, \sigma \models A-\triangledown Rng(\phi) \implies M, \sigma \triangledown \phi \models A$

(3) $M, \sigma \triangledown \phi \models A \wedge \texttt{extl} \implies M, \sigma \models A-\triangledown Rng(\phi)$

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

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

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

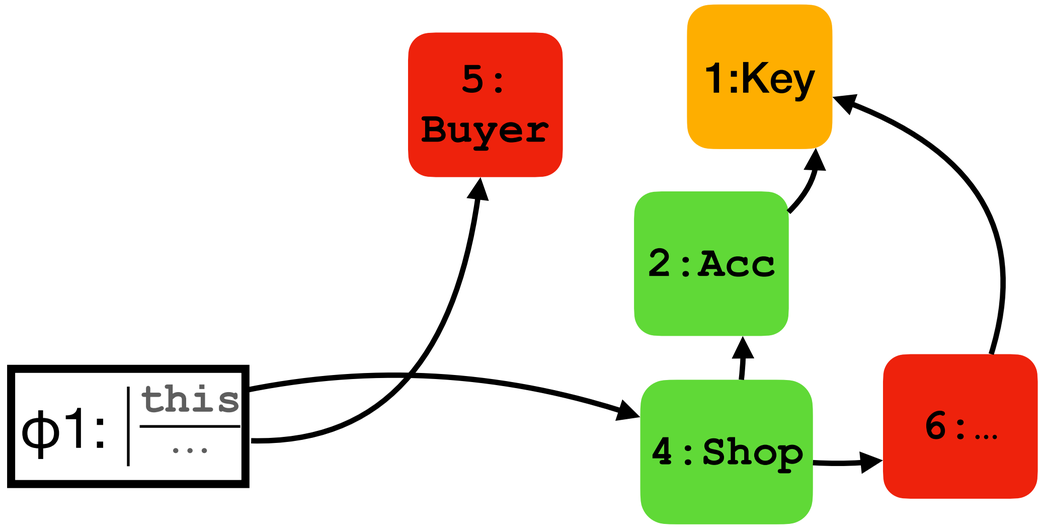
$$\frac{\vdash \; M \; : \; \forall \overline{x : D}\{A\}}{M \vdash \{ \; y_0 : \texttt{ext} \wedge \overline{x : D} \wedge A \multimap \forall \overline{y} \; \} \, u := y_0.m(y_1, ..y_n) \, \{ \; A \multimap \forall \overline{y} \; \} \; \| \; \{ \; A \; \}} \; [\textsc{Call\_Ext}]$$

Using [Call_Ext] we can, indeed, prove

$$\{\ \mathtt{buyer}:\mathtt{extl} \wedge \langle\!\langle \mathtt{this.accnt.key} \rangle\!\rangle \leftrightarrow \mathtt{buyer} \wedge \mathtt{this.accnt.blnce} = b\ \}$$

$$\mathtt{buyer.pay(this.accnt,price)}$$

$$\{\ \mathtt{this.accnt.blnce} \geq b\ \} \ldots$$



Using our quadruples, we have proven

$$M_{good} \vdash S2 \wedge S4 \wedge S5$$

$$M_{fine} \vdash S2 \wedge S4 \wedge S5$$

Moreover, we have proven

$$\vdash M \wedge M \vdash \{A\}\ stmt\ \{A'\} \,||\, \{A''\} \Rightarrow M \models \{A\}\ stmt\ \{A'\} \,||\, \{A''\}$$

$$\vdash M \Rightarrow M \models S$$

Summary

- Distinction between external/internal objects
- ⟪e ⟫: expresses that e is protected from reachable external objects
- Specifications talk about necessary conditions for effect:

  ∀ x: … { ⟪e ⟫ ∧ A }

  means that A is preserved as long as **capability** e is ptotected
- API-agrnositc spec,
- “Algorithmic” inference system system,
- Reason with open calls
- Protaction, ⟪e ⟫ relative to frame. Use —▽ to switch view

Surprises

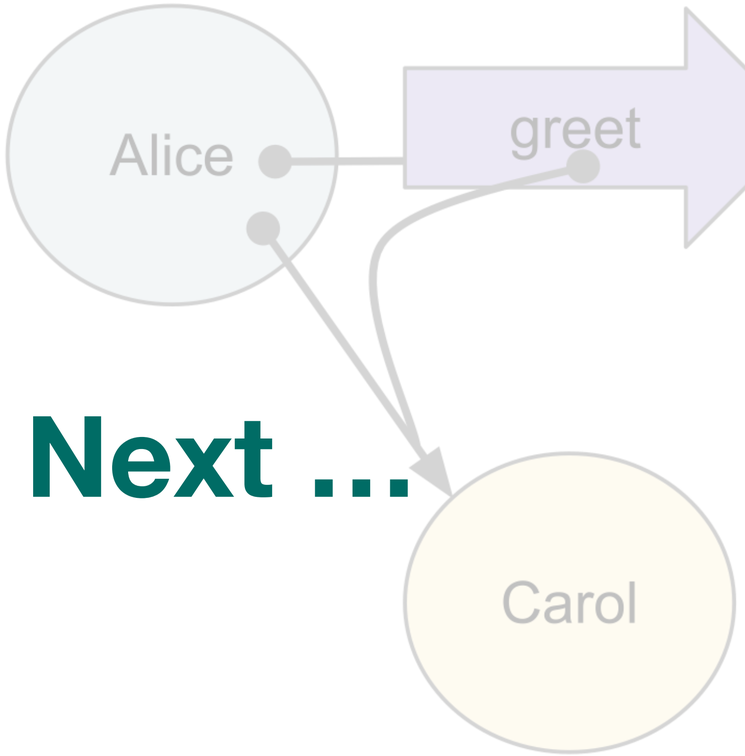
- Frame-related concepts
  - Protected object
  - Scoped Invarinats
  - ▽ to switch view to callee frame
- Started with *necessary conditions*,
  but ended up using *sufficient conditions* to reason about them
- Started with temporal logics,
  but ended up using invariants
- Hoare logic extensions
- Invariants — twenty years later …

A Unified Framework for Verification Techniques for Object Invariants

S. Drossopoulou[1], A. Francalanza[2], P. Müller[3], and A. J. Summers[1]

[1] Imperial College London,
[2] University of Southampton,
[3] Microsoft Research, Redmond

Abstract. Object invariants define the consistency of objects. They have subtle semantics because of call-backs, multi-object invariants and subclassing. Several visible-state verification techniques for object in-



Next …

- Modules
- Mechanize proofs
- Completeness?
- Revisit protection:
  - What if more than one capability for an effect?
  - Ownership types, membranes etc?
  - Instance-level protection?
  - assertions rather than objects to protect
- Other programming Paradigms (Ethereum, ECMAscript)
- Better interaction with underlying Hoare logics
- Tool

Thank You!

by the end of next week