

Holistic Specifications for Robust Code

Sophia Drossopoulou
Imperial College London and MSR Cambridge

in collaboration with James Noble and Julian Maccay (VU Wellington), Toby Murray (Uni Melbourne) , Mark Miller (Agorics), and Susan Eisenbach, Shupeng Loh and Emil Klasan (Imperial)



Holistic Specifications for Robust Code

with support from

- New Zealand Marsden grants
- Ethereum Foundation
- Facebook
- Agoric
- copious “free time”



Today

- Traditional Specifications do not adequately address Robustness
- Holistic Specifications — Summary and by Example
- Holistic Specification Semantics

Today

- **Traditional Specifications do not adequately address Robustness**
- Holistic Specifications — Summary and Examples
- Holistic Specification Semantics

Traditional Specs

- *closed* world
- *sufficient* conditions for some action/effect
- talk of *individual* function

Robustness considerations

- *open* world
- *necessary* conditions for some action/effect
- *emergent* behaviour

An Example



```

class SafeOne{
    private fld secret;
    private fld treasure;

    mthd take(s) {
        if (s==secret)
        { t=treasure;
          treasure = null;
          return t; } }

    constr SafeOne(s,t) {
        secret=s; treasure = t; }
    }
}

```

```

class SafeTwo{
    private fld secret;
    private fld treasure;

    mthd take(s) {
        ... }

    constr SafeTwo(s,t) {
        ... }

    mthd set(s)
    { this.secret=s; }
}

```

```

class SafeThree{
    private fld secret;
    private fld treasure;

    mthd take(s) {
        ... }

    constr SafeThree(s,t) {
        ... }

    mthd set(s,sold)
    { if sold==secret then
      { this.secret=s; } }
}

```

SpecA:

```

{ true }
    sf.take(s)
{ s=sf.secret -> sf.treasurepost = null
  ^
  s≠sf.secret -> sf.treasurepre, sf.treasurepost }

```

traditional

traditional

SpecB: $\forall sf:Safe. [sf.secret_{pre} = sf.secret_{post}]$

$SafeOne \models SpecA$

SafeTwo $\models SpecA$

SafeTree $\models SpecA$

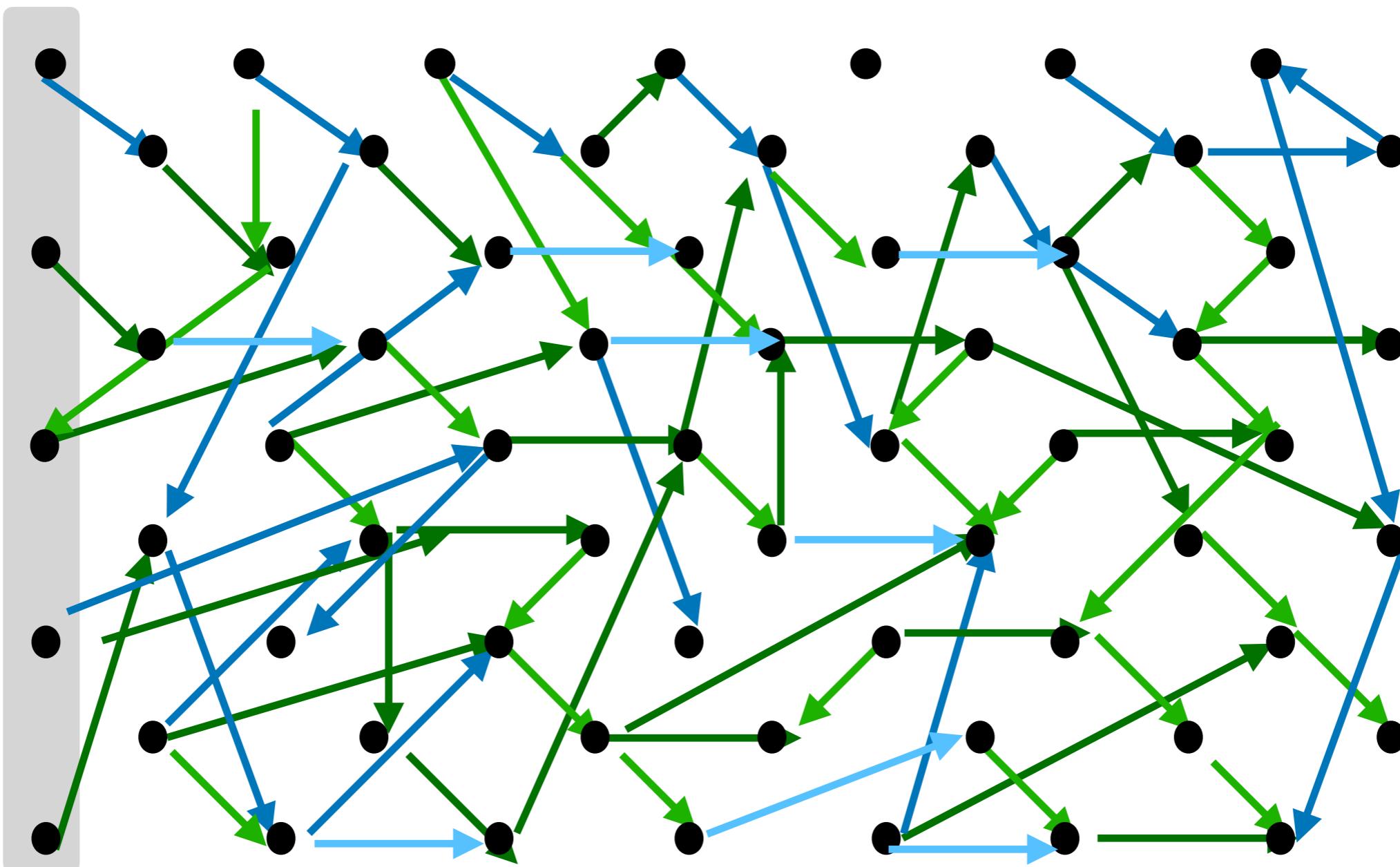
$SafeOne \models SpecB$

SafeTwo $\not\models SpecB$

SafeThree $\not\models SpecB$

traditional specs do *not* express robustness!

traditional



```

class SafeOne{
  private fld secret;
  private fld treasure;

  mthd take(s) {
    if (s==secret)
    { t=treasure;
      treasure = null;
      return t; } }

  constr SafeOne(s,t) {
    secret=s; treasure = t; }

}

```

```

class SafeTwo{
  private fld secret;
  private fld treasure;

  mthd take(s) {
    ... }

  constr SafeTwo(s,t) {
    ... }

  mthd set(s)
  { this.secret=s; }

}

```

```

class SafeThree{
  private fld secret;
  private fld treasure;

  mthd take(s) {
    ... }

  constr SafeThree(s,t) {
    ... }

  mthd set(s,sold)
  { if sold==secret then
    { this.secret=s; } }

}

```

SpecA:

```

{ true }
  sf.take(s)
{ s=sf.secret -> sf.treasurepost = null
  ^
  s≠sf.secret -> sf.treasurepost = sf.treasurepre }

```

SafeOne \models SpecA
SafeTwo \models SpecA
SafeThree \models SpecA

SpecB: $\forall sf:Safe. [sf.secret_{pre} = sf.secret_{post}]$

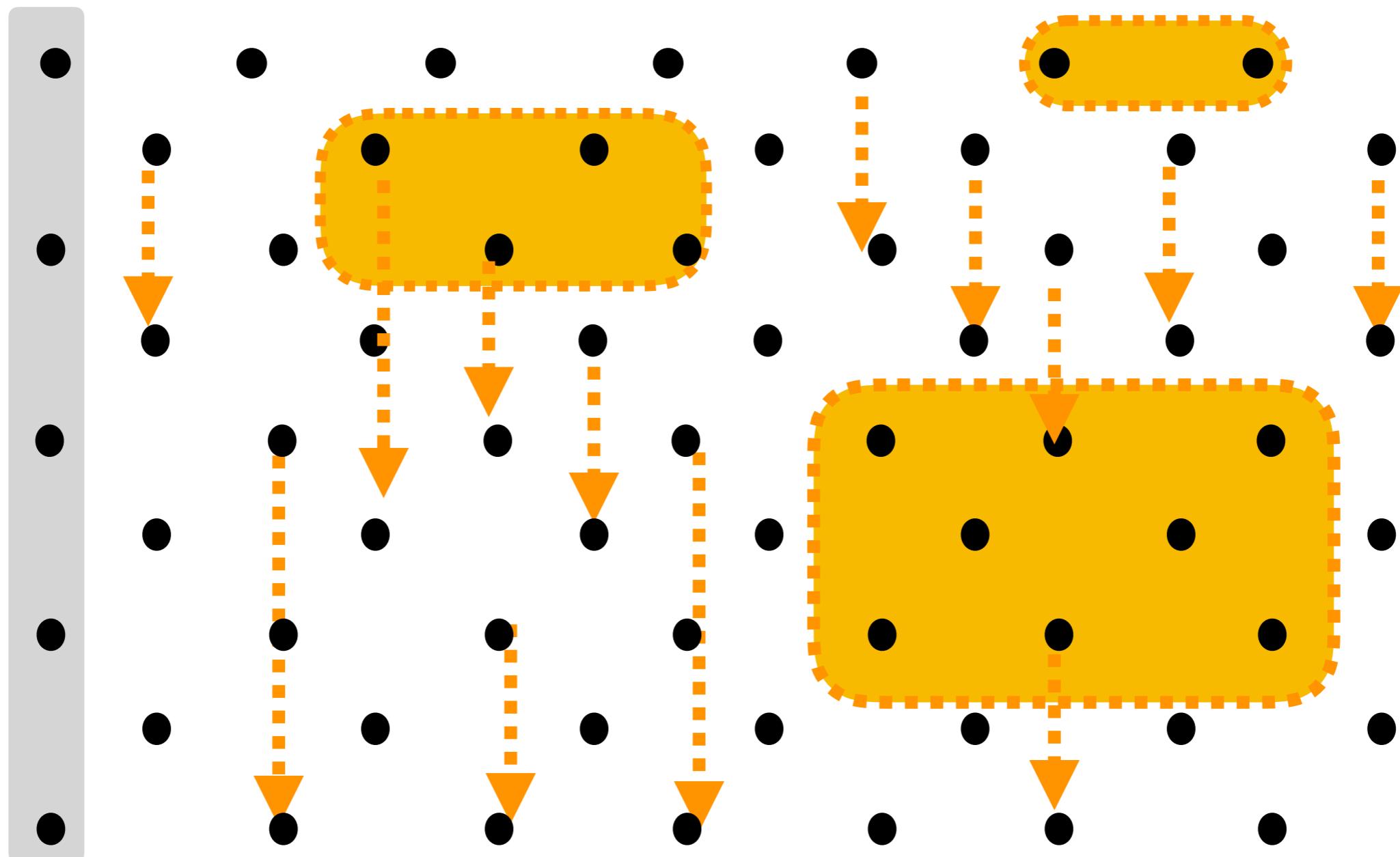
SafeOne \models SpecB
SafeTwo $\not\models$ SpecB
SafeThree $\not\models$ SpecB

SpecC: $\forall sf:Safe. [\langle \mathbf{will<Changes<} sf.treasure \rangle \rangle$
 \rightarrow
 $\exists o. [\mathbf{External} \langle o \rangle \wedge \langle o \mathbf{Access} sf.secret \rangle]$

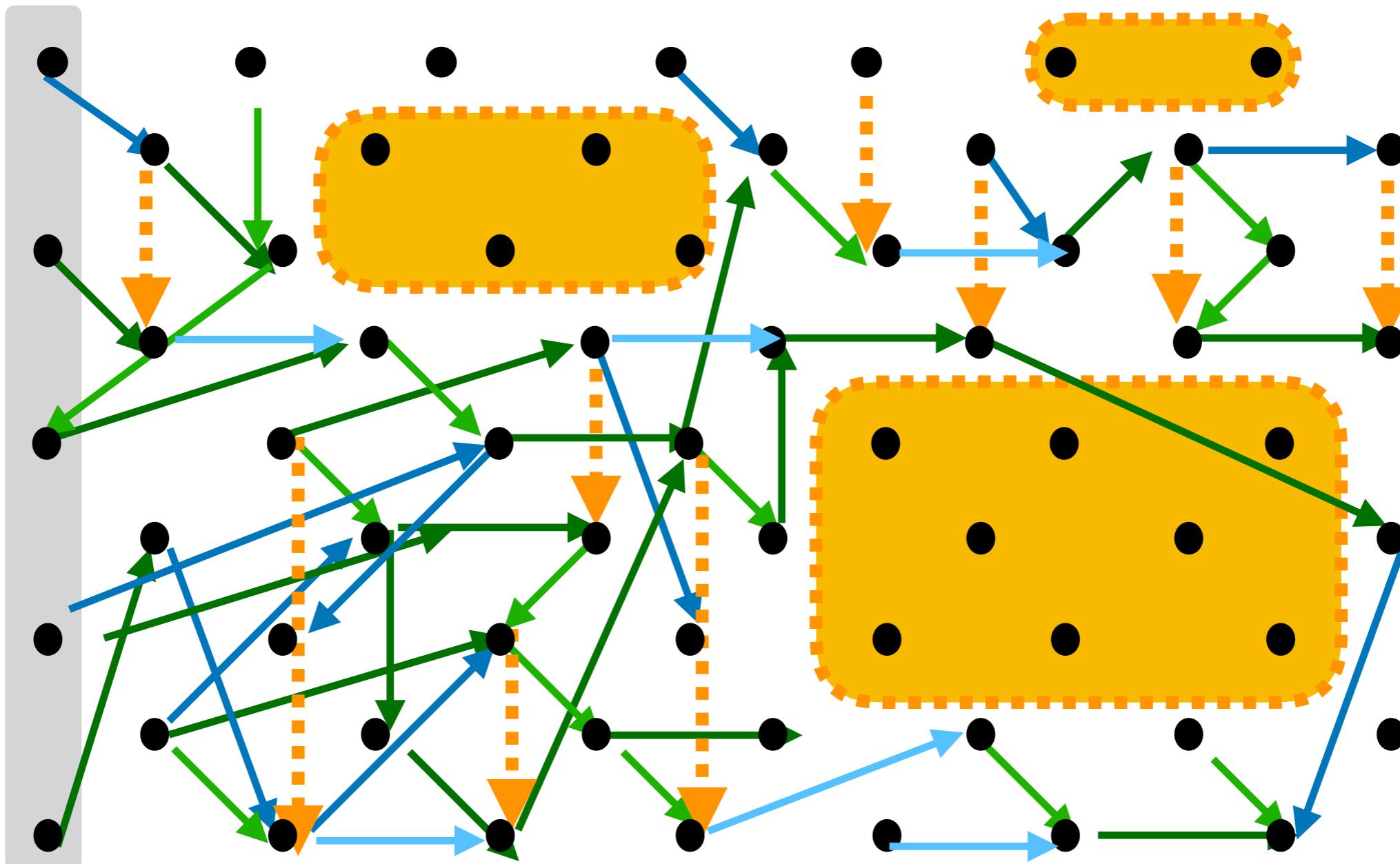


SafeOne \models SpecC
SafeTwo $\not\models$ SpecC
SafeThree \models SpecC

holistic



traditional + holistic



Today

- Traditional Specifications do not adequately address Robustness
- **Holistic Specifications — Summary Examples**
- Holistic Specification Semantics

Holistic Assertions – summary

$e ::= \text{this} \mid x \mid e.\text{fld} \mid \dots$

$A ::= e>e \mid e=e \mid \dots$

$\mid A \rightarrow A \mid A \wedge A \mid \exists x. A \mid \dots$

$\mid \mathbf{Access}(e,e')$ permission

$\mid \mathbf{Changes}(e)$ authority

$\mid \mathbf{Will}(A) \mid \mathbf{Was}(A) \mid \mathbf{Prev}(A) \mid \mathbf{Next}(A)$ time

$\mid A \mathbf{in} S \mid \mathbf{External}(e)$ space

$\mid x.\mathbf{Calls}(y,m,z_1,\dots,z_n)$ control

$\mid x \mathbf{obeys} A$ trust

Holistic Assertions — examples

- ERC20
- DAO
- DOM attenuation
- Bank & Account
- Escrow

Example1: ERC20

a popular standard for initial coin offerings;

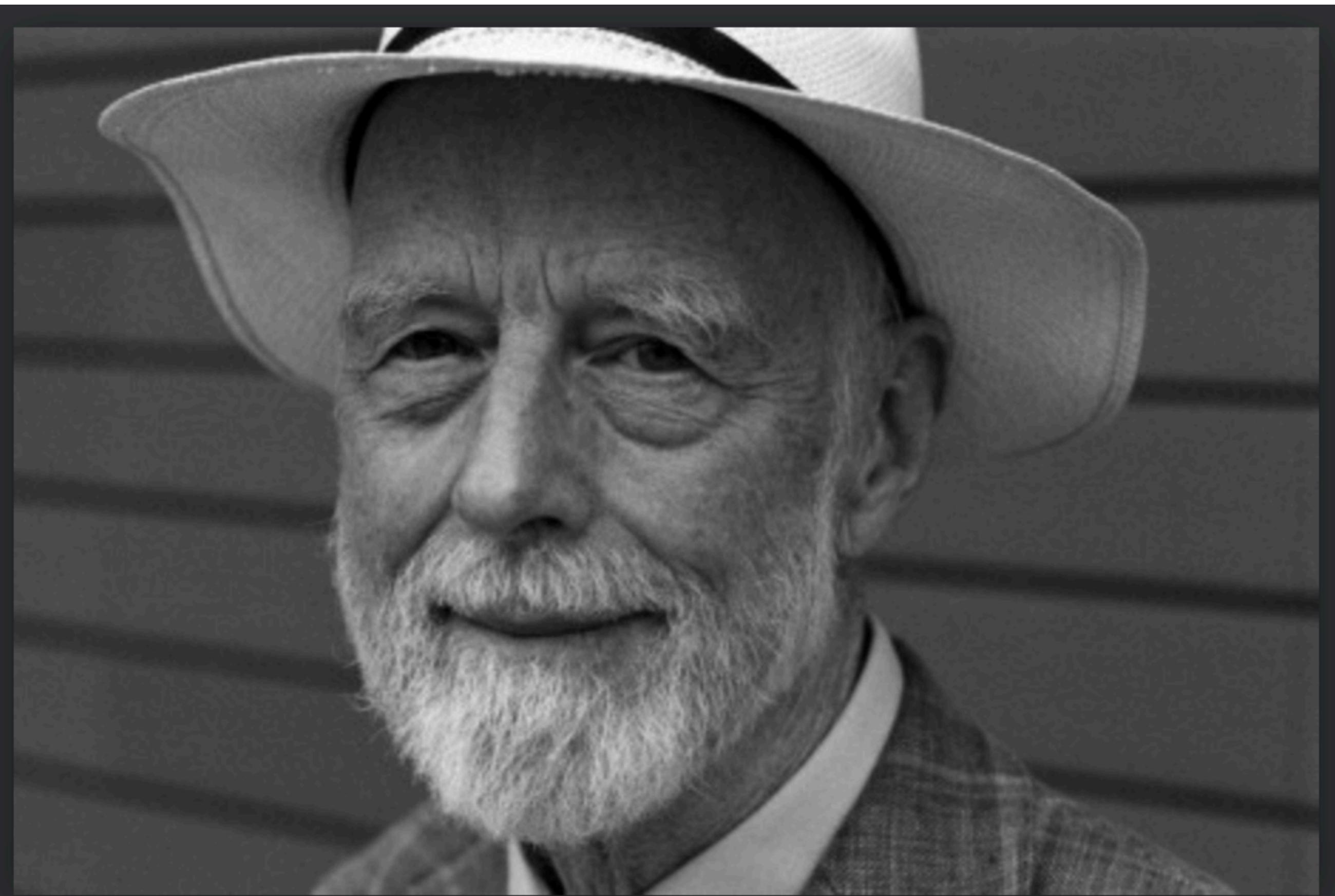
clients can buy and transfer tokens, and can designate other clients to transfer on their behalf.

In particular, a client may call

- transfer: transfer some of her tokens to another clients,
- approve: authorise another client to transfer some of her tokens on her behalf.
- transferFrom: cause another client's tokens to be transferred

Moreover, ERC20 keeps for each client

- balance the number of tokens she owns



ERC20 classical spec - transfer

For any ERC20 contract e , and different clients c_1, c_2 .

c_1 's balance is larger than m .

{ c_1 calls $e.\text{transfer}(c_2, m)$ }

c_1 's balance decreases by m , and c_2 's balance increases by m .

$e:\text{ERC20} \wedge \text{this} = c_1 \neq c_2 \wedge e.\text{balance}(c_1) > m$

{ $e.\text{transfer}(c_2, m)$ }

$e.\text{balance}(c_1) = e.\text{balance}(c_1)_{\text{pre}} - m \wedge e.\text{balance}(c_2) = e.\text{balance}(c_2)_{\text{pre}} + m$

sufficient condition

effect

ERC20 classical spec - transfer - 2

What if c_1 's balance not large enough?

```
e:ERC20 ∧ this = c ∧ e.balance(c1) < m  
    { e.transfer(c2, m) }  
∀ c. e.balance(c) = e.balance(c)pre
```

ERC20 classical spec
- what about the other functions?

ERC20 classical spec

Is that robust?

```
e..  
^ e.balance(c1) ≥ m  
{ e.transferFrom(c1, c2, m)  
e.balance(c1) = e.balance(c1) - m  
e.balance(c2) = e.balance(c2) + m  
}
```

a “super-client,”
authorised on all?

a function that
takes 0.5% from
each account?

sufficient
for change of balance



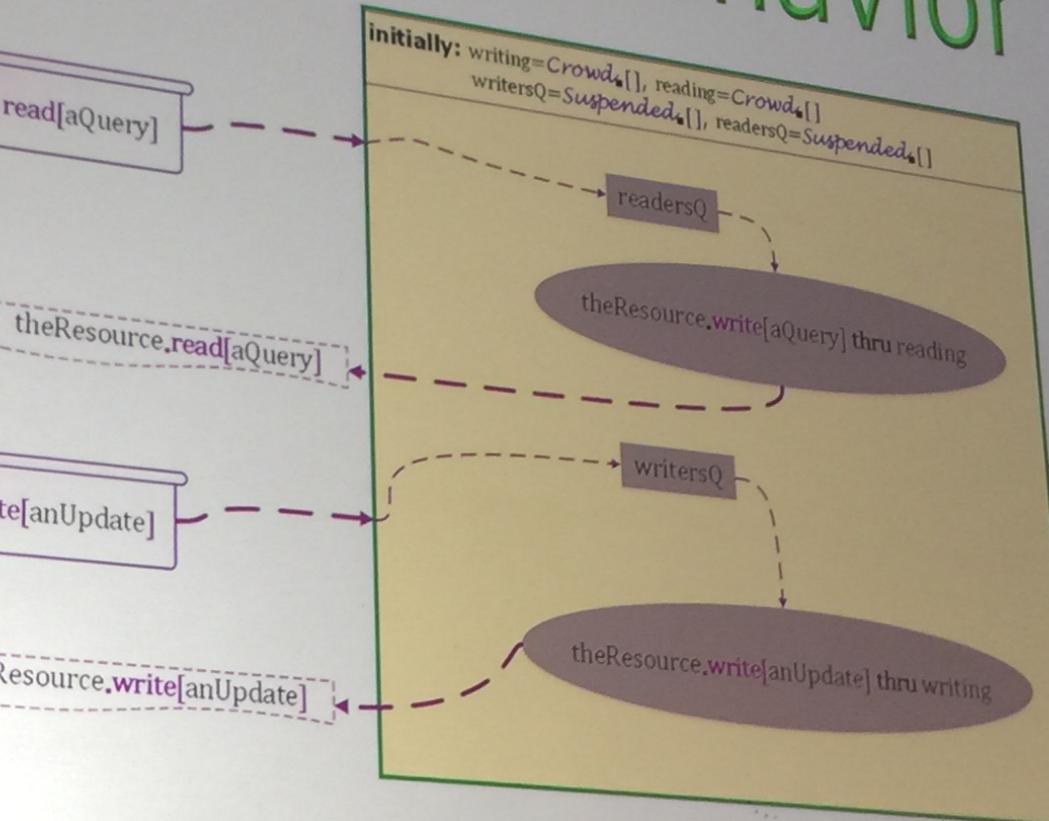
I am worried
about who/what can
reduce my balance

can authority increase?

```
and (c) pre ^  
c2, m, → e.Authorized(c1, c2, m) ]
```

```
totalSupply() }  
... { e.allowanceOf(c2) } ...  
... { e.balanceOf(c) } ...
```

Invariant Behavior



holistic specs - invariants ++

- state
- **time**
- **space**
- **control**
- **permission**
- **authority**
- **(when/where do they hold)**

holistic spec - reduce balance

$\forall e:\text{ERC20}. \forall c1:\text{Client}. \forall m:\text{Nat}.$

[$e.\text{balance}(c1) = \text{Prev}(e.\text{balance}(c1)) - m$

($\exists c2, c3: \text{Client}.$

$\text{Prev}(\ c1.\text{Calls}(e, \text{transfer}, c2, m) \)$

\vee

$\text{Prev}(\ e.\text{Authorized}(c1, c2, m) \wedge$
 $c2.\text{Calls}(e, \text{transferFrom}, c1, c3, m) \) \]$

effect

necessary
condition

A client's balance decreases *only if* that client, or somebody authorised by that client, made a payment.

holistic spec - authority

effect

$e.\text{Authorized}(c1, c2, m) \doteq$

$\text{Prev}(c1.\text{Calls}(e, \text{approve}, c2, m))$

\vee

$\text{Prev}(e.\text{Authorized}(c1, c2, m+m') \wedge c2.\text{Calls}(e, \text{transferFrom}, c1, _, m'))$

\vee

$\text{Prev}(e.\text{Authorized}(c1, c2, m) \wedge \neg c2.\text{Calls}(e, \text{transferFrom}, c1, _, _))$

necessary
conditions

$c2$ is authorised by $c1$ for m iff

in previous step $c1$ informed e that it authorised $c2$ for m
or

in previous step $c2$ was authorised for $m+m'$ and spent m' for $c1$
or

in previous step $c2$ was authorised for m and did not spend $c1$

classical vs holistic

e:ERC20 \wedge e.balance(cl) >m \wedge e.balance(cl') = m' \wedge cl ≠ cl'
 { e.transfer(cl',m) \wedge Caller=cl }

e.balance(cl) = e.balance(cl)_{pre} -m \wedge e.balance(cl')_{pre} = m'+m

e:ERC20 \wedge e.balance(cl) >m \wedge e.balance(cl') = m' \wedge cl ≠ cl'
 \wedge Authorized(e, cl, cl")
 { e.transferFrom(cl',m) \wedge Caller=cl"}

e.balance(cl) = e.balance(cl)_{pre} -m \wedge e.balance(cl')_{pre} = m'+m

e:ERC20 \wedge e.balance(cl) >m \wedge e.balance(cl') = m'
 { e.allow(cl') \wedge Caller=cl }
 Authorized(e, cl, cl")

another 7 specs

....

....

Classical

- per function; *sufficient* conditions for some action/ effect
- *explicit* about individual function, and *implicit* about emergent behaviour

Holistic

- *necessary* conditions for some action/effect
- *explicit* about emergent behaviour

$\forall e:\text{ERC20}. \forall \text{cl: Client}.$
 [e.balance(cl) = **Prev**(e.balance(cl)) - m]
 \rightarrow
 ($\exists \text{cl}', \text{cl}'' : \text{Client}.$
Prev (cl.**Calls**(e.transfer(cl',m)))
 \vee
Prev (Authorized(e, cl, cl") \wedge cl".**Calls**(e.transferFrom(cl, cl', m)))]

Authorized(cl, cl') \triangleq $\exists m : \text{Nat}.$ **Prev**(cl.**Calls**(e.approve(cl', m)))

Example2: DAO simplified

DAO, a “hub that disperses funds”; (<https://www.ethereum.org/dao>).

... clients may contribute and retrieve funds :

- payIn (m) pays into DAO m on behalf of client
- repay () withdraws all moneys from DAO

Vulnerability: Through a buggy version of repay (), a client could re-enter the call and deplete all funds of the DAO.

classical spec

Assuming DAO keeps a directory of contributions, and require:

R1: directory is compatible with the amount of ether kept in the DAO, and

R2: that withdraw reduces the ether by that amount.

R1: $\forall d:\text{DAO}. \ d.\text{ether} = \sum_{c1 \in \text{dom}(d.\text{directory})} d.\text{directory}(c1)$

R2: $d:\text{DAO} \wedge n:\text{Nat} \wedge d.\text{directory}(cl)=n > 0 \wedge \text{this}=cl$
 $\{ d.\text{repay}() \}$

$d.\text{directory}(cl)=0 \wedge d.\text{Calls}(cl,\text{send},n)$

$d:\text{DAO} \wedge n:\text{Nat} \wedge d.\text{directory}(cl)=0 \wedge \text{this}=cl$
 $\{ d.\text{repay}() \}$

“nothing changes”

This spec avoids the vulnerability, 😅

provided the attack goes through the function `repay`. 😢

To avoid the vulnerability in general, we need to inspect the specification of *all* the functions in the DAO. DAO - interface has *nineteen* functions. 🤕

holistic

$\forall \text{cl:External. } \forall \text{d:DAO. } \forall \text{n:Nat.}$
[$\text{cl.Calls(d.repay())} \wedge \text{d.Balance(cl)} = n$
 \rightarrow
 $\text{d.ether} \geq n \wedge \text{Will(d.Calls(cl.send(n)))}$]

This specification avoids the vulnerability, regardless of which function introduces it:

The DAO will always be able to repay all its customers.

$$\begin{aligned} \text{d.Balance(cl)} = & \quad 0 && \text{if cl.Calls(d.initialize())} \\ & m+m' && \text{if Was(d.Balance(cl),m) } \wedge \text{cl.Calls(d.payIn(m'))} \\ & 0 && \text{if Was(cl.Calls(d.repayIn()))} \\ \text{Was(d.Balance(cl))} & \quad \text{otherwise} \end{aligned}$$

classical vs holistic

$\forall d:\text{DAO} . \ d.\text{ether} = \sum_{cl \in \text{dom}(d.\text{directory})} d.\text{directory}(cl)$

$d:\text{DAO} \wedge n:\text{Nat} \wedge d.\text{directory}(cl)=n>0 \wedge \text{this}=cl$
 { $d.\text{repay}()$ }
 $d.\text{directory}(cl)=0 \wedge d.\text{Calls}(cl.\text{send}(n))$

$d:\text{DAO} \wedge n:\text{Nat} \wedge d.\text{directory}(cl)=0 \wedge \text{this}=cl$
 { $d.\text{repay}()$ }
“nothing changes”

... ... specs for another 19 functions

....

Holistic

- *necessary* conditions for some action/effect
- *explicit* about emergent behaviour

Classical

- per function; *sufficient* conditions for some action/effect
- *explicit* about individual function, and *implicit* about emergent behaviour

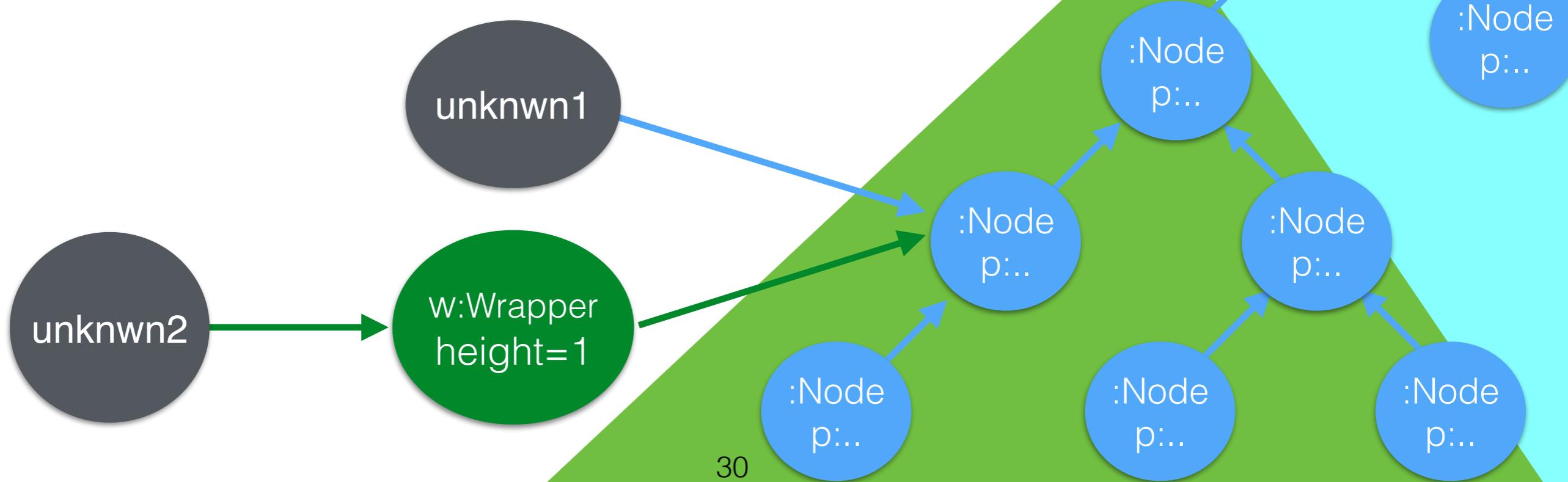
$\forall cl:\text{External} . \ \forall d:\text{DAO} . \ \forall n:\text{Nat} .$
[$cl.\text{Calls}(d.\text{repay}()) \wedge d.\text{Balance}(cl) = n$
→
 $d.\text{ether} \geq n \wedge \text{Will}(d.\text{Calls}(cl.\text{send}(n)))$]

$d.\text{Balance}(cl) =$	m	if $cl.\text{Calls}(d.\text{initialize}, m)$
	$m+m'$	if $\text{Prev}(d.\text{Balance}(cl), m)$ $\wedge \text{Prev}(cl.\text{Calls}(d.\text{payIn}(m')))$
	0	if $\text{Prev}(cl.\text{Calls}(d.\text{repayIn}(),))$

Example 3: DOM attenuation

Access to any Node gives access to **complete** tree

Wrappers have a height;
Access to Wrapper w allows modification of
Nodes under the $w \cdot \text{height}$ -th parent
and nothing else



DOM attenuation

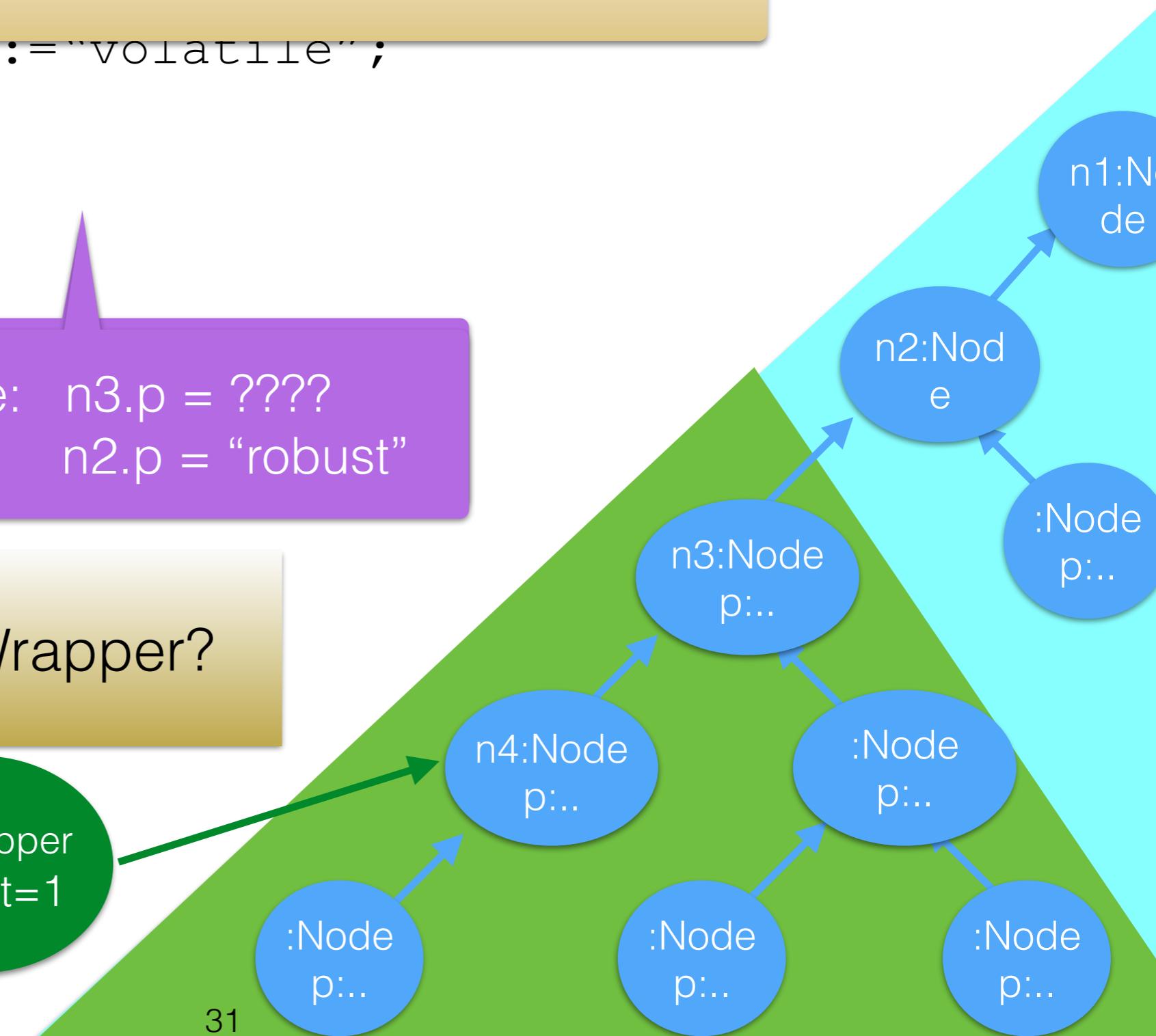
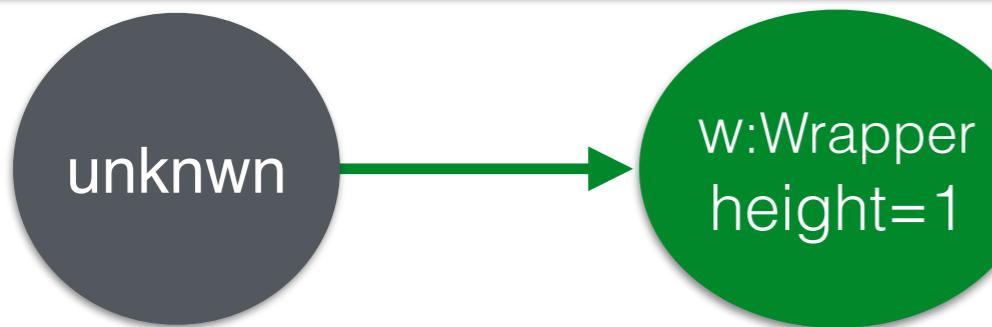
```
function mm(unknwn)
  n1:=Node (...); n2:=
  n2.p:="robust"; s.p:="volatile";
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
  ...
  ...
```

open world

:=Node (n3, ...);

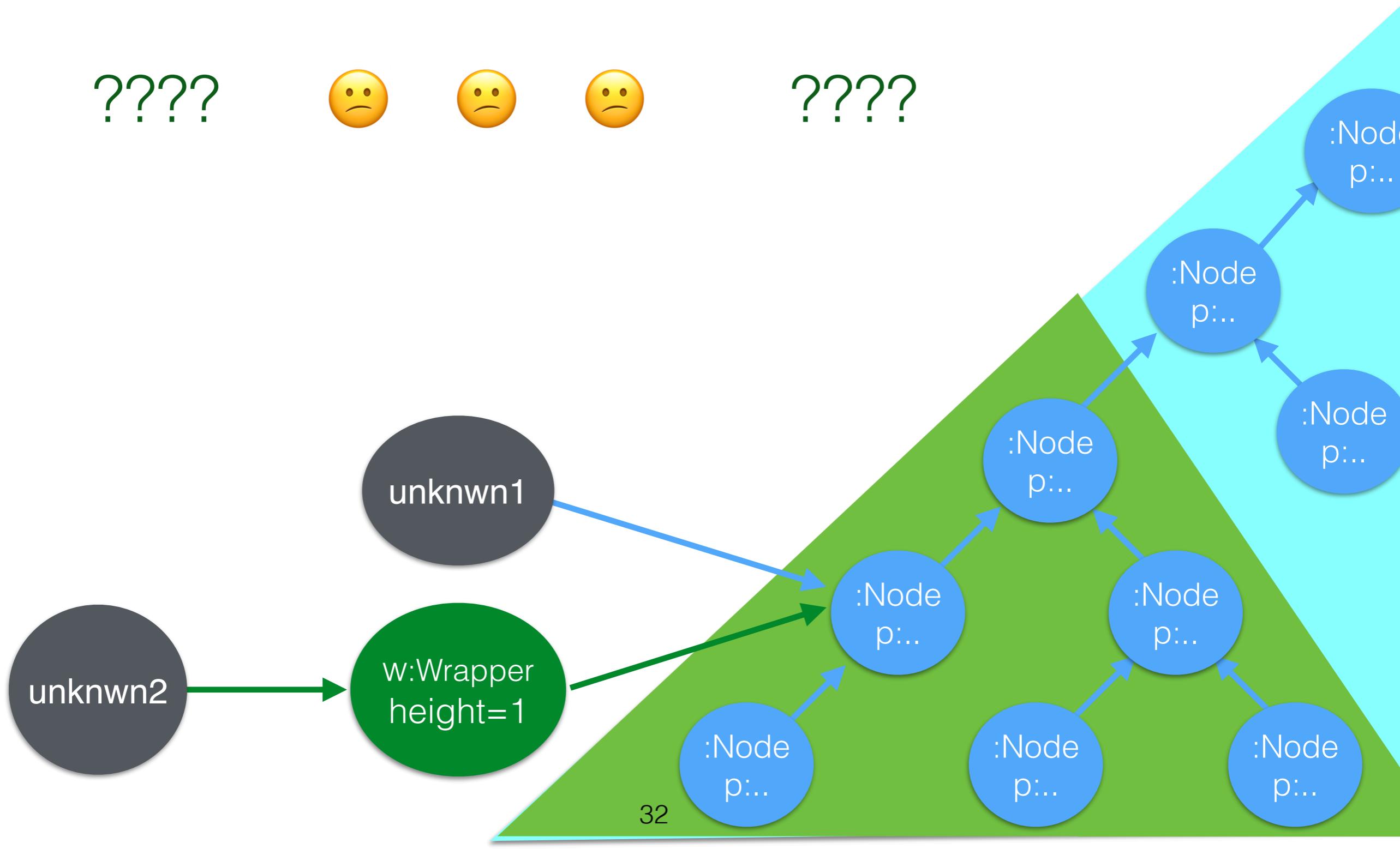
Here: n3.p = ???
n2.p = "robust"

How do we specify Wrapper?



classical

Access to Wrapper w allows modification of Nodes under the $w.height$ -th parent and nothing else



holistic

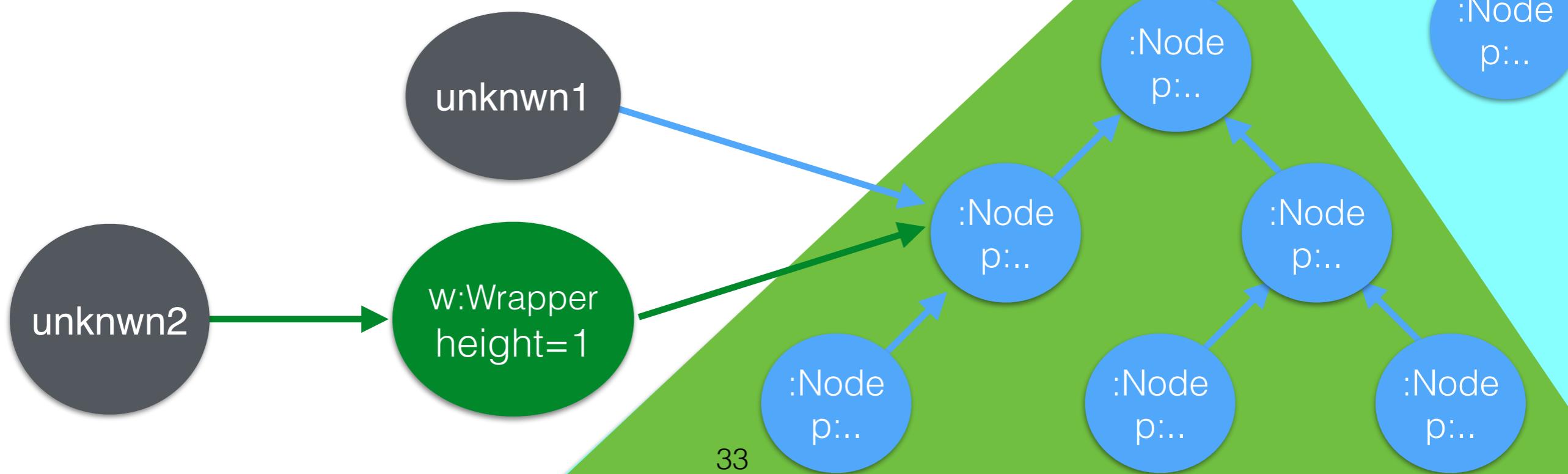
If

a node nd is “outside” set S

then

any execution involving no more than S does *not* modify $nd.p$

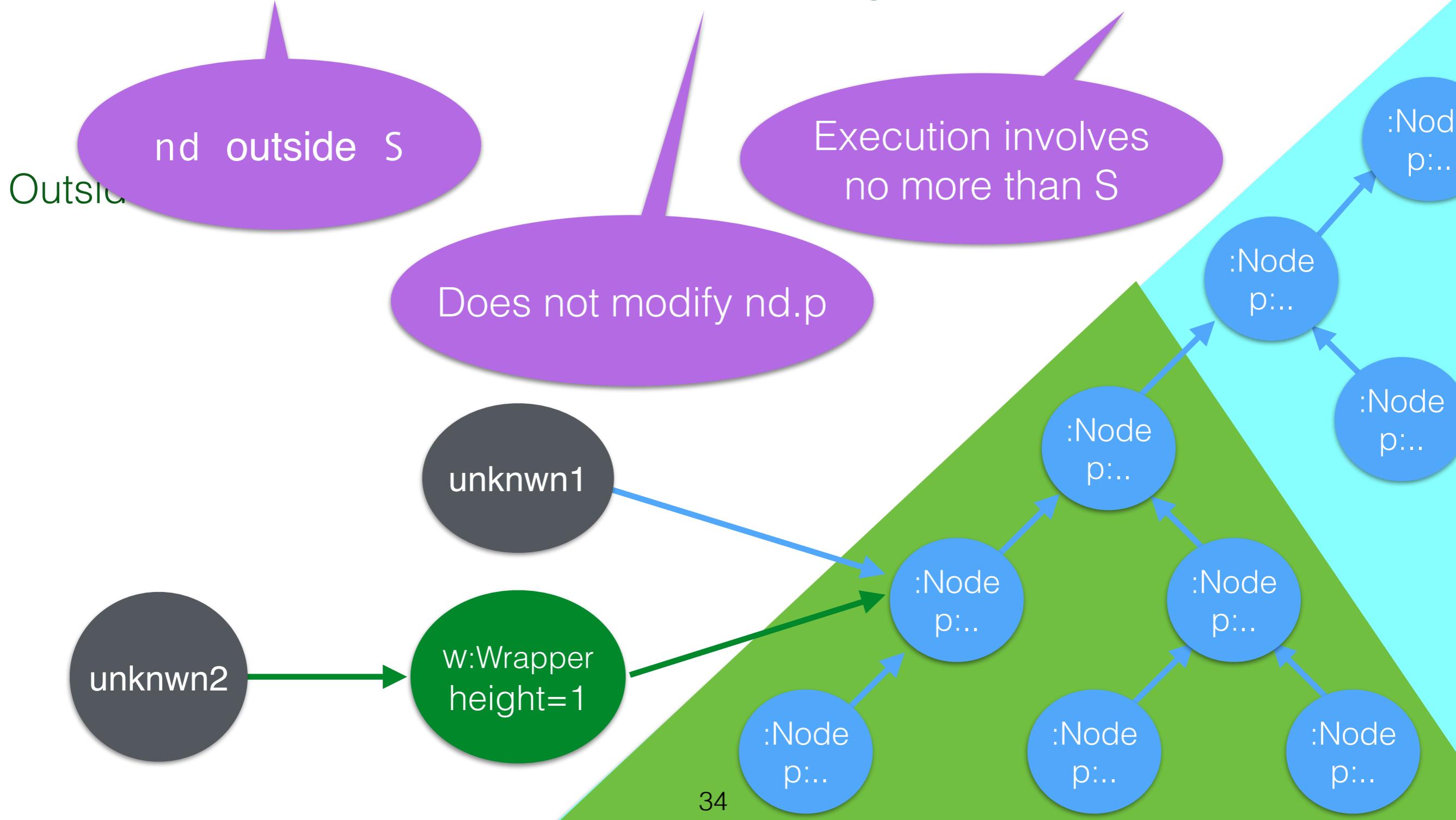
$\text{Outside}(nd, S)$ iff



holistic

$\forall S: \text{Set}, \forall nd:\text{Node}.$

[$\text{Outside}(nd, S) \rightarrow \neg (\text{WillChanges}(nd.p)) \text{ in } S$]



holistic

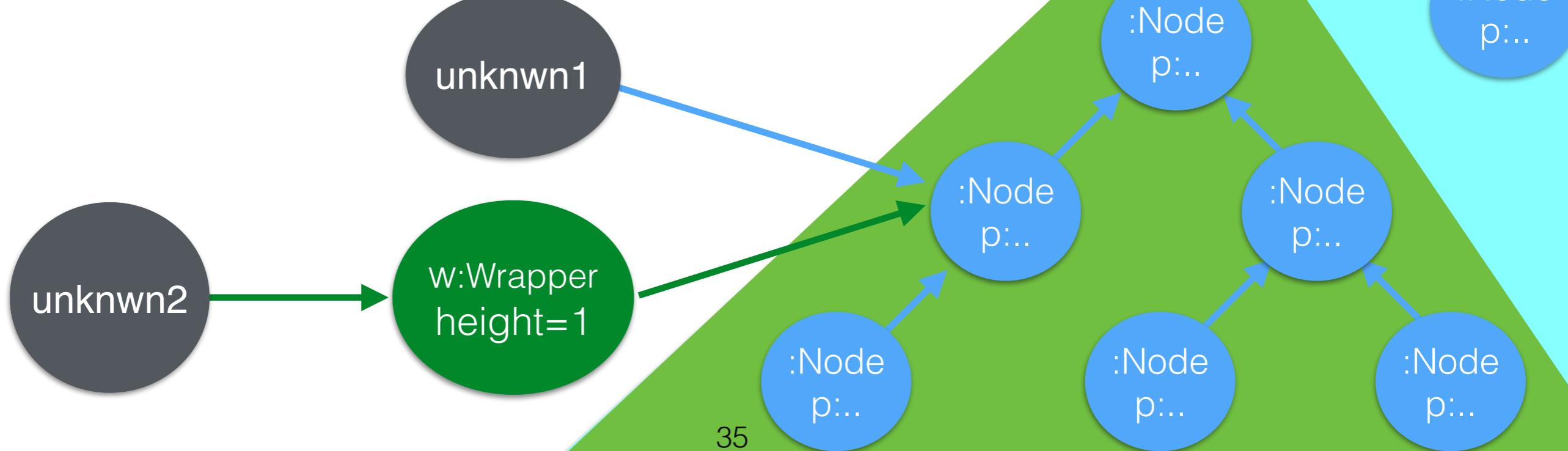
$\forall S: \text{Set. } \forall nd:\text{Node.}$

[Outside(nd,S) \rightarrow $\neg (\text{Will}(\text{Changes}(nd.p)) \text{ in } S)$]

Outside(nd,S) iff $\forall o \in S. \forall \text{path}$

[$o.\text{path} = nd \rightarrow$
 [$o:\text{Node} \vee$
 $\exists \text{path}', fs. (\text{path} = \text{path}'.fs \wedge o.\text{path}' : \text{Wrapper} \wedge$
 $\text{Distance}(o.\text{path}', nd) > o.\text{path}'.\text{height})]]$

$\text{Distance}(nd, nd') = \min\{ k \mid nd.\text{parent}^k = nd'.\text{parent}^j \}$

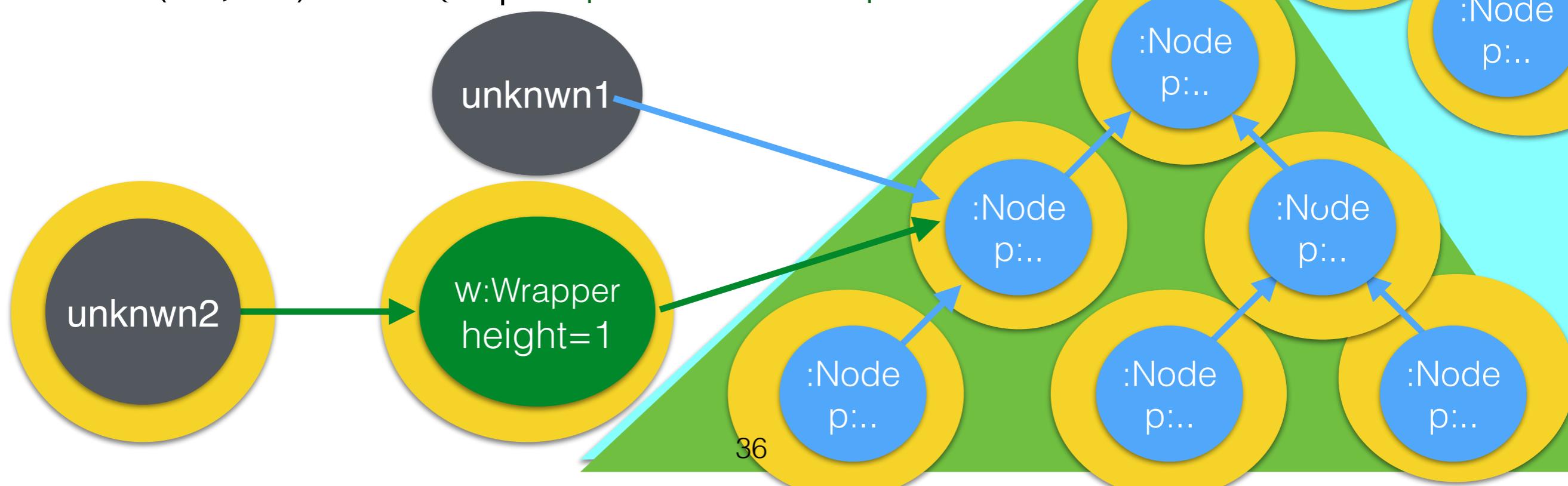


Outside(**RedNode**, **YellowSet**)

Outside(nd,S) iff $\forall o \in S. \forall \text{path}$

[$o.\text{path} = nd \rightarrow$
 [$o:\text{Node} \vee$
 $\exists \text{path}', fs. (\text{path} = \text{path}'.fs \wedge o.\text{path}' : \text{Wrapper} \wedge$
 $\text{Distance}(o.\text{path}', nd) > o.\text{path}'.\text{height})]]$

$\text{Distance}(nd, nd') = \min\{ k \mid nd.\text{parent}^k = nd'.\text{parent}^j \}$

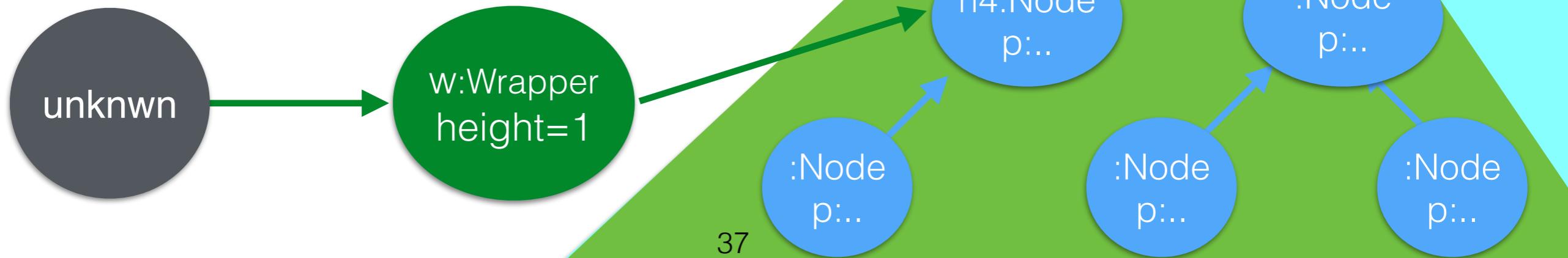


using holistic spec

```
function mm(unknwn) {  
    n1:=Node (...); n2:=Node (n1, ...); n3:=Node (n2, ...); n4:=Node (n3, ...);  
    n2.p:="robust"; n3.p:="volatile";  
    w=Wrapper (n4, 1);  
    unknwn.untrusted (w);  
    ...  
}
```



With holistic spec we can show
that despite the call to unknown object,
at this point:
 $n2.p = "robust"$



Bank and Account

- Banks and Accounts
- Accounts hold money
- Money can be transferred between Accounts
- A banks' currency = sum of balances of accounts held by bank

[Miller et al, Financial Crypto 2000]

Bank/Account - 2

classical

robustness

- **Pol_1:** With two accounts of same bank one can transfer money between them.
- **Pol_2:** Only someone with the Bank of a given currency can violate conservation of that currency
- **Pol_3:** The bank can only inflate its own currency
- **Pol_4:** No one can affect the balance of an account they do not have.
- **Pol_5:** Balances are always non-negative.
- **Pol_6:** A reported successful deposit can be trusted as much as one trusts the account one is depositing to.

[Miller et al, Financial Crypto 2000]

Pol_4 – classical

- **Pol_4:** No-one can affect the balance of an account they do not have

????



????

Pol_4 – holistic

- **Pol_4**: No-one can affect the balance of an account they do not have

$a:\text{Account} \wedge \text{Will}(\text{Changes}(a.\text{balance})) \text{ in } S$



$\exists o \in S. [\text{External}(o) \wedge \text{Access}(o,a)]$

necessary condition

This says: If some execution starts now and involves at most the objects from S , and modifies $a.\text{balance}$ at some future time, then at least one of the objects in S is external to the module, and can access a directly now.

Today

- Traditional Specifications do not adequately address Robustness
- Holistic Specifications — Summary and by Example
- **Holistic Specification Semantics**

Chainmail

Swapsies on the Internet

First Steps towards Reasoning about Risk and Trust in an Open World

Sophia Drossopoulou¹, James Noble², Mark S. Miller³

¹Imperial College London, ²Victoria University Wellington, ³Google Inc.

How to Break the Bank: Semantics of Capability Policies

Sophia Drossopoulou¹ and James Noble²

¹ Imperial College London s.drossopoulou@imperial.ac.uk

² Victoria University of Wellington k.jx@ecs.vuw.ac.nz

Holistic Specifications for Robust Programs

Sophia Drossopoulou¹[0000-1111-2222-3333], James Noble²[0000-1111-2222-3333], Julian Mackay²[0000-1111-2222-3333], and Susan Eisenbach¹[0000-1111-2222-3333]

¹ Imperial College London {scd, susan}@springer.com

² Victoria University of Wellington {julian.mackay, k.jx}@ecs.vuw.ac.nz

Abstract Functional specifications describe what program components *can* do: the *sufficient* conditions to invoke a component's operations. They allow us to reason about the use of components in the *closed world* setting, where the component interacts with known client code, and where the client code must establish the appropriate pre-conditions before calling into the component.

Sufficient conditions are not enough to reason about the use of components in the *open world* setting, where the component interacts with external code, possibly of unknown provenance, and where the component itself may evolve over time. In this open world setting, we must also consider the *necessary* conditions,



Loo syntax

```
ClassDescr ::= class ClassId { ( FieldDecl )* ( MethDecl )* ( GhosDecl )* }
```

```
FieldDecl ::= field f
```

```
MethDecl ::= method m( x* ) { Stmts }
```

```
Stmts ::= Stmt | Stmt ; Stmts
```

```
Stmt ::= x.f := x | x := x.f | x := x.m( x* ) | x := new C( x* ) | return x
```

```
GhostDecl ::= ghost f( x* ) { e }
```

```
e ::= true | false | null | x | e=e | if e then e else e | e.f( e* )
```

```
x, f, m ::= Identifier
```

Giving meaning to holistic Assertions

We define in a “conventional” way (omit from slides):

module $M : \text{Ident} \rightarrow \text{ClassDef} \cup \text{PredicateDef} \cup \text{FunctionDef}$
configuration $\sigma : \text{Heap} \times \text{Stack} \times \text{Code}$
execution $M, \sigma \rightsquigarrow \sigma'$

Define module concatenation $*$ so that

M^*M' undefined, iff $\text{dom}(M) \cap \text{dom}(M') \neq \emptyset$

otherwise

$(M^*M')(id) = M(id)$ if $M'(id)$ undefined, else $M'(id)$

We can also define

- $M^*M' = M_1^*M_2 \models A$
- $(M_1^*M_2)^*M_3 \models \text{Initial}(M_1) \wedge \text{Final}(M_2) \wedge \text{Missing}(M_3)$
- $M, \sigma \rightsquigarrow \sigma' \models M^*M' \text{ defined} \longrightarrow M^*M', \sigma \rightsquigarrow \sigma'$

Giving meaning to holistic Assertions

We define in a “conventional” way (omit from slides):

module $M : \text{Ident} \rightarrow \text{ClassDef} \cup \text{PredicateDef} \cup \text{FunctionDef}$
configuration $\sigma : \text{Heap} \times \text{Stack} \times \text{Code}$
execution $M, \sigma \rightsquigarrow \sigma'$

Define module concatenation $*$ so that

M^*M' undefined, iff $\text{dom}(M) \cap \text{dom}(M') \neq \emptyset$

otherwise

$(M^*M')(id) = M(id)$ if $M'(id)$ undefined, else $M'(id)$

We will define $M, \sigma \models A$

$\text{Initial}(\sigma)$ and $\text{Arising}(M)$

$M \models A$

Holistic Assertions – summary

$e ::= \text{this} \mid x \mid e.\text{fld} \mid \dots$

$A ::= e>e \mid e=e \mid \dots$
 $\mid A \rightarrow A \mid A \wedge A \mid \exists x. A \mid \dots$

| **Access**(e, e') permission

| **Changes**(e) authority

| **Will**(A) | **Was**(A) time

| **A in S** | **External**(e) space

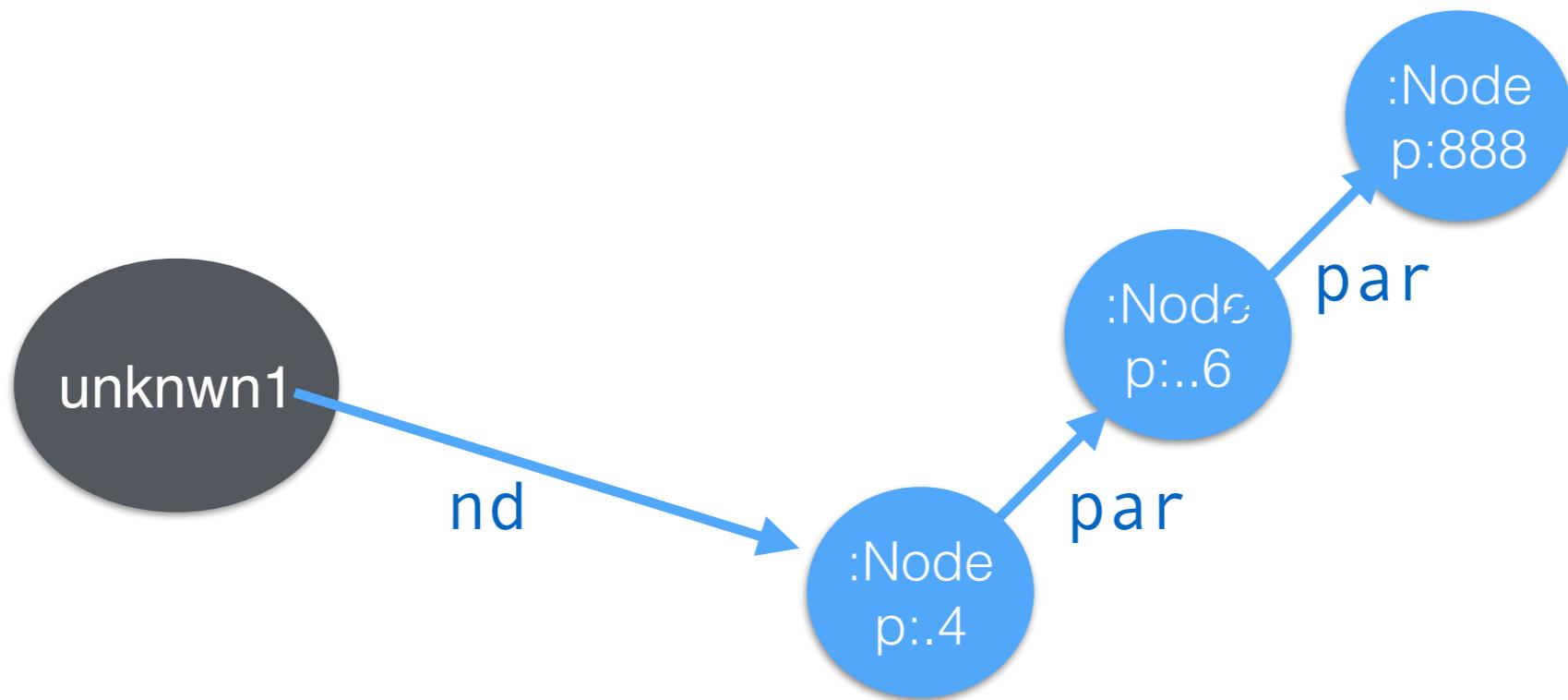
| $x.\text{Call}(y, m, z_1, \dots, z_n)$ control

| $x \text{ obeys } A$ trust

Semantics of Expressions

$e ::= \text{this} \mid x \mid e.\text{fld} \mid \text{func}(e_1, \dots, e_n) \mid \dots$

Define $\llbracket e \rrbracket_{M, \sigma}$ as expected



Eg, $\llbracket \text{unknwn1}.nd.par.par.p \rrbracket_{M, \sigma} = 888$

Semantics of holistic Assertions

“Conventional part”

$$A ::= e > e \mid A \rightarrow A \mid \exists x. A \mid \dots$$

We define $M, \sigma \models A$

$$M, \sigma \models e > e' \quad \text{iff} \quad \lfloor e \rfloor_{M, \sigma} > \lfloor e' \rfloor_{M, \sigma}$$

$$M, \sigma \models A \rightarrow A' \quad \text{iff} \quad M, \sigma \models A \text{ implies } M, \sigma \models A'$$

$$M, \sigma \models \exists x. A \quad \text{iff} \quad M, \sigma[z \mapsto l] \models A[x \mapsto z] \\ \text{for some } l \in \text{dom}(\sigma.\text{heap}), \text{ and } z \text{ free in } A$$

Semantics of holistic Assertions

“Unconventional part”

$A ::= \text{Access}(x,x') \mid \text{Changes}(e) \mid \text{Will}(A) \mid A \text{ in } S \mid x.\text{Calls}(y,m,z_1..z_n)$

$M, \sigma \models \text{Access}(x,x')$ iff $\lfloor x \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma} \vee \lfloor x.\text{fld} \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$ for some field fld $\vee \lfloor \text{this} \rfloor_{M,\sigma} = \lfloor x \rfloor_{M,\sigma} \wedge \lfloor y \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$ $\wedge y$ is formal parameter of current function

$M, \sigma \models \text{Changes}(e)$ iff $M, \sigma \rightsquigarrow \sigma' \wedge \lfloor e \rfloor_{M,\sigma} \neq \lfloor e \rfloor_{M,\sigma'}$

$M, \sigma \models \text{Will}(A)$ iff $\exists \sigma'. [M, \sigma \rightsquigarrow^* \sigma' \wedge M, \sigma' \models A]$

$M, \sigma \models A \text{ in } S$ iff $M, \sigma @ Os \models A$ where $Os = \lfloor S \rfloor_{M,\sigma}$

$M, \sigma \models x.\text{Calls}(y,m,z_1..z_n)$ iff $\lfloor \text{this} \rfloor_{M,\sigma} = \lfloor x \rfloor_{M,\sigma} \wedge \sigma.\text{code}=y'.m(z_1'..z_n')$ $\wedge \dots$

Semantics of holistic Assertions

- the full truth -

$M, \sigma \models \text{Access}(e, e')$ iff ... as before ...

$M, \sigma \models \text{Changes}(e)$ iff $M, \sigma \rightsquigarrow \sigma' \wedge \llbracket e \rrbracket_{M, \sigma} \neq \llbracket e[z \mapsto y] \rrbracket_{M, \sigma'[y \mapsto \sigma(z)]}$
where $\{z\} = \text{Free}(e) \wedge y \text{ fresh in } e, \sigma, \sigma'$

$M, \sigma \models \text{Will}(A)$ iff $\exists \sigma', \sigma'', \phi. [\sigma = \sigma'.\phi \wedge M, \phi \rightsquigarrow^* \sigma' \wedge M, \sigma'[y \mapsto \sigma(z)] \models A[z \mapsto y]]$
where $\{z\} = \text{Free}(A) \wedge y \text{ fresh in } A, \sigma, \sigma'$

$M, \sigma \models A \text{ In } S$ iff $M, \sigma @ Os \models A$ where $Os = \llbracket S \rrbracket_{M, \sigma}$

$M, \sigma \models x. \text{Calls}(y, m, z_1,..z_n)$ iff ... as before ...

Arising Configurations

A runtime configuration is *initial* iff

- 1) The heap contains only one object, of class Object
- 2) The stack consists of just one frame, where **this** points to that object.

The code can be arbitrary

$$\text{Initial}(\sigma) \text{ iff } \sigma.\text{heap} = (1 \mapsto (\text{Object}, \dots)) \wedge \sigma.\text{stack} = (\text{this} \mapsto 1).[]$$

A runtime configuration σ arises from a module M if there is some initial configuration σ_0 whose execution M reaches σ in a finite number of steps.

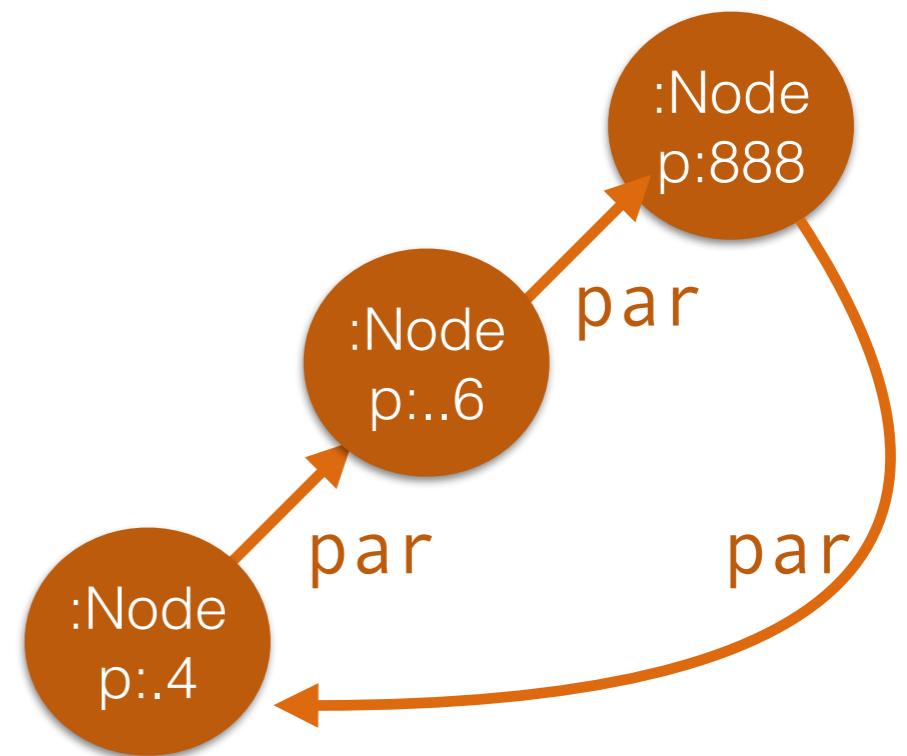
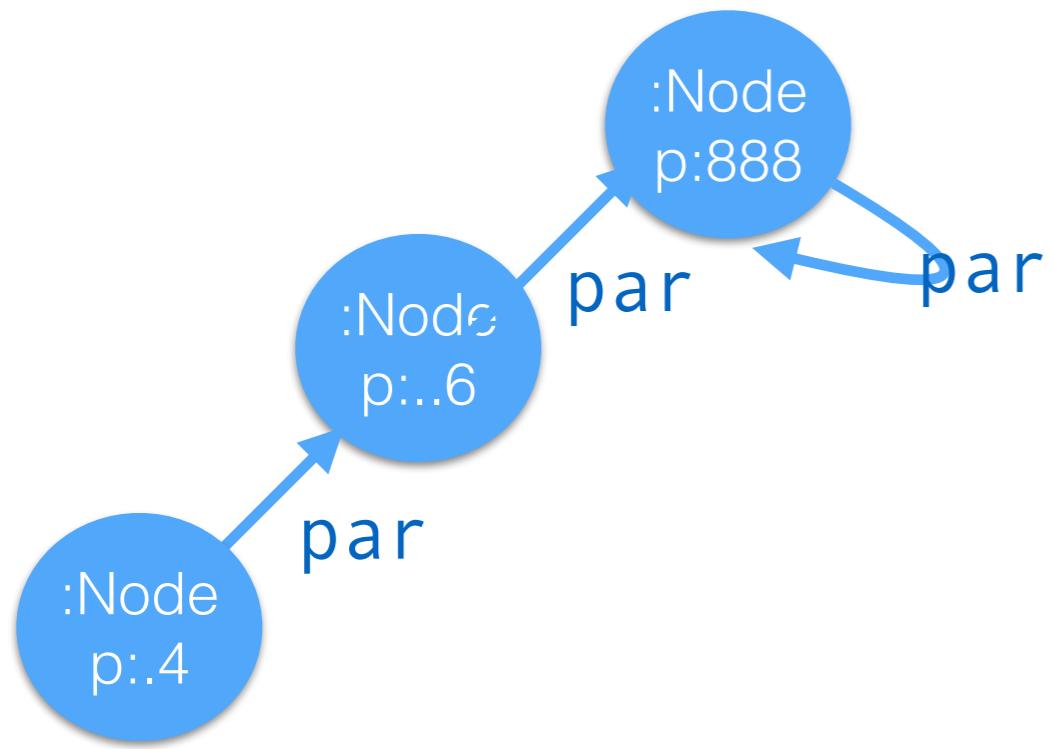
$$\text{Arising}(M) = \{ \sigma \mid \exists \sigma_0. \text{Initial}(\sigma_0) \wedge M, \sigma_0 \rightsquigarrow^* \sigma \}$$

Arising expresses “defensiveness”

Assume a Tree-module, M_{tree} .

blue configuration arises from $M_{tree} * M'$ for some module M'

brown configuration does not arise from $M_{tree} * M'$ for any module M'



Giving meaning to Assertions

$$M \models A \text{ iff } \boxed{\forall M'}. \forall \sigma \in \text{Arising}(M ; M'). M ; M', \sigma \models A$$

A module M satisfies an assertion A if all runtime configurations σ which arise from execution of code from $M ; M'$ (for any module M'), satisfy A .

open world

- quantify over external modules M'
- M' is external; intermediate states observable
- M is internal; intermediate states are not observable

Summary of our Proposal

$A ::= e > e \mid e = e \mid f(e_1, \dots, e_n) \mid \dots$

$\mid A \rightarrow A \mid A \wedge A \mid \exists x. A \mid \dots$

$\mid \mathbf{Access}(x, y)$ permission

$\mid \mathbf{Changes}(e)$ authority

$\mid \mathbf{Will}(A) \mid \mathbf{Was}(A) \mid \mathbf{Next}(A) \mid \mathbf{Prev}(A)$ time

$\mid A \mathbf{in} S$ space

$\mid x. \mathbf{Calls}(y, m, z_1, \dots, z_n)$ call

$M ; M', \sigma \models A$

Arising($M ; M'$)

$M \models A$

Summary

- We argue that robustness characterisation requires a mix of classical and holistic specifications.
- We have made a proposal for holistic specifications and applied to examples from contracts and from object capabilities literature.

Classical vs Holistic Specification

- fine-grained
- per function
- ADT as a whole
- emergent behaviour

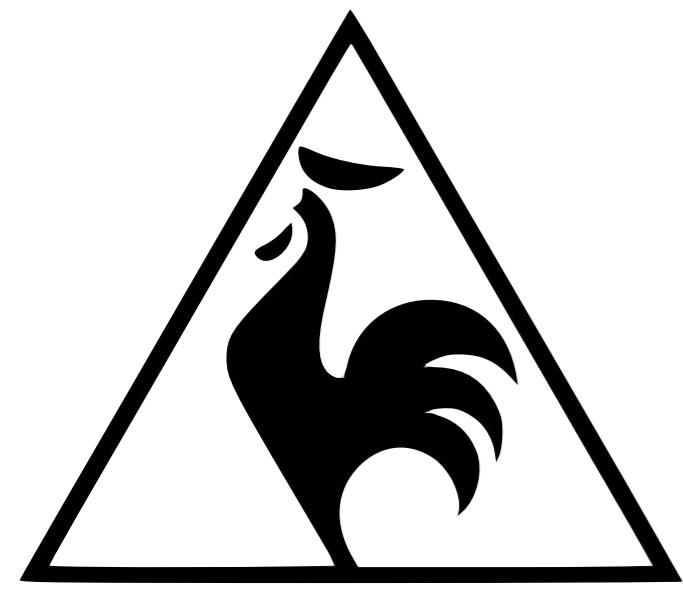
Which is “stronger”?

“Closed” ADT with classical spec implies holistic spec.

(closed: no functions can be added, all functions have classical specs, ghost state has known representation)

Why do we need holistic specs?

- * “closed ADT” is sometimes too strong a requirement.
- * Holistic aspect is cross-cutting (eg no payment without authorization)
- * Allows reasoning in open world (eg DOM wrappers)



Properties of Linking

- (1) $A \wedge \neg A \equiv \text{false}$
- (2) $A \vee \neg A \equiv \text{true}$
- (3) $A \vee A' \equiv A' \wedge A$
- (4) $A \wedge A' \equiv A' \wedge A$
- (5) $(A \vee A') \vee A'' \equiv A \vee (A' \vee A'')$

- (1) moduleLinking_associative
- (2) moduleLinking_commutative_1
- (3) moduleLinking_commutative_2
- (4) linking_preserves_reduction

- (1) $A \wedge \neg A \equiv \text{false}$
- (2) $A \vee \neg A \equiv \text{true}$
- (3) $A \vee A' \equiv A' \wedge A$
- (4) $A \wedge A' \equiv A' \wedge A$
- (5) $(A \vee A') \vee A'' \equiv A \vee (A' \vee A'')$
- (6) $(A \vee A') \wedge A'' \equiv (A \wedge A'') \vee (A' \wedge A'')$
- (7) $(A \wedge A') \vee A'' \equiv (A \vee A'') \wedge (A' \vee A'')$
- (8) $\neg(A \wedge A') \equiv (\neg A \vee \neg A')$
- (9) $\neg(A \vee A') \equiv (\neg A \wedge \neg A')$
- (10) $\neg(\exists x.A) \equiv \forall x.(\neg A)$
- (11) $\neg(\exists S.A) \equiv \forall S.(\neg A)$
- (12) $\neg(\forall x.A) \equiv \exists x.(\neg A)$
- (13) $\neg(\forall S.A) \equiv \exists S.(\neg A)$

- (1) sat_and_nsat_equiv_false
- (2) -
- (3) and_commutative
- (4) or_commutative
- (5) or_associative

- (1) sat_and_nsat_equiv_false
- (2) -
- (3) and_commutative
- (4) or_commutative
- (5) or_associative
- (6) and_distributive
- (7) or_distributive
- (8) neg_distributive_and
- (9) neg_distributive_or
- (10) not_ex_x_all_not
- (11) not_ex_Σ_all_not
- (12) not_all_x_ex_not
- (13) not_all_Σ_ex_not

Challenges

- two module execution and assertion
- adaptations for time, Was<...>
- dealing with space <.. in ..>
- ghostfields, (treat cycles)
- keeping assertions classical
- Language properties,
eg only connectivity begets connectivity
- Coq model and proofs of language properties
- Coq proof of some of the holistic specs

Still to do ..

- Proof logic using holistic assertions (in progress)
- Proof logic to establish holistic assertions.
Can we avoid explicit naming configurations?
- Trust and Risk A ::= ... | e **obeys** Spec *trust*

Thank you

