Holistic Specifications
classification of robustness

Sophia Drossopoulou

and

James Noble (VU Wellington),
Mark Miller (Agorics),
Toby Murray (Uni Melbourne)
Shupeng Loh and Emil Klasan (Imperial)
Functional

• *services offered* by objects/data structure to *clients*,

• what *will* happen, under *correct use*

• *sufficient* conditions
Functional vs

• services offered by objects/data structure to clients,

• what will happen, under correct use

• sufficient conditions
Functional vs Robust

- services offered by objects/data structure to clients,

- what will happen, under correct use

- sufficient conditions
Functional vs Robust

- *services offered* by objects/data structure to *clients*,
- what *will* happen, under *correct use*
- *sufficient conditions*

- *preserved properties* of the objects/data structure
- what *will not* happen, under *arbitrary use*
- *necessary conditions*
• Functional ≠ Robustness

- Robustness in terms of the Bank/Account Example

• Holistic Specification: “Classical assertions”
  + Time
  + Space
  + Access
  + Authority
  “in an open world”

• Examples
Bank/Account

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

[Miller et al, Financial Crypto 2000]
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]
Bank/Account - 2

- **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]
Bank/Account - 2

**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_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_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_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]
Bank/Account - 2

- **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_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]
1: Ban

? ? ? ?

2: Acc  3: Acc  4: Acc
Should the following be possible?
- 21 takes money from 4.
- 21 takes money from 2.
- 10 affects the currency.
- 10 takes money from 4.
- 21 finds out 2’s balance.
Should the following be possible?

- 21 takes money from 4.
- 21 takes money from 2.
- 10 affects the currency.
- 10 takes money from 4.
- 21 finds out 2’s balance.

- **Pol_2**: Only someone with the Bank of a given currency can violate conservation of that currency.
- **Pol_4**: No one can affect the balance of an account they do not have.
Should the following be possible?

- 21 takes money from 4.  
- 21 takes money from 2.  
- 10 affects the currency.  
- 10 takes money from 4.  
- 21 finds out 2’s balance.

- **Pol_2**: Only someone with the Bank of a given currency can violate conservation of that currency.
- **Pol_4**: No one can affect the balance of an account they do not have.
class Account {
    fld myBank       // a Bank
    fld balance      // a number

    Account(aBank,amt){ myBank=aBank; balance=amt }

    fun deposit(destination,amt)
    {
        if myBank==destination.myBank then
        {
            this.balance-=amt;
            destination.balance+=amt
        }
    }
}

class Bank {
    // class definitions
}

MBA1: Code
class Account {
    fld myBank     // a Bank
    fld balance    // a number
    Account(aBank, amt) { myBank = aBank; balance = amt }
    fun deposit(destination, amt) {
        if myBank == destination.myBank then
            this.balance -= amt;
            destination.balance += amt
    }
}

class Bank {
}

Note: bank.currency is a model field
MBA1: Objects
class Bank{  }
class Bank{  }

class Account {  
    fld myBank  
    fld balance  
    ...
}

class Bank{  }
class Account {  
  fld myBank
  fld balance
  ....
}
**MBA1: Adherence to Policies**

- **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_4**: No one can affect the balance of an account they do not have.
**MBA1: Adherence to Policies**

- **Pol_1**: With two accounts of the 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_4**: No one can affect the balance of an account they do not have.
**MBA1: Adherence to Policies**

- **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_4**: No one can affect the balance of an account they do not have.
**MBA1: Adherence to Policies**

- **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_4**: No one can affect the balance of an account they do not have.
MBA1: Adherence to Policies

Pol_1: With two accounts of same bank one can transfer money between them.

MBA1 ⊨ Pol_1

Pol_2: Only someone with the Bank of a given currency can violate conservation of that currency

MBA1 ⊭ Pol_2

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

MBA1 ⊨ Pol_4
class Account {
    class private fld myBank     // a Bank
    class private fld balance    // a number

    Account(aBank, amt){ myBank = aBank; balance = amt }

    fun deposit(destination, amt) {
        if myBank == destination.myBank then
            this.balance -= amt;
            destination.balance += amt
    }
}

class Bank {

}
MBA1s: Adherence to Policies

- **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_4**: No one can affect the balance of an account they do not have.

_indicates a private field_
MBA1s: Adherence to Policies

- **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_4**: No one can affect the balance of an account they do not have.
class Bank {
    fld book // a Node
    Bank() { book=null }
    fun makeAccount(amt){ ... }
    fun deposit(src,dest,amt){
        srce=book.get(src);
        destn=book.get(dest);
        if srce.balance>amt then
            { srce.balance-=amt;
              destn.balance+=amt }  }
}

class Node {
    fld balance // a number
    fld next // a Node
    fld theAccount // an Account
    fun get(acc){
        if theAccount==acc
            then{ this; }
        else{ ... next.get(acc) ... }
    }
}

class Account {
    fld myBank // a Bank
    Account(aBank) { myBank=aBank }
    fun deposit(destination,amt)
        { myBank.deposit(this,destination,amt) }
    }

MBA2: Code
class Bank {
    fld book // a Node
    Bank() { book=null }
    fun makeAccount(amt){ ... }
    fun deposit(src, dest, amt) {
        srce=book.get(src);
        destn=book.get(dest);
        if srce.balance>amt then 
            { srce.balance-=amt;
              destn.balance+=amt } 
    }
}

MBA2: Code

class Account {
    fld myBank // a Bank
    Account(aBank) { myBank=aBank }
    fun deposit(destination, amt) {
        myBank.deposit(this, destination, amt) 
    }
}

class Node {
    fld balance // a number
    fld next // a Node
    fld theAccount // an Account
    fun get(acc) {

    }
}

bank.currency is model field
account.balance is model field
MBA2: Objects
class Bank {
    fld book
    ...
}

MBA2: Objects
class Bank {
  fld book

  ...
}

class Node {
  fld balance
  fld next
  fld theAccount

  ...
}
class Bank {
    fld book
    ...
}

class Node {
    fld balance
    fld next
    fld theAccount
    ...
}

class Account {
    fld myBank
    ....
}
class Bank {
    fld book
    ...
}

class Node {
    fld balance
    fld next
    fld theAccount
    ...
}

class Account {
    fld myBank
    ...
}

class Unknown1 {
    fld ...
    ...
}

class Unknown1 {
    fld ...
    ...
}
- **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_4**: No one can affect the balance of an account they do not have.
**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_4**: No one can affect the balance of an account they do not have.
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_4: No one can affect the balance of an account they do not have.
**MBA2: Adherence to Policies**

- **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_4**: No one can affect the balance of an account they do not have.
MBA2: Adherence to Policies

- **Pol_1**: With two accounts of the same bank, one can transfer money between them.  
  - MBA2 ⊨ Pol_1

- **Pol_2**: Only someone with the Bank of a given currency can violate conservation of that currency.  
  - MBA2 ⫬ Pol_2

- **Pol_4**: No one can affect the balance of an account they do not have.  
  - MBA2 ⫬ Pol_4

- **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_4**: No one can affect the balance of an account they do not have.
class Bank {
    instance private fld book // a Node
    Bank( ){ book=null }
    fun makeAccount(amt){ ... }
    fun deposit(src,dest,amt){
        srce=book.get(source);
        destn=book.get(dest);
        if srce.balance>amt then
            { srce.balance-=amt;
              destn.balance+=amt }
    }
}

class Node {
    fld balance // a number
    fld next // a Node
    fld theAccount // an Account
    fun get(acc){
        if theAccount==acc
            then{ this; }
        else{ ... next.get(acc) ... }
    }
}

class Account {
    instance private fld myBank // a Bank
    Account(aBank){ myBank=aBank }
    fun deposit(destination,amt)
        { myBank.deposit(this,destination,amt) }
}

MBA2s: safe Code
MBA2s: Adherence to Policies

- **Pol_1**: With two accounts of the same bank, one can transfer money between them.

- **Pol_2**: Only someone with the bank of a given currency can violate the conservation of that currency.

- **Pol_4**: No one can affect the balance of an account they do not have.
MBA2s: Adherence to Policies

- **Pol_1**: With two accounts of the 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_4**: No one can affect the balance of an account they do not have.
Research Questions:

- Formalize policies such as $Pol_1, \ldots, Pol_5$
- Meaning of $Mx \models Pol_v$
Functional ≠ Robustness

Robustness in terms of the Bank/Account Example

Holistic Specification:
“Classical assertions” + Time
+ Space
+ Permission
+ Authority
+ “in an open world”

Examples
Assertions
Assertions

e ::= this | x | e.fld | func(e1,...,en) | ...
Assertions

e ::= this | x | e.fld | func(e1,...en) | ...

A ::= e>e | e=e | P(e1,..en) | ...
Assertions

e ::= this | x | e.fld | func(e1,...en) | ...

A ::= e>e | e=e | P(e1,..en) | ...
   | A → A | A ∧ A | ∃x. A | ...


Assertions

e ::= this | x | e.fld | func(e1,...en) | ...
A ::= e>e | e=e | P(e1,..en) | ...
    | A → A | A ∧ A | ∃x. A | ...
    | Access(e,e')
Assertions

\[
e ::= \text{this} \mid x \mid e.fld \mid \text{func}(e_1,\ldots,e_n) \mid \ldots
\]

\[
A ::= e>e \mid e=e \mid P(e_1,\ldots,e_n) \mid \ldots
\]

\[
\mid A \rightarrow A \mid A \land A \mid \exists x. A \mid \ldots
\]

\[
\mid \text{Access}(e,e')
\]

\[
\mid \text{Changes}(e)
\]
Assertions

e ::= this | x | e.fld | func(e1,...en) | ...

A ::= e>e | e=e | P(e1,..en) | ...
    | A → A | A ∧ A | ∃x. A | ...
    | Access(e,e')
    | Changes(e)
    | •A | ◦A
Assertions

\[ e ::= \text{this} \mid x \mid e.fld \mid \text{func}(e_1, \ldots, e_n) \mid \ldots \]

\[ A ::= e > e \mid e = e \mid P(e_1, \ldots, e_n) \mid \ldots \]

\[ \mid A \rightarrow A \mid A \land A \mid \exists x. A \mid \ldots \]

\[ \mid \text{Access}(e, e') \]

\[ \mid \text{Changes}(e) \]

\[ \mid \bullet A \mid \circ A \]

\[ \mid A \mathrel{@} S \]
Assertions

e ::= this | x | e.fld | func(e1,...en) | ...
A ::= e>e | e=e | P(e1,..en) | ...
    | A → A | A ∧ A | ∃x. A | ...
    | Access(e,e')
    | Changes(e)
    | •A | ◦A
    | A @ S
    | x.Call(y,m,z1,..zn)
# Assertions

\[ e ::= \text{this} | x | e.fld | \text{func}(e_1,\ldots,e_n) | \ldots \]

\[ A ::= e>e | e=e | P(e_1,\ldots,e_n) | \ldots \]

\[ \begin{array}{|c|c|c|c|}
\hline
A & A \rightarrow A & A \land A & \exists x. A \\
\hline
\text{Access}(e,e') & \text{Changes}(e) & \text{•A} & \circ A \\
\hline
A @ S & \text{x.Call}(y,m,z_1,\ldots,z_n) & \text{permission} & \text{authority} & \text{time} & \text{space} & \text{call} \\
\hline
\end{array} \]
**Formalizing Pol_1**

- **Pol_1**: With two accounts of same bank one can transfer money between them.
Formalizing Pol_1

- **Pol_1**: With two accounts of same bank one can transfer money between them.

\[
\text{Pol}_1 \equiv a1:\text{Account} \land a2:\text{Account} \land a1 \neq a2 \land a1.\text{myBank} = a2.\text{myBank} \land \\
\text{..................................................} \land \\
\text{..................................................} \land \\
\text{..................................................} \land \\
\text{..................................................} \land \\
\]

→
Formalizing Pol_1

Pol_1: With two accounts of same bank one can transfer money between them.

\[ \text{Pol}_1 \equiv a1:\text{Account} \land a2:\text{Account} \land a1 \neq a2 \land a1.\text{myBank} = a2.\text{myBank} \land \]
\[ \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \land \]
\[ \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \land \]
\[ \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \land \]
\[ \rightarrow \]

\bullet ( a1.\text{balance} = \ldots - \text{amt} \land a2.\text{balance} = \ldots + \text{amt} ) \]
Formalizing Pol_1

**Pol_1**: With two accounts of same bank one can transfer money between them.

\[
\text{Pol}_1 \equiv a_1: \text{Account} \land a_2: \text{Account} \land a_1 \neq a_2 \land a_1.\text{myBank} = a_2.\text{myBank} \land a_1.\text{balance} = b_1 > \text{amt} \land a_2.\text{balance} = b_2 \land _{\text{Call}}(a_1,\text{transfer},a_2,\text{amt})
\]
Formalizing **Pol_1**

**Pol_1**: With two accounts of same bank one can transfer money between them.

\[
\text{Pol}_1 \equiv a1: \text{Account} \land a2: \text{Account} \land a1 \neq a2 \land \\
a1.\text{myBank} = a2.\text{myBank} \land \\
a1.\text{balance} = b1 > \text{amt} \land \\
a2.\text{balance} = b2 \land \\
_.\text{Call}(a1,\text{transfer},a2,\text{amt}) \land \\
\rightarrow \\
\bullet \( a1.\text{balance} = b1 - \text{amt} \land \\
a2.\text{balance} = b2 + \text{amt} \)
\]
Formalizing Pol_2

**Pol_2**: Only someone with the Bank of a given currency can violate conservation of that currency

*This says:* If some execution which starts now and which involves at most the objects from $S$ modifies $b$.currency at some future time, then at least one of the objects in $S$ can access $b$ directly now, and this object is not internal to $b$. 
Formalizing Pol_2

- **Pol_2**: Only someone with the Bank of a given currency can violate conservation of that currency

\[
\text{Pol}_2 \triangleq \ b: \text{Bank} \land \\
\bullet \ (\ \text{Changes}(b.\text{currency})) \ @ \ S
\]

*This says:* If some execution which starts now and which involves at most the objects from \( S \) modifies \( b.\text{currency} \) at some future time, then at least one of the objects in \( S \) can access \( b \) directly now, and this object is not internal to \( b \).
Formalizing Pol_2

Pol_2: Only someone with the Bank of a given currency can violate conservation of that currency

\[ \text{Pol}_2 \equiv b: \text{Bank} \land (\text{Changes}(b.\text{currency})) \rightarrow \exists o \in S. [\text{Access}(o,b) \land o \notin \text{Internal}(b) ] \]

*This says:* If some execution which starts now and which involves at most the objects from \( S \) modifies \( b.\text{currency} \) at some future time, then at least one of the objects in \( S \) can access \( b \) directly now, and this object is not internal to \( b \).
Formalizing Pol_2 -2

- **Pol_2**: Only someone with the Bank of a given currency can violate conservation of that currency

\[
\text{Pol}_2 \triangleq b: \text{Bank} \land \left( \bullet ( \text{Changes}(b.\text{currency})) \right) \mathbin{@} S \rightarrow \exists o \in S. [ \text{Access}(o,b) \land o \notin \text{Internal}(b) ]
\]

equivalent to

\[
b: \text{Bank} \land \left( \forall o \in S. [ \neg \text{Access}(o,b) \lor o \in \text{Internal}(b) ] \right) \rightarrow \neg \left( \bullet ( \text{Changes}(b.\text{currency})) \right) \mathbin{@} S
\]
Formalizing Pol_2 -3

Pol_2: Only someone with the Bank of a given currency can violate conservation of that currency

Pol_2 reformulated

\[ \text{b:Bank} \land \forall o \in S. [ \text{Access}(o, b) \rightarrow o \in \text{Internal}(b) ] \rightarrow \neg (\bullet (\text{Changes}(b.\text{currency}) ) @ S ) \]

*This says:* A set S whose elements have direct access to b only if they are internal to b is insufficient to modify b.currency.
Internal
MBA1:
Internal

MBA1:

\[
\text{Internal}(b) \equiv \{ b \} \cup \{ a \mid a: \text{Account} \land a.\text{myBank} = b \}
\]

MBA2:
MBA1:

$\text{Internal}(b) \equiv \{ b \} \cup \{ a \mid a:\text{Account} \land a.\text{myBank} = b \}$

MBA2:
MBA1:

\[
\mathbf{\text{Internal}}(b) \equiv \\
\{ b \} \cup \{ a \mid a\text{:Account} \land a.\text{myBank} = b \}
\]

MBA2:

\[
\mathbf{\text{Internal}}(b) \equiv \\
\{ b \} \cup \{ n \mid n\text{:Node} \land \exists k. b.\text{book}^k.\text{next}=n \} \cup \\
\{ n \mid a\text{:Account} \land \exists k. b.\text{book}^k.\text{myAccount}=a \}
\]
MBA1:

\[
\text{Internal}(b) \equiv \{ b \} \cup \{ a \mid a:\text{Account} \land a.\text{myBank} = b \}
\]

MBA2:

\[
\text{Internal}(b) \equiv \{ b \} \cup \{ n \mid n:\text{Node} \land \exists k. \text{b.book}^k.\text{next}=n \} \cup \{ n \mid a:\text{Account} \land \exists k. \text{b.book}^k.\text{myAccount}=a \}
\]
Guarantees of \( \text{Pol}_2 \) in MBA1s

\[
\text{Pol}_2 \equiv b:\text{Bank} \land \\
\forall o \in S. [ \neg \text{Access}(o,b) \lor o \in \text{Internal}(b) ] \\
\rightarrow \\
\neg (\Diamond (\text{Changes}(b.\text{currency})) @ S)
\]
Guarantees of Pol_2 in MBA1s

\[
\text{Pol}_2 \equiv \text{b:Bank } \land \\
\forall o \in S. [ \neg \text{Access}(o, b) \lor o \in \text{Internal}(b) ] \\
\rightarrow \\
\neg (\bullet (\text{Changes}(b.\text{currency})) @ S)
\]
Guarantees of Pol_2 in MBA1s

\[ \text{Pol}_2 \equiv b: \text{Bank} \land \forall o \in S. [ \neg \text{Access}(o,b) \lor o \in \text{Internal}(b) ] \rightarrow \neg (\bullet (\text{Changes}(b.\text{currency}) ) \ @ S ) \]

1 is not on stack, and execution involves at most 1,2,3,4,20,21
Guarantees of Pol_2 in MBA1s

Pol_2 = \text{b:Bank} \land \\
\forall o \in S. [ \neg \text{Access}(o, b) \lor o \in \text{Internal}(b) ] \\
\rightarrow \\
\neg ( \bullet ( \text{Changes}(b.\text{currency}) ) \) @ S \\

1 is not on stack, and execution involves at most 1,2,3,4,20,21 \\
\rightarrow no change in 1.\text{currency}
Guarantees of Pol_2 in MBA2s

\[ \text{Pol}_2 \equiv b: \text{Bank} \land \forall o \in S. [ \neg \text{Access}(o, b) \lor o \in \text{Internal}(b) ] \rightarrow \neg (\bullet (\text{Changes}(b.\text{currency}) ) @ S ) \]

1 is not on stack, and execution involves at most 1, 2, 3, 4, 5, 6, 7, 20, 21;

\[ \rightarrow \text{no change in 1.currency} \]
Absence of Guarantee of Pol_2 in MBA1s

Pol_2 ≡ b:Bank ∧

∀o ∈ S. [ ¬ Access(o, b) ∨ o ∈ Internal(b) ]

→

¬ ( • ( Changes(b.currency) ) @ S )

1 is not on stack, and execution involves at most 1, 2, 3, 4, 10, 11;
Absence of Guarantee of Pol_2 in MBA1s

Pol_2 \equiv b:Bank \land \\
\forall o \in S. [ \neg \text{Access}(o,b) \lor o \in \text{Internal}(b) ] \rightarrow \\
\neg ( \bullet ( \text{Changes}(b.\text{currency})) @ S )

1 is not on stack, and execution involves at most 1,2,3,4,10,11; 
change in 1.\text{currency} possible
Absence of Guarantee of Pol_2 in MBA2

\[ \text{Pol}_2 \equiv \text{b:Bank} \land \forall o \in S. [ \neg \text{Access}(o,\text{b}) \lor o \in \text{Internal}(\text{b}) ] \rightarrow \neg (\bullet (\text{Changes}(\text{b.currency}) ) \ @ \ S ) \]

1 is not on stack, and execution involves at most 1,2,3,4,10,11; change in 1.currency possible
Formalizing Pol_4

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

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

Pol_4 \equiv a : \text{Account} \land \neg ( \text{Changes}(a.\text{balance}) ) @ S
Formalizing Pol_4

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

\[
\text{Pol}_4 \equiv \quad a:\text{Account} \quad \land \\
\quad \bullet ( \text{Changes}(a.\text{balance}) ) \quad @ \quad S \\
\quad \rightarrow \\
\quad \exists o \in S. [ \text{Access}(o,a) \quad \land \quad o \notin \text{Internal}(a) ]
\]
Guarantees of Pol_4 in MBA2s

Pol_2  ≡  a:Account ∧ a∉FrameVals ∧ 
• ( Changes(a.balance) ) @ S 
  → 
  ∃o∈S. [ Access(o,a) ∧ o∉Internal(a) ]

2 not on stack, and execution involves at most 1,2,3,4,5,6,7,20,21; 
——> no change in 2.balance
Giving a meaning to our Assertions
Giving a meaning to our Assertions

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

module $M : \text{Ident} \longrightarrow \text{ClassDef} \cup \text{PredicateDef} \cup \text{FunctionDef}$

configuration $\sigma : \text{Heap} \times \text{Continuations} \times \text{Expression}$

execution $M, \sigma \rightarrow \sigma'$
Giving a meaning to our Assertions

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

module \( M \) : Ident \( \rightarrow \) ClassDef \( \cup \) PredicateDef \( \cup \) FunctionDef

configuration \( \sigma \) : Heap \( \times \) Continuations \( \times \) Expression

execution \( M, \sigma \rightsquigarrow \sigma' \)

Define module concatenation * so that
\( M \ast M' \) undefined, iff \( \text{dom}(M) \cap \text{dom}(M') \neq \emptyset \)

otherwise
\( M \ast M'(id) = M(id) \) if \( M'(id) \) undefined, else \( M'(id) \)
Giving a meaning to our Assertions

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

module \( M : \text{Ident} \rightarrow \text{ClassDef} \cup \text{PredicateDef} \cup \text{FunctionDef} \)
configuration \( \sigma : \text{Heap} \times \text{Continuations} \times \text{Expression} \)
execution \( M, \sigma \rightarrow \sigma' \)

Define module concatenation * so that
\( M^*M' \) undefined, iff \( \text{dom}(M) \cap \text{dom}(M') \neq \emptyset \)
otherwise
\( M^*M'(\text{id}) = M(\text{id}) \) if \( M'(\text{id}) \) undefined, else \( M'(\text{id}) \)

We will define \( M, \sigma \models A \)
Initial(\( \sigma \)) and Arising(\( M \))
\( M \models A \)
Giving a meaning to our Assertions

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

module \( M : \text{Ident} \rightarrow \text{ClassDef} \cup \text{PredicateDef} \cup \text{FunctionDef} \)

configuration \( \sigma : \text{Heap} \times \text{Continuations} \times \text{Expression} \)

execution \( M, \sigma \rightarrow \sigma' \)

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 \( M'(id) \)

**Lemma** \( M^*M' = M'^*M \)

We will define \( M, \sigma \models A \)
Initial(\( \sigma \)) and Arising(\( M \))
\( M \models A \)
Expressions and Assertions - reminder

\[
e := \text{this} \mid x \mid e.fld \mid \text{func}(e_1,\ldots,e_n) \mid \ldots
\]

\[
A := e > e \mid e = e \mid P(e_1,\ldots,e_n) \mid \ldots
\]

\[
\mid A \rightarrow A \mid A \land A \mid \exists x. A \mid \ldots
\]

| Access(e,e’) | permission |
| Changes(e) | authority |
| •A | ○A | time |
| A @ S | space |
| Call(x,m,x_1,\ldots,x_n) | call |
Giving a meaning to Expressions

\[ e ::= \text{this} | \ x | \ e.fld \quad | \ \text{func}(e1,\ldots,\text{en}) \quad | \ \ldots \]

Define \[ \mathcal{L}_{M,\sigma} \] as expected
Giving a meaning to Expressions

\[ e ::= \text{this} \mid x \mid e.\text{fld} \mid \text{func}(e_1, \ldots, e_n) \mid \ldots \]

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

Eg, with \( x \) mapping to 3, we have \( \llbracket x.\text{myBank.book.next} \rrbracket_{M,\sigma} = 6 \)
Giving a meaning to Assertions

“Conventional part”

\[ A ::= e>e \quad \mid \quad A \rightarrow A \quad \mid \quad P(e1,..en) \quad \mid \quad \exists x.A \quad \mid \quad ... \]
Giving a meaning to Assertions

“Conventional part”

\[ A ::= e > e \mid A \rightarrow A \mid P(e_1,..e_n) \mid \exists x.A \mid \ldots \]

We define \( M, \sigma \models A \)
Giving a meaning to Assertions

“Conventional part”

\[ A ::= \ e > e \mid A \rightarrow A \mid P(e_1,..e_n) \mid \exists x. A \mid \ldots \]

We define \( M, \sigma \models A \)

\[
M,\sigma \models e > e' \iff \underline{e} \downarrow_{M,\sigma} > \underline{e'} \downarrow_{M,\sigma}
\]

\[
M,\sigma \models A \rightarrow A' \iff M,\sigma \models A \text{ implies } M,\sigma \models A'
\]

\[
M,\sigma \models P(e_1,..e_n) \iff M,\sigma \models M(P)[x_1 \mapsto \underline{e_1} \downarrow_{M,\sigma}, \ldots, x_n \mapsto \underline{e_n} \downarrow_{M,\sigma}]
\]

\[
M,\sigma \models \exists x. A \iff M,\sigma[z \mapsto i] \models A[x \mapsto z]
\]

for some \( i \in \text{dom}(\sigma.\text{heap}) \), and \( z \) free in \( A \)
Giving meaning to Assertions

“Unconventional part”

\[ A ::= \text{Access}(x,y) \mid \text{Changes}(e) \mid \bullet A \mid \circ A \mid A \oplus S \mid \text{Call}(x,m,x_1,..x_n) \]
Giving meaning to Assertions

“Unconventional part”

\[ A ::= \text{Access}(x,y) \mid \text{Changes}(e) \mid \bullet A \mid \circ A \mid A @ S \mid \text{Call}(x,m,x_1,..x_n) \]

\[ M,\sigma \models \text{Access}(e,e') \text{ iff } \mathcal{L}.\text{fld}_M,\sigma = \mathcal{L}.e'_M,\sigma \text{ for some field fld} \lor \mathcal{L}.\text{this}_M,\sigma = \mathcal{L}.e'_M,\sigma \land \mathcal{L}y_M,\sigma = \mathcal{L}.e'_M,\sigma \]

\[ \land y \text{ is a parameter of the current function} \]

\[ M,\sigma \models \text{Changes}(e) \text{ iff } M,\sigma \leadsto \sigma' \land \mathcal{L}.e_M,\sigma \neq \mathcal{L}.e'_M,\sigma' \]

\[ M,\sigma \models \bullet A \text{ iff } \exists \sigma',\sigma'',\phi.[ \sigma = \sigma'.\phi \land M,\phi \leadsto^{*} \sigma'' \land M,\sigma'' \models A ] \]

\[ M,\sigma \models \circ A \text{ iff } \forall \sigma_0.[ \text{Initial}(\sigma_0) \land M,\sigma_0 \leadsto^{*} \sigma \rightarrow \exists \sigma_1.( M,\sigma_0 \leadsto^{*} \sigma_1 \land M,\sigma_1 \leadsto^{+} \sigma \land M,\sigma_1 \models A ) ] \]

\[ M,\sigma \models A@S \text{ iff } M,\sigma@I_S \models A \text{ where } I_S = \mathcal{L}S_M,\sigma \text{ and } \sigma@I_S \ldots \]

\[ M,\sigma \models x.\text{Call}(y,m,z_1,..z_n) \text{ iff } \mathcal{L}.\text{this}_M,\sigma = \mathcal{L}.x_M,\sigma \land \ldots \]
Giving meaning to Assertions
- the full truth -

\[ M, \sigma \vDash \text{Access}(e, e') \iff \ldots \text{as before}\ldots \]

\[ M, \sigma \vDash \text{Changes}(e) \iff M, \sigma \rightarrow \sigma' \land \{ e \}_{M, \sigma} \neq \{ e[Z \mapsto y] \}_{M, \sigma'[y \mapsto \sigma(z)]} \]

\[ \text{where } \{ z \} = \text{Free}(e) \land y \text{ fresh in } e, \sigma, \sigma' \]

\[ M, \sigma \vDash \bullet A \iff \exists \sigma', \sigma'', \phi. [ \sigma = \sigma'. \phi \land M, \phi \rightarrow^* \sigma' \land \]

\[ M, \sigma'[y \mapsto \sigma(z)] \vDash A[z \mapsto y] \] \]

\[ \text{where } \{ z \} = \text{Free}(A) \land y \text{ fresh in } A, \sigma, \sigma' \]

\[ M, \sigma \vDash \circ A \iff \forall \sigma_0. [ \text{Initial}(\sigma_0) \rightarrow \]

\[ \exists \sigma_1. ( M, \sigma_0 \rightarrow^* \sigma_1 \land M, \sigma_1 \rightarrow^+ \sigma \land \]

\[ M, \sigma_1[y \mapsto \sigma(z)] \vDash A[z \mapsto y] ] \]

\[ \text{where } \{ z \} = \text{Free}(A) \land y \text{ fresh in } A, \sigma_1, \sigma \]

\[ M, \sigma \vDash A@S \iff \ldots \text{as before}\ldots \]

\[ M, \sigma \vDash x.\text{Call}(y, m, z_1, \ldots, z_n) \iff \ldots \text{as before}\ldots \]
Giving meaning to Assertions

- the full truth -

\[ M, \sigma \models \text{Access}(e, e') \iff \ldots \text{as before} \ldots \]

\[ M, \sigma \models \text{Changes}(e) \iff M, \sigma \rightarrow \sigma' \land \mathcal{L}_M, \sigma \not= \mathcal{L}_e[z \mapsto y] \]

where \{z\} = \text{Free}(e) \land y \text{ fresh in } e, \sigma, \sigma'

\[ M, \sigma \models \bigcirc A \iff \exists \sigma', \sigma'', \phi.[ \sigma = \sigma'. \phi \land M, \phi \sim^* \sigma' \land \]

\[ M, \sigma' [y \mapsto \sigma(z)] \models A[z \mapsto y] \]

where \{z\} = \text{Free}(A) \land y \text{ fresh in } A, \sigma, \sigma'

\[ M, \sigma \models \circlearrowleft A \iff \forall \sigma_0.[ \text{Initial}(\sigma_0) \rightarrow \]

\[ \exists \sigma_1.( M, \sigma_0 \rightarrow^* \sigma_1 \land M, \sigma_1 \rightarrow^+ \sigma \land \]

\[ M, \sigma_1[y \mapsto \sigma(z)] \models A[z \mapsto y] \]

where \{z\} = \text{Free}(A) \land y \text{ fresh in } A, \sigma_1, \sigma

\[ M, \sigma \models A \circ S \]

\[ M, \sigma \models x.\text{Call}(y, m, z_1, \ldots z_n) \iff \ldots \text{as before} \ldots \]
Giving meaning to Assertions
- the full truth -

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

\( M, \sigma \models \text{Changes}(e) \) iff \( M, \sigma \vdash \sigma' \land \not M, \sigma \models e[z \mapsto y] \models M, \sigma'[y \mapsto \sigma(z)] \)

where \( \{z\} = \text{Free}(e) \land y \) fresh in \( e, \sigma, \sigma' \)

\( M, \sigma \models \circ A \) iff \( \exists \sigma', \sigma'', \phi. [ \sigma = \sigma'. \phi \land M, \phi \vdash^* \sigma' \land \\
M, \sigma'[y \mapsto \sigma(z)] \models A[z \mapsto y] ] \)

where \( \{z\} = \text{Free}(A) \land y \) fresh in \( A, \sigma, \sigma' \)

\( M, \sigma \models A \oplus S \) iff \( \ldots \) as before \( \ldots \)

\( M, \sigma \models x. \text{Call}(y,m,z_1,..z_n) \) iff \( \ldots \) as before \( \ldots \)
Giving meaning to Assertions

- the full truth -

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

\[ M, \sigma \models \text{Changes}(e) \iff M, \sigma \overset{\rightarrow}{\rightarrow} \sigma' \land e \not\in e_{M, \sigma} \land e_{M, \sigma[y \mapsto \sigma(z)]} \]

where \{z\} = \text{Free}(e) \land y \text{ fresh in } e, \sigma, \sigma'

\[ M, \sigma \models \bullet A \iff \exists \sigma', \sigma'', \phi. [\ \sigma = \sigma'. \phi \land M, \phi \overset{\rightarrow}{\rightarrow} \sigma' \land \]

\[ M, \sigma'[y \mapsto \sigma(z)] \models A[z \mapsto y] \]

where \{z\} = \text{Free}(A) \land y \text{ fresh in } A, \sigma, \sigma'

\[ M, \sigma \models \circ A \iff \forall \sigma_0. [\ \text{Initial}(\sigma_0) \rightarrow \]

\[ \exists \sigma_1. ( M, \sigma_0 \overset{\rightarrow}{\rightarrow} \sigma_1 \land M, \sigma_1 \overset{\rightarrow}{\rightarrow} \sigma \land \]

\[ M, \sigma_1[y \mapsto \sigma(z)] \models A[z \mapsto y] \]

where \{z\} = \text{Free}(A) \land y \text{ fresh in } A, \sigma_1, \sigma

\[ M, \sigma \models A@S \iff \text{... as before ...} \]

\[ M, \sigma \models x. \text{Call}(y, m, z_1, \ldots, z_n) \iff \text{... as before ...} \]
Giving meaning to Assertions
- the full truth -

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

\[ M, \sigma, \sigma' \models \text{Changes}(e) \iff M, \sigma \rightarrow \sigma' \wedge \nexists \ell \in M, \sigma \neq \ell[e[z\mapsto y]]_{M, \sigma', y\mapsto \sigma(z)} \]

where \( \{z\} = \text{Free}(e) \wedge y \text{ fresh in } e, \sigma, \sigma' \)

\[ M, \sigma, \sigma' \models \bullet A \iff \exists \sigma', \sigma'', \phi. [ \sigma = \sigma'. \phi \wedge M, \phi \leadsto \sigma' \wedge \]

\[ M, \sigma'[y\mapsto \sigma(z)] \models A[z\mapsto y] \]

where \( \{z\} = \text{Free}(A) \wedge y \text{ fresh in } A, \sigma, \sigma' \)

\[ M, \sigma, \sigma_1 \models \circ A \iff \forall \sigma_0. [ \text{Initial}(\sigma_0) \rightarrow \]

\[ \exists \sigma_1. ( M, \sigma_0 \leadsto \sigma_1 \wedge M, \sigma_1 \leadsto+ \sigma \wedge \]

\[ M, \sigma_1[y\mapsto \sigma(z)] \models A[z\mapsto y] \]

where \( \{z\} = \text{Free}(A) \wedge y \text{ fresh in } A, \sigma_1, \sigma \)

\[ M, \sigma \models A@S \iff \text{... as before ...} \]

\[ M, \sigma \models x.\text{Call}(y, m, z_1, \ldots, z_n) \iff \text{... as before ...} \]
Giving meaning to Assertions

\[ M, \sigma ⊨ Access(x,y) \iff \text{Initial} > \mathcal{L}y_{M,\sigma} \lor \mathcal{L}x.f_{M,\sigma} > \mathcal{L}y_{M,\sigma} \lor \mathcal{L}\text{this}_{M,\sigma} = \mathcal{L}x_{M,\sigma} \land \mathcal{L}y_{M,\sigma} = \mathcal{L}z_{M,\sigma} \land \ldots \]

\[ M, \sigma ⊨ \text{Changes}(e) \iff M, \sigma \rightsquigarrow \sigma' \land \mathcal{L}e_{M,\sigma} \neq \mathcal{L}e_{M,\sigma'} \]

\[ M, \sigma ⊨ \mathbb{A} \iff \exists \sigma'.[ M, \sigma \rightsquigarrow \sigma' \land M, \sigma' ⊨ A ] \]

\[ M, \sigma ⊨ \Diamond A \iff \exists \sigma_0, \sigma_1.[ \text{Initial}(\sigma_0) \land M, \sigma_0 \rightsquigarrow* \sigma_1 \land M, \sigma_1 \rightsquigarrow* \sigma \land M, \sigma_1 ⊨ A ] \]

\[ M, \sigma ⊨ A@S \iff M, \sigma@ls ⊨ A \text{ where } \text{ls} = \mathcal{L}S_{M,\sigma} \text{ and } \sigma@ls \ldots \]

\[ M, \sigma ⊨ \text{Call}(x,m,x_1,..x_n) \iff \mathcal{L}\text{this}_{M,\sigma} = \mathcal{L}x_{M,\sigma} \land \ldots \]
Giving meaning to Assertions

\( M, \sigma \models \text{Access}(x, y) \iff \text{Initial} > \downarrow y_{M, \sigma} \lor \downarrow x.f_{M, \sigma} > \downarrow y_{M, \sigma} \lor \downarrow \text{this}_{M, \sigma} = \downarrow x_{M, \sigma} \land \downarrow y_{M, \sigma} = \downarrow z_{M, \sigma} \land \ldots \)

\( M, \sigma \models \text{Changes}(e) \iff M, \sigma \rightarrow \sigma' \land \downarrow e_{M, \sigma} \neq \downarrow e_{M, \sigma'} \)

\( M, \sigma \models \bullet A \iff \exists \sigma'.[ M, \sigma \rightarrow \sigma' \land M, \sigma' \models A ] \)

\( M, \sigma \models \circ A \iff \exists \sigma_0, \sigma_1.[ \text{Initial}(\sigma_0) \land M, \sigma_0 \rightarrow^{*} \sigma_1 \land M, \sigma_1 \rightarrow^{*} \sigma \land M, \sigma_1 \models A ] \)

\( M, \sigma \models A@S \iff M, \sigma@ls \models A \text{ where } ls = \downarrow S_{M, \sigma} \text{ and } \sigma@ls \ldots \)

\( M, \sigma \models \text{Call}(x, m, x_1, \ldots, x_n) \iff \downarrow \text{this}_{M, \sigma} = \downarrow x_{M, \sigma} \land \ldots \)
Giving meaning to Assertions

\[ M, \sigma \models \text{Access}(x,y) \iff \text{Initial} > \downarrow y \downarrow_{M,\sigma} \lor \downarrow x.f \downarrow_{M,\sigma} > \downarrow y \downarrow_{M,\sigma} \lor \downarrow \text{this} \downarrow_{M,\sigma} = \downarrow x \downarrow_{M,\sigma} \land \downarrow y \downarrow_{M,\sigma} = \downarrow z \downarrow_{M,\sigma} \land \ldots \]

\[ M, \sigma \models \text{Changes}(e) \iff M, \sigma \leadsto \sigma' \land \downarrow e \downarrow_{M,\sigma} \neq \downarrow e \downarrow_{M,\sigma} \]

\[ M, \sigma \models \bullet A \iff \exists \sigma'. [ M, \sigma \leadsto \sigma' \land M, \sigma' \models A ] \]

\[ M, \sigma \models \circ A \iff \exists \sigma_0, \sigma_1. [ \text{Initial}(\sigma_0) \land M, \sigma_0 \leadsto^* \sigma_1 \land M, \sigma_1 \leadsto^* \sigma \land M, \sigma_1 \models A ] \]

\[ M, \sigma \models A@S \iff M, \sigma@ls \models A \text{ where } ls = \downarrow S \downarrow_{M,\sigma} \text{ and } \sigma@ls \ldots \]

\[ M, \sigma \models \text{Call}(x,m,x_1,\ldots,x_n) \iff \downarrow \text{this} \downarrow_{M,\sigma} = \downarrow x \downarrow_{M,\sigma} \land \ldots \]
\( M, \sigma \models \text{Access}(x,y) \) iff \( \text{Initial} > \llbracket y \rrbracket_{M, \sigma} \lor \llbracket x \cdot f \rrbracket_{M, \sigma} > \llbracket y \rrbracket_{M, \sigma} \lor \llbracket \text{this} \rrbracket_{M, \sigma} = \llbracket x \rrbracket_{M, \sigma} \land \llbracket y \rrbracket_{M, \sigma} = \llbracket z \rrbracket_{M, \sigma} \land \ldots \)

\( M, \sigma \models \text{Changes}(e) \) iff \( M, \sigma \rightarrow \sigma' \land \llbracket e \rrbracket_{M, \sigma} \neq \llbracket e \rrbracket_{M, \sigma'} \)

\( M, \sigma \models \bullet A \) iff \( \exists \sigma'. [ M, \sigma \rightarrow \sigma' \land M, \sigma' \models A ] \)

\( M, \sigma \models \circ A \) iff \( \exists \sigma_0, \sigma_1. [ \text{Initial}(\sigma_0) \land M, \sigma_0 \rightarrow^* \sigma_1 \land M, \sigma_1 \rightarrow^* \sigma \land M, \sigma_1 \models A ] \)

\( M, \sigma \models A@S \) iff \( M, \sigma@\mathbf{ls} \models A \) where \( \mathbf{ls} = \llbracket S \rrbracket_{M, \sigma} \) and \( \sigma@\mathbf{ls} \ldots \)

\( M, \sigma \models \text{Call}(x,m,x_1,..x_n) \) iff \( \llbracket \text{this} \rrbracket_{M, \sigma} = \llbracket x \rrbracket_{M, \sigma} \land \ldots \)
A runtime configuration is initial iff
1) The heap contains only one object, of class Object
2) The continuation consists of just one frame, where this points to that object.

Note: The expression can be arbitrary

$$\text{Initial}(\sigma) \text{ iff } \sigma.\text{heap} = (1 \mapsto (\text{Object}, \ldots)) \land \sigma.\text{continuations} = (\text{this} \mapsto 1).$$
outstanding definitions: @
σ@ls = ( σ.heap@ls, σ.continuations, σ.expression)
\[ \sigma@ls = (\sigma.heap@ls, \sigma.continuations, \sigma.expression) \]

where \( \text{dom}(hp.@IS) = IS \) and \( \forall \alpha \in S. \ hp@IS(\alpha) = hp(\alpha) \)
σ@Is = (σ.heap@Is, σ.continuations, σ.expression)

where \(\text{dom}(hp.@IS) = IS\) and \(\forall \alpha \in S. \ hp@IS(\alpha) = hp(\alpha)\)

eg, given hp:
\[ \sigma@Is = (\sigma.heap@Is, \sigma.continuations, \sigma.expression) \]

where \( \text{dom}(hp.@IS) = Is \) and \( \forall \alpha \in S. \ hp@IS(\alpha) = hp(\alpha) \)

eg, given \( hp: \)

\[ hp@\{1,2,4,10,20,21\}: \]
σ@Is = (σ.heap@Is, σ.continuations, σ.expression)

where \( \text{dom}(hp.@IS) = IS \) and \( \forall \alpha \in S. \ hp@IS(\alpha) = hp(\alpha) \)

eg, given \( hp \):

\[
\begin{align*}
1: Ba & \quad 10 \quad 11 \\
2: Ac & \rightarrow 3: Ac \\
4: Ac & \rightarrow 20 \quad 21
\end{align*}
\]

\[
\begin{align*}
1: Ba & \rightarrow 10 \\
2: Ac & \rightarrow 3: Ac \rightarrow 4: Ac \rightarrow 20 \quad 21
\end{align*}
\]

\( hp@\{1,2,4,10,20,21\} \):
Giving meaning to Assertions

\[ M \models A \iff \forall \sigma \in \text{Arising}(M*M'). M*M', \sigma \models A \]

A module \( M \) satisfies an assertion \( A \) if all runtime configurations \( \sigma \) which arrive from execution of code from \( M*M' \) (for any module \( M' \)), satisfy \( A \).
Arising(\(M\)) = \{ \sigma \mid \exists M', \sigma_0. \ [\text{Initial}(\sigma_0) \land M^*M' \text{ is defined} \land M'^*M, \sigma_0 \rightarrow^* \sigma] \}
outstanding definitions: Arising

\[ \text{Arising}(M) = \{ \sigma \mid \exists M', \sigma_0. \ [ \text{Initial}(\sigma_0) \land M^*M' \text{ is defined} \land M'^*M, \sigma_0 \rightarrow^* \sigma ] \} \]
outstanding definitions: Arising

$$\text{Arising}(M) = \{ \sigma \mid \exists M',\sigma_0. \ [(\text{Initial}(\sigma_0) \land M \ast M' \text{ is defined}) \land M' \ast M, \sigma_0 \rightarrow^* \sigma] \}$$

E.g., Arising(MBA2s).heap and Arising(MBA2).heap contain:
Arising(M) = \{ \sigma \mid \exists M', \sigma_0. \ [ \text{Initial}(\sigma_0) \land M*M' \text{ is defined} \land M'*M, \sigma_0 \rightarrow^* \sigma ] \} \\

Also, Arising(MBA2s).heap and Arising(MBA2).heap contain:
Arising(\(M\)) = \{ \sigma \mid \exists M', \sigma_0. \left[ \text{Initial}(\sigma_0) \land M^*M' \text{ is defined} \land M'^*M, \sigma_0 \xrightarrow{*} \sigma \right] \}
Arising(M) = \{ \sigma \mid \exists M', \sigma_0. [ Initial(\sigma_0) \land M^*M' \text{ is defined} \land M'^*M, \sigma_0 \rightarrow^* \sigma ] \}

But the following is in Arising(MBA2).heap but is not in Arising(MBA2s).heap
Giving meaning to Assertions

\[ M \models A \text{ iff } \forall \sigma \in \text{Arising}(M^*M'). \quad M^*M', \sigma \models A \]
Giving meaning to Assertions

\[ M \models A \iff \forall \sigma \in \text{Arising}(M^*M'). \ M^*M', \ \sigma \models A \]

“Lemma”
Giving meaning to Assertions

\[ M \models A \iff \forall \sigma \in \text{Arising}(M^*M'). \quad M^*M', \sigma \models A \]

“Lemma”

- MBA1s \models Pol_1
- MBA1s \models Pol_2
- MBA1s \models Pol_4
- MBA2s \models Pol_1
- MBA2s \models Pol_2
- MBA2s \models Pol_4

Proof sketches are “holistic”.

Proof sketches use more framing notions, and require frames are self-framing.
Entailments

Definitions

- \( M \models A \subseteq A' \) iff \( \forall \sigma \in \text{Arising}(M). [ M, \sigma \models A \rightarrow M, \sigma \models A' ] \)
- \( M \models A \varnothing A' \) iff \( M \models A \rightarrow M \models A' \)
Entailments

Definitions

- $M \models A \subset A'$ iff $\forall \sigma \in \text{Arising}(M). \ [ M, \sigma \models A \rightarrow M, \sigma \models A' ]$
- $M \models A \sqsubseteq A'$ iff $M \models A \rightarrow M \models A'$

Facts

- $M \models A \subset A'$ implies $M \models A \sqsubseteq A'$
- $M \models A \sqsubseteq A'$ does not imply $M \models A \subset A'$
- $M \models (\Diamond A \rightarrow A') \sqsubseteq (A' \rightarrow \Box A)$
Entailments

Definitions

- $M \vdash A \subseteq A'$ iff $\forall \sigma \in \text{Arising}(M). [ M, \sigma \vdash A \rightarrow M, \sigma \vdash A' ]$
- $M \vdash A \subseteq A'$ iff $M \vdash A \rightarrow M \vdash A'$

Facts

- $M \vdash A \subseteq A'$ implies $M \vdash A \subseteq A'$
- $M \vdash A \subseteq A'$ does not imply $M \vdash A \subseteq A'$
- $M \vdash (\bullet A \rightarrow A') \subseteq (A' \rightarrow \circ A)$

- $M \vdash A @ S$ and $M \vdash S \subseteq S'$ imply $M \vdash A @ S'$
- $M, \sigma \vdash (\bullet A)@S$ imply $M \vdash (\bullet (A@S))$
Entailments

Definitions

- \( M \models A \subseteq A' \) iff \( \forall \sigma \in \text{Arising}(M). [ M, \sigma \models A \rightarrow M, \sigma \models A' ] \)
- \( M \models A \sqsubseteq A' \) iff \( M \models A \rightarrow M \models A' \)

Facts

- \( M \models A \subseteq A' \) implies \( M \models A \subseteq A' \)
- \( M \models A \subseteq A' \) does not imply \( M \models A \subseteq A' \)
- \( M \models ( \bullet A \rightarrow A' ) \sqsubseteq ( A' \rightarrow \circ A ) \)
Entailments

Definitions

- \( M \models A \subseteq A' \) iff \( \forall \sigma \in \text{Arising}(M). [ M, \sigma \models A \rightarrow M, \sigma \models A' ] \)
- \( M \models A \sqsubseteq A' \) iff \( M \models A \rightarrow M \models A' \)

Facts

- \( M \models A \subseteq A' \) implies \( M \models A \sqsubseteq A' \)
- \( M \models A \sqsubseteq A' \) does not imply \( M \models A \subseteq A' \)
- \( M \models ( \bullet A \rightarrow A' ) \sqsubseteq ( A' \rightarrow \circ A ) \)
- \( M \models A @ S \) and \( M \models S \subseteq S' \) does not imply \( M \models A @ S' \)
- We call \( A \) monotonic, if \( M,\sigma \models A @ S \) and \( M,\sigma \models S \subseteq S' \) imply \( M,\sigma \models A @ S' \)
- If \( A \) monotonic, then \( M,\sigma \models ( \bullet A ) @ S \) and \( M,\sigma \models S' = \text{Allocated} \) imply \( M,\sigma \models \bullet ( A @ ( S \cup \text{Allocated} \setminus S' ) ) \)
Example 2: DAO - simplified

DAO, a “hub that disperses funds”; (https://www.ethereum.org/dao). In a simplified form it allows clients to contribute and retrieve their funds (by calling `payIn(...)` and `repay()`).
Example 2: DAO - simplified

DAO, a “hub that disperses funds”; (https://www.ethereum.org/dao). In a simplified form it allows clients to contribute and retrieve their funds (by calling `payIn(...)` and `repay()`).

\[
\text{Pol}_{\text{DAO\_withdraw}} \equiv \\
\forall \text{cl:External.} \forall \text{d:DAO.} \forall \text{n':Nat.} \\
[ \text{cl.Calls(d.repay())} \land \circ (\text{cl.Calls(d.payIn(n))} \\
\land \neg (\circ \text{cl.Calls(d.repay())}) \\
\rightarrow \\
\text{d.ether} \geq n \land \bullet (\text{d.Calls(cl.send(n))}) ]
\]
Example2: DAO - simplified

DAO, a “hub that disperses funds”; (https://www.ethereum.org/dao).
In a simplified form it allows clients to contribute and retrieve their funds (by calling payIn(...) and repay()).

\[ Pol_{\text{DAO\_withdraw}} \]

\[
\forall \ cl:\text{External.} \ \forall d:\text{DAO.} \ \forall n':\text{Nat.} \\
[ \ cl.\text{Calls}(d.\text{repay}()) \ \land \circ ( cl.\text{Calls}(d.\text{payIn}(n)) ) \\
\land \neg( \circ cl.\text{Calls}(d.\text{repay}()) ) \\
\rightarrow \\
\quad d.\text{ether} \geq n \land \bullet (d.\text{Calls}(cl.\text{send}(n))) ]
\]

*This says:* If a client cl asks to be repaid (cl.Calls(d.repay()) and in the past they had contributed (\circ ( cl.Calls(d.payIn(n)) )) and not withdrawn their contribution (\neg ( \circ cl.Calls(d.repay()) )), then the DAO will have enough funds (d.ether \geq n) and will send the money to client (\bullet d.Calls(cl.send(n))).
Example 2: DAO continued

**Vulnerability**: Through repeated calls of a buggy version of `repay()`, a client could deplete all funds of the DAO and thus the DAO could not repay its other clients.
Example 2: DAO continued

**Vulnerability:** Through repeated calls of a buggy version of `repay()`, a client could deplete all funds of the DAO and thus the DAO could not repay its other clients.

\[
\text{Pol} \_\text{DAO} \_\text{withdraw} \\
\equiv \\
\forall \text{cl:External.} \forall \text{d:DAO.} \forall \text{n:Nat.} \\
[ \text{cl.Calls(d.repay())} \land \circ (\text{cl.Calls(d.payIn(n))})\\
\land \neg (\circ \text{cl.Calls(d.repay())}) \\
\to \\
\text{d.ether} \geq n \land \bullet (\text{d.Calls(cl.send(n))}) ]
\]
Vulnerability: Through repeated calls of a buggy version of repay(), a client could deplete all funds of the DAO and thus the DAO could not repay its other clients.

\[
\text{Pol}_{\text{DAO}}_{\text{withdraw}} \equiv \forall \text{cl:External.} \forall \text{d:DAO.} \forall \text{n:Nat.} \left[ \text{cl.Calls(d.repay())} \land \circ (\text{cl.Calls(d.payIn(n))} \land \neg (\circ \text{cl.Calls(d.repay())} \rightarrow \text{d.ether} \geq n \land \bullet (\text{d.Calls(cl.send(n))}) \right]
\]

This specification avoids the vulnerability: A contract which satisfies \text{Pol}_{\text{DAO}}_{\text{withdraw}} will always be able to repay all its customers.
Example 2: DAO continued

Vulnerability: Through repeated calls of a buggy version of `repay()`, a client could deplete all funds of the DAO and thus the DAO could not repay its other clients.

\[
\text{Pol\textsubscript{DAO\_withdraw}} \equiv \\
\forall \text{cl:External. } \forall \text{d:DAO. } \forall \text{n:Nat. } \\
[ \text{cl.Calls(d\_repay()) } \land \circ (\text{cl.Calls(d\_payIn(n))} ) \\
\land \neg (\circ \text{cl.Calls(d\_repay())} ) ] \\
\rightarrow \\
\text{d.ether} \geq \text{n} \land \bullet (\text{d.Calls(cl\_send(n))} )
\]

This specification avoids the vulnerability:
A contract which satisfies \text{Pol\textsubscript{DAO\_withdraw}} will always be able to repay all its customers.
Example 2: a possible classical spec

Assume that the DAO keeps a directory of contributions, and require:
R1: that the directory is compatible with the amount of ether kept in the DAO, and
R2: that withdraw reduces the ether but that amount.
Example2: a possible classical spec

Assume that the DAO keeps a directory of contributions, and require:
R1: that the directory is compatible with the amount of ether kept in the DAO, and
R2: that withdraw reduces the ether but that amount.

R1: \( \forall d: \text{DAO}. \quad d.\text{ether} = \sum_{cl \text{ such that } d.\text{directory}(cl) \text{ defined}} d.\text{directory}(cl) \)
Example2: a possible classical spec

Assume that the DAO keeps a directory of contributions, and require:
R1: that the directory is compatible with the amount of ether kept in the DAO, and
R2: that withdraw reduces the ether but that amount.

R1: \( \forall d: \text{DAO}. \quad d.\text{ether} = \sum_{cl} \text{such that } d.\text{directory}(cl) \text{ defined } d.\text{directory}(cl) \)

R2: \( \text{cl:External } \land \text{d:DAO } \land \text{n:Nat } \land \text{d.directory(cl)=n} \)
\( \{ d.\text{re}p\text{ay}() \land \text{caller=cl} \} \)
\( d.\text{directory(cl)=0 } \land d.\text{Calls(cl.send(n))} \)
Example2: a possible classical spec

Assume that the DAO keeps a directory of contributions, and require:
R1: that the directory is compatible with the amount of ether kept in the DAO, and
R2: that withdraw reduces the ether but that amount.

R1: ∀d:DAO. d.ether = ∑ cl such that d.directory(cl) defined d.directory(cl)

R2: cl:External ∧ d:DAO ∧ n:Nat ∧ d.directory(cl)=n
{ d.repay() ∧ caller=cl }
  d.directory(cl)=0 ∧ d.Calls(cl.send(n))

R2 says: If client cl has m tokens (d.directory(cl)=n) and asks to be repaid (cl calls d.repay()) then all his tokens will be sent (d.Calls(cl.send(n))) and no tokens will be left (d.directory(cl)=0).
Together with R2, this spec avoids the vulnerability, provided the attack goes through the function repay.
Example2: classical spec vs holistic spec

Assume that the DAO keeps a directory of contributions, and require:
R1: that the directory is compatible with the amount of ether kept in the DAO, and
R2: that withdraw reduces the ether but that amount.

R1: \( \forall d:\text{DAO}. \ d.\text{ether} = \sum_{cl \text{ such that } d.\text{directory}(cl) \text{ defined}} d.\text{directory}(cl) \)

R2: \( \text{cl:External} \land d:\text{DAO} \land n:\text{Nat} \land d.\text{directory}(cl)=n \)
{\( d.\text{repay}() \land \text{caller}=\text{cl} \) }
\( d.\text{directory}(cl)=0 \land d.\text{Calls}(\text{cl.send}(n)) \)

This classical specification is insufficient to avoid the vulnerability in general, as it does not prevent other functions from affecting the contents of \( d.\text{directory} \).

To avoid the vulnerability in general, we would need to either manually inspect the specification of all the functions in the DAO, or add another holistic spec, promising, eg that only calls by \( cl \) can affect the contents of \( d.\text{directory}(cl) \).
Example 3: ERC20 - simplified

a popular standard for initial coin offerings. (https://theethereum.wiki/w/index.php/ERC20_Token_Standard); allows clients to buy and transfer tokens, and to designate other clients to transfer on their behalf.
Example 3: ERC20 - simplified

a popular standard for initial coin offerings. (https://theethereum.wiki/w/index.php/ERC20_Token_Standard); allows clients to buy and transfer tokens, and to designate other clients to transfer on their behalf.

Pol_{ERC20\_withdraw}
\equiv
\forall e:\text{ERC20.} \forall cl:\text{Client.}\left[ e.\text{balance}(cl) < e.\text{balance}(cl)_{\text{pre}} \right)
\rightarrow
\left[ \circ( \exists cl': \text{Client.} \exists m: \text{Nat.}\left[ cl.\text{Calls}(e.\text{transfer}(cl',m)) \right) \lor \exists cl'': \text{Client.} \text{Authorized}(c, cl'') \land c''.\text{Calls}(e.\text{transferFrom}(c, cl', m)) \right) \right]
Example 3: ERC20 - simplified

a popular standard for initial coin offerings. (https://theethereum.wiki/w/index.php/ERC20_Token_Standard); allows clients to buy and transfer tokens, and to designate other clients to transfer on their behalf.

Pol_ERCV20_withdraw

\[ \forall e:ERC20. \forall cl:Client. \]
\[ \left[ e.balance(cl) < e.balance(cl)_{pre} \right] \rightarrow \]
\[ \left[ \circ ( \exists cl': Client. \exists m: Nat. \left[ cl.Calls(e.transfer(cl',m)) \right] \right) \rightarrow \]
\[ \circ ( \exists cl'': Client. Authorized(c, cl'') \land c''.Calls(e.transferFrom(c, cl', m)) ) ] \]

This says: A client’s balance decreases only if that client, or somebody authorised by that client, made a payment.
Example3: ERC20 - Authorized
Example 3: ERC20 - Authorized

\[ \text{Authorized}(c, c') \iff \exists m : \text{Nat.} \circ (c.\text{Calls}(e.\text{approve}(c', m))) \]
Example3: ERC20 - Authorized

\[ \text{Authorized}(c, c') \equiv \exists m: \text{Nat. } \circ (c.\text{Calls}(e.\text{approve}(c', m))) \]

*This says*: A client cl’ is authorised by another client cl, iff at some time in the past the latter informed the tokenholder that it authorised the former.
Example3: ERC20 - classical spec
Example 3: ERC20 - classical spec

e:ERC20 ∧ e.balance(cl) > m ∧ e.balance(cl’) = m’ ∧ cl ≠ cl’
   { e.transfer(cl’, m) ∧ Caller = cl }  

Authorized(e, cl, cl’’)
   { e.transferFrom(cl’, m) ∧ Caller = cl’’ }  

e.balance(cl) = e.balance(cl)_{pre} - m ∧ e.balance(cl’)_{pre} = m’ + m

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)

Authorized(e, cl, cl’’)
Example 3: ERC20 - classical vs holistic

\[
e: ERC20 \land e.\text{balance}(cl) > m \land e.\text{balance}(cl) = m' \land cl \neq cl' \\
\{ e.\text{transfer}(cl',m) \land \text{Caller} = cl \}
\]

\[
e.\text{balance}(cl) = e.\text{balance}(cl)_{\text{pre}} - m \land e.\text{balance}(cl)_{\text{pre}} = m' + m
\]

\[
e: ERC20 \land e.\text{balance}(cl) > m \land e.\text{balance}(cl') = m' \land cl \neq cl' \\
\land \text{Authorized}(e, cl, cl'') \\
\{ e.\text{transferFrom}(cl',m) \land \text{Caller} = cl'' \}
\]

\[
e.\text{balance}(cl) = e.\text{balance}(cl)_{\text{pre}} - m \land e.\text{balance}(cl')_{\text{pre}} = m' + m
\]

\[
e: ERC20 \land e.\text{balance}(cl) > m \land e.\text{balance}(cl') = m' \\
\{ e.\text{allow}(cl') \land \text{Caller} = cl \}
\]

Authorized(e, cl, cl'')
Example 3: ERC20 - classical vs holistic

\[ e:ERC20 \land e.balance(cl) > m \land e.balance(cl') = m' \land cl \neq cl' \]
\[
\{ e.transfer(cl',m) \land \text{Caller}=cl \}
\]
\[ e.balance(cl) = e.balance(cl)_{\text{pre}} - m \land e.balance(cl')_{\text{pre}} = m' + m \]

\[ e:ERC20 \land e.balance(cl) > m \land e.balance(cl') = m' \land cl \neq cl' \land \text{Authorized}(e, cl, cl'') \]
\[
\{ e.transferFrom(cl',m) \land \text{Caller}=cl'' \}
\]
\[ e.balance(cl) = e.balance(cl)_{\text{pre}} - m \land e.balance(cl')_{\text{pre}} = m' + m \]

\[ e:ERC20 \land e.balance(cl) > m \land e.balance(cl') = m' \land \text{allow}(cl') \land \text{Caller}=cl \}
\]
\[ \text{Authorized}(e, cl, cl'') \]

The above does not determine whether there are other means to transfer tokens, or to authorise clients. For this we would need to inspect the classic specs of all the functions, or add holistic aspects.
Example 4: DOM attenuation

Access to any Node gives access to **complete** tree
Example 4: 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.height-th parent and nothing else.
Example 4: DOM attenuation - 2
Example 4: DOM attenuation - 2
Example 4: DOM attenuation - 2

\[ \text{Pol}_W \overset{\equiv}{=} \]
\[ \forall S : \text{Set. } \forall \text{nd:Node.} [ \]
\[ [ \text{Access}(s, \text{nd}) \rightarrow s : \text{Node} \lor s : \text{Wrapper} \land \text{Distance}(s.\text{node}, \text{nd}) > s.\text{height} ] \]
\[ \rightarrow \]
\[ \neg ( ( \bullet \text{Changes}(\text{nd.p}))@S ) ] \]

where
\[ \text{Distance}(\text{nd}, \text{nd}') = k \quad \text{iff} \quad \exists j . [ \text{nd.parent}^k = \text{nd}'.\text{parent}^j ] \]
Example 4: DOM attenuation - 3

unknown1

Diagram showing a tree structure with nodes connected by arrows.
Example 4: DOM attenuation - 3
Example 4: DOM attenuation - 3

\[ \text{Pol}_W \triangleq \forall S : \text{Set.} \forall \text{nd:Node.} [ \begin{array}{c} \forall \text{Access}(s, \text{nd}) \rightarrow s : \text{Node} \vee s : \text{Wrapper} \land \text{Distance}(s.\text{node}, \text{nd}) > s.\text{height} \end{array} ] \rightarrow \neg ( ( \circ \text{Changes}(\text{nd}.p)) @ S ) ] \]
Example 4: DOM attenuation - 3

\[ \text{Pol}_W \equiv \forall S \text{: Set. } \forall \text{nd: Node.} [ [ \text{Access}(s, \text{nd}) \rightarrow s: \text{Node} \lor s: \text{Wrapper} \land \text{Distance}(s.\text{node}, \text{nd}) > s.\text{height} ] \rightarrow \neg ( ( \bullet \text{Changes}(\text{nd}.p)) @ S ) ] \]

*This means:*  
A set of objects where any object which can directly access nd is either a Node, or a Wrapper with height smaller than its distance to nd, is insufficient to modify nd.p
Example 4: DOM attenuation - use

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);
...
function mm(unknown) {
    n1 := Node(...); n2 := Node(n1, ...); n3 := Node(n2, ...); n4 := Node(n3, ...);
    n2.p := "robust"; n3.p := "volatile";
    w := Wrapper(n4, 1);
    unknown.untrusted(w);
    ...
}
Example 4: DOM attenuation - use

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

With Pol_W we can show that despite the call to unknown object, at this point:

n2.p := "robust"
### Summary of our Proposal

A ::= e>e | e=e | P(e1,..en) | ...  
| A → A | A ∧ A | ∃x. A | ...  
| Access(x,y) |  permission  
| Changes(e) |  authority  
| •A | ◦A |  time  
| A @ S |  space  
| x.Calls(y,m,z1,..zn) |  call  

Initial(σ) \[ M, \sigma \models A \]

Arising(M) \[ M \models A \]
Functional vs Robust

- services offered by objects/data structure to clients,
- what will happen, under correct use
- sufficient conditions

- preserved properties of the objects/data structure
- what will not happen, under arbitrary use
- necessary conditions
Functional vs Robust

- *services offered* by objects/data structure to *clients*,

- what *will* happen, under *correct use*

- *sufficient conditions*

\[ M \models A \{ \text{code} \} A' \]

- *preserved properties* of the objects/data structure

- what *will not* happen, under *arbitrary use*

- *necessary conditions*
Functional vs Robust

- services offered by objects/data structure to clients,
- what will happen, under correct use
- sufficient conditions

\[ M \models A \{ \text{code} \} A' \]

- preserved properties of the objects/data structure
- what will not happen, under arbitrary use
- necessary conditions

\[ M \models A \]
Functional vs Robust

- services offered by objects/data structure to clients,
- what will happen, under correct use
- sufficient conditions

M ⊨ A \{ code \} A’

- preserved properties of the objects/data structure
- what will not happen, under arbitrary use
- necessary conditions

M ⊨ A
Functional vs Robust

- services offered by objects/data structure to clients,

- what will happen, under correct use

- sufficient conditions

\[ M \models A \{ \text{code} \} A' \]

- preserved properties of the objects/data structure

- what will not happen, under arbitrary use

- necessary conditions

\[ M \models A \]
\[ M \models \Box A \rightarrow A' \]
Functional vs Robust

- *services offered* by objects/data structure to *clients*,

- what *will* happen, under *correct use*

- *sufficient conditions*

\[ M \models A \{ \text{code} \} A' \]

- *preserved properties* of the objects/data structure

- what *will not* happen, under *arbitrary use*

- *necessary conditions*

\[ M \models A \]
\[ M \models \bullet A \rightarrow A' \]
\[ M \models A' \rightarrow \neg (\bullet A) \]
Classical Specification vs Holistic Specification

- fine-grained
- per function

Which one is more accurate?
Classical.

Which one is more expressive?
For a “closed” ADT (no functions can be added, all functions have classical specs, and ghost state has known representation), the holistic specs can be proven.

When do we need holistic specs?
* When the holistic aspect more important (eg cannot lose money unless I authorised).
* When we do not have “closed ADTs.
* When we want to reason in an open world (eg DOM attenuation)
Classical Specification vs Holistic Specification

- fine-grained
- per function

Which one is more accurate? Classical.

Which one is more expressive?
For a “closed” ADT (no functions can be added, all functions have classical specs, and ghost state has known representation), the holistic specs can be proven.

When do we need holistic specs?
* When the holistic aspect more important (eg cannot lose money unless I authorised).
* When we do not have “closed ADTs.
* When we want to reason in an open world (eg DOM attenuation)