

External Calls

Sophia

September 10, 2025

1 Questions we discussed during the Feb/March and then April meetings in London

1. Things we can do next

- (a) better notation
- (b) logic rules for deducing "inside"
- (c) an operator, say \oplus , that combines two holistic specs. That is, assuming that $M_a \models S_a$, and $M_b \models S_b$, we should obtain $M_a \circ M_b \models S_a \oplus S_b$.
- (d) complete Rule-1, and Rule-2
- (e) Things we can do in the future
- (f) Is there use for an inside predicate which specifies the module it refers to?
- (g) Make "inside" "object"-specific, rather than "module"-specific as is now
- (h) Is our logic complete? And do we need all of the logic – compare with ProVerif!

2. How are Necessity Logic and Incorrectness Logic related?

We discussed this, and have a crisp statement. Sophia has written it up in ??.

3. Can we find a different running example?

???

4. Do we need/want to change the specification language? In particular, use ideas from adversarial logic?

At least we will change the notation, cf. sect ???. Also some new concepts possibly, eg *Path(...)*. **STOP PRESS:** : Some new ideas, with which we would need only binary operators for next-only-if and only-if (do not think only-through could be binary); cf. sect ??.

5. How can we reason about methods which allow external calls? How to extend our Logic, so that we can make the argument from sect 2.3.1, from the OOPSLA paper this document, and in particular

- Can we link modules, and how much of spec can we inherit from small to large? ANSWER: yes, we can, and should work on this now!
 - ???
6. Do we want "deep" or "shallow" meaning of "next observable" state in the semantics?
We (provisionally) settled on "shallow". **STOP PRESS:** : We had thought that "shallow" cannot express the DAO vulnerability, but now Sophia think it does, c.f. Sect??.
 7. Title for next paper: "Local reasoning about non-local properties" SD thinks it is a great title, but fears it is not true.
 8. When we consider whether an assertion is satisfied in a state, do we a) **From_Current** only consider the objects which are transitively accessible from the currently top frame, or b) **From_All** all the objects which are transitively accessible from the all the frames? Sophia prefers a), but Julian developed the `Blackadder` example, which, he convinced us all on Friday, would create problems for **From_Current** . **STOP PRESS:** : But now Sophia thinks it does not!.
 9. Sophia's report about the meeting with Jules,
 - (a) He asked about tool support for NL, and whether NL is modular, and how NL treats the universal quantifiers
 - (b) He pointed us to Unrealizability Logic, from POPL 2023
 - (c) Discussed relation with Incorrectness Logic
 - (d) Asked about relation with temporal logic, and suggested (very humbly) that we use more succinct an "mathematical" notation
 - (e) Sophia thinks that we can adopt "adversarial" notation and so facilitate the explanation of the "external steps—" semantics.

2 Comparison with Hoare Logic and Incorrectness Logic

Hoare/Incorrectness Logic are concerned with the study of the effect of some given code. They answer the question whether some specific code (`cmd`) might lead have a certain effect (going from A_1 to A_2). Thus they under/under approx the post condition (A_2) given a precondition (here A_1) and some code (here `cmd`).

In contrast, NL is concerned with security properties; in security, the code being executed might come from any, untrusted, third party, and therefore is unknown. Therefore, NL is not concerned with a *particular* code; instead, it is concerned with the conditions under which some effect (going from A_1 to A_2) , which take place. NL can be used to prove that certain effect will never take place.

Hoare Logic $\{ A_1 \} \text{ cmd } \{ A_2 \}$	$\forall \sigma, \sigma'. [\sigma \models A_1 \wedge \sigma, \text{cmd} \rightsquigarrow^* \sigma' \longrightarrow \sigma' \models A_2]$ A_2 over-approximates the outcome of executing cmd in A_1
Incorrectness Logic $[A_1] \text{ cmd } [A_2]$	$\forall \sigma'. [\sigma' \models A_2 \longrightarrow \exists \sigma. (\sigma \models A_1 \wedge \sigma, \text{cmd} \rightsquigarrow^* \sigma')]$ A_2 under-approximates the outcome of executing cmd in A_1
Necessity Logic $\{ A_1 \Diamond A_2 \} \subseteq \{ A_3 \}$	$\forall \sigma, \sigma', \text{cmd}. [\sigma \models A_1 \wedge \sigma, \text{cmd} \rightsquigarrow^* \sigma' \wedge \sigma' \models A_2 \longrightarrow \sigma \models A_3]$ A_3 over-approximates preconditions that lead from A_1 to A_2

In Hoare Logic as well as in Incorrectness Logic, A_1 is a *sufficient* condition for the *particular* command cmd to reach A_2 . But in Necessity Logic, A_3 is a *necessary* precondition for *any* command starting at A_1 to reach A_2 .

In particular, if we have proven that a module satisfies $\{ A_1 \Diamond A_2 \} \subseteq \{ false \}$, then we know that a transition from A_1 to A_2 will never happen. Note that we could also have Incorrectness-Necessity Logic, where we underapproximate the necessary condition, defined below. Now if we can prove that $\{ A_1 \Diamond A_2 \} \supseteq \{ A_3 \}$ and A_3 is not *false*, then we would have proven that a transition from A_1 to A_2 can happen.¹

I-Necessity Logic (future work ?) $\{ A_1 \Diamond A_2 \} \supseteq \{ A_3 \}$	$\forall \sigma, [\sigma \models A_3 \wedge \sigma \models A_1 \implies \exists \text{cmd}, \sigma' (\sigma, \text{cmd} \rightsquigarrow^* \sigma' \wedge \sigma' \models A_2)]$ A_3 under-approximates preconditions that lead from A_1 to A_2
--	--

3 Notation

3.1 Specifications

Proposed new notation for specifications

¹SOPHIA: not clear this is so; would we need the logic to be complete? TO-THINK.

For *from A1 to A2 onlyIf A3*, is: $\{ A1 \Diamond A2 \} \subseteq \{ A3 \}$.
 For *from A1 next A2 onlyIf A3*, it is: $\{ A1 \bigcirc A2 \} \subseteq \{ A3 \}$.
 For *from A1 to A2 onlyThrough A3*, is: $\{ A1 \Diamond A2 \} \frown \{ A3 \}$.

Proposal Should we perhaps only write necessity specs which will never happen, eg rather than $\{ A1 \Diamond A2 \} \subseteq \{ A3 \}$ have something like $\text{Never}((A1 \wedge \neg A3) \Diamond A2)$. In terms of temporal logic, this would have the meaning: $\Box(\neg((A1 \wedge \neg A3) \wedge \Diamond A2))$. Or, we could go one step further, and specify in the terms of $\Box((A1 \wedge \neg A3) \Diamond \neg A2)$. This would turn our specs into binary operators.

We could have something like

$$(*) \quad \{ A \} \Diamond \{ A' \}$$

which would be like a shorthand for the temporal $\Box(A \rightarrow \Diamond A')$, which means

$$(**) \quad \forall \sigma, \sigma', \text{cmd}. [\sigma \models A \wedge \sigma, \text{cmd} \rightsquigarrow^* \sigma' \longrightarrow \sigma' \models A'].$$

Then, our Bank spec could have the form

$$\{ a : \text{Account} \wedge \text{Insd}(a.\text{passwd}) \wedge a.\text{balance} = b \} \Diamond \{ a.\text{balance} \geq b \}$$

Discussion Having only binary operators would be good, but would open two problems a) how to describe "only-through", b) the "narrative" would be different. We would not have "necessary preconditions" any more. Our spec would be more like 2-state invariants.

Specification Implication

Definition 3.1. A specification S' is *stronger* than a specification S :

$$S' \prec S \triangleq \forall M. [M \models S' \longrightarrow M \models S]$$

2

3.2 Assertions

Proposed new notation for access: $\text{Acc}(y, x)$, and for external: $\text{Extrn}(p)$. Therefore, we will have nothing like "words" or "keywords" in the assertions, or the specifications/

Based on the above, we define "outside", as something that is accessible from something external:

Definition 3.2. We define the predicates $\text{Outsd}(_)$, and $\text{Insd}(_)$ as follows:

$$M, \sigma \models \text{Outsd}(x) \triangleq M, \sigma \models \exists y. [\text{Acc}(y, x) \wedge \text{Extrn}(y)]$$

$$M, \sigma \models \text{Insd}(x) \triangleq \neg (M, \sigma \models \text{Outsd}(x))$$

²We should explore some \prec relations, in the future. And relate with similar in temporal logic. But perhaps all the \prec are already covered in the inference rules that Julian had created? Check whether Julian's rules are complete?

Implication Implication in assertions must be \longrightarrow . But what would it be in the meta-language, eg in Def. ??.

4 Code Examples

Here we will write examples of the codes. We denote though `untrust.unkn(...)` the call to an untrusted (external) object.

4.1 Something

These method s have no "story"; they serve as warnings about things we should not take for grated.

```
1 class Account
2
3     void warning( ) {
4         p := new Password
5         p' := new Password
6         a := new Accoount.
7         a.setPassword(null,p);
8         untrust.unkn(a)
9         this.password := p /* and in variant\_1: */ this.password := p'
10        return p
11    }
12
13    void dare_you(){
14        this.status := frozen // to transfer we need the account to be unfrozen
15        untrust.unkn(this.password)
16        this.password:=new Password
17        this.frozen := false
18    } // SD: hmhhh, essentially we de-activated the account! We need 2.3.1!!!
19
20 ...
```

Deactivate and leak old

This method writes a new Passowrd into the account, and thus essentially disables it. And it returns the old password.

```
1 class Account
2
3     Password deacrctivate_and_leak_old( ) {
4         p := this.password
5         this.password := new Password
6         return p
7     }
8
9     ...
```

Julian will write how this function can be used in the client to xxxx.

4.2 Transfer with external

This method does half the job of transfer, then calls "outside", and then does the rest. Was meant to demo proof of preservation of currency. Proof of pre- and post in the presence of unknown.

```
1 class Account
2
3     void transfer_2 (Password p, Account to, int amt) {
4         if (p==this.password){
5             this.balance -= 2
6             to.balance += 2
7             untrust.unkn();
8             this.balance -= amt -2
9             to.balance += amt-2
10
11         }
12
13     ...
```

4.3 Bank may leak unused Passwords

Bank keeps all unused passwords in a list. Sophia will expand it once "deactive and leak old" is done.

```
1
2 class Bank
3
4     Password makePassword( ) {
5         p := new Password
6         this.listUnusedPwds.enter(p)
7         return p;
8     }
9
10    Password getMeAPassword(){
11        p := this.listUnusedPwds.popAndTop();
12        return p;
13    }
14
15    ...
```

4.4 Safe External Call

Here we overwrite the password before making the external calls

```
1 class MrBean
2     method transfer(from, to, pwd)
3         from.transfer(to, pwd)
4         p = new Password()
5         from.setP(pwd, p)
6         log.logger(from)
7         from.setP(p, pwd)
```

4.5 BlackAdder's Box

Below is a variant of the example of 2.3.1 where the Bank Account has a box field that may contain the password. If it contains the password, then it will allow anyone to change the balance.

This example has "ambient authority", but the authority is not explicit. INTERESTING!

```
1 module BankAccount
2   class BankAccount
3     field password
4     field balance
5     field box
6     method transfer(pwd, to)
7       if pwd == password || box.open() == password
8         this.balance -= 10
9         to.balance += 10
10    method setBox(b)
11      this.box = b
12    method setPassword(pwd, newP)
13      if this.password == pwd
14        this.password = newP
15
16 module Ext
17   class Box
18     field thing
19     method open()
20       return thing
21     method setThing(t)
22       this.thing = t
23
24 module Client
25   class BlackAdder
26     method doSomething()
27       a = new Account()
28       p = new Password()
29       b = new Box()
30       b.setThing(p)
31       a.setPassword(null, p)
32       a.setBox(b)
33       u = new Unknown()
34       u.stuff(a)
```

On Friday, we had thought that RULE-2 from Sect. ?? would be unsound if applied on line 34 (the call `u.stuff(a)`). Namely, we can see that $\text{BankAccount} \models S_{\text{pwd_prot_bal}}$. Also, in line 34, the account is no longer protected, as its box contains the password. On Friday we thought that RULE-2 would erroneously guarantee that the balance would not decrease.

Here we overwrite the password before making the external calls. We can prove it with this week's work. **STOP PRESS:** But SD now argues that Rule-2 would make no such guarantee, Namely, in line 34, we are passing `a` as an argument, and `a` has

access to its box (which is external), and which has access to the password. Therefore, in we consider only the objects that are (transitively) accessible from the frame that gets created in line 34, then $Insd(a.pwd)$ would not hold. And therefore $S_{pwd_prot_bal}$ would not be applicable.

4.6 DAO

STOP PRESS: Sophia maintains that the DAO breaks the invariant even if we consider "shallow semantics" Here the DAO problem paraphrased:

```

1 private class Bank
2     field money: int
3
4     method pay(u: Unknown, amt: int)
5     {
6         money := amt
7         ...
8     }
9
10 class DAOAccount
11     field balance: int
12     field myBank: Bank
13
14     method payOut(u: Unknown) {
15         if balance > 0 then
16             myBank.pay(u, this.balance);
17             u.notify("made the payment")
18         }
19 }

```

On line 17 we are making an external call. At this point, $S_{curr_consistent}$ from Sect. ?? mandates that we have to establish that $myBank.currncy = myBank.money$. However, $myBank.currncy = myBank.money$ does not hold. Note that for this argument we make no special use of NL, and it makes no difference whether we have shallow or deep semantics.

5 Specification Examples

Here we will write examples of the specs

5.1 Flavours

There is an open question as to whether the semantics of from-to... is "deep" or "shallow". By "deep" we mean the semantics where the "to"-part can see inside the future call stack. For example, if external method "m1" called internal method "m2", and if "m2" called external method "m3", then, for the semantic of from-to, do we consider pairs of states where σ is in "m1" and σ' is in "m3"? The "deep" definition would say "yes", and the shallow definition would say "no". The choice does make a difference.

5.2 Passwords protect the balance

$$S_{\text{pwd_prot_bal}} \triangleq \forall a. \forall b. \\ \{a : \text{Account} \wedge a.\text{balance} = b \diamond a.\text{balance} < b\} \subseteq \{\text{Outsd}(a.\text{pwd})\}$$

5.3 No Password get leaked

$$S_{\text{no_leak_val}} \triangleq \forall a. \forall p. \\ \{a : \text{Account} \wedge a.\text{pwd} = p \diamond \text{Outsd}(p)\} \subseteq \{\text{Outsd}(p)\}$$

$$S_{\text{no_leak_fld}} \triangleq \forall a. \forall p. \\ \{a : \text{Account} \diamond \text{Outsd}(a.\text{pwd})\} \subseteq \{\text{Outsd}(a.\text{pwd})\}$$

$$S_{\text{no_leak_strong}} \triangleq \forall p. \\ \{p : \text{Password} \diamond \text{Outsd}(p)\} \subseteq \{\text{Outsd}(p)\}$$

$$S_{\text{no_leak_wrong}} \triangleq \forall p. \\ \{\text{true} \diamond p : \text{Password} \wedge \text{Outsd}(p)\} \subseteq \{\text{Outsd}(p)\}$$

Here the following holds:

$$S_{\text{no_leak_wrong}} \prec S_{\text{no_leak_strong}} \prec S_{\text{no_leak_val}} \\ S_{\text{no_leak_strong}} \prec S_{\text{no_leak_fld}}$$

Note also that $S_{\text{no_leak_wrong}}$ is too strong, because it precludes the creation of any external Password. Notice the difference between $S_{\text{no_leak_fld}}$ and $S_{\text{no_leak_val}}$:

- $S_{\text{no_leak_fld}}$ says that if a password to an account is not externally known, then the field will never be externally known. This does not preclude the example in §??, as it only says that at any moment, the present value will not be externally known. This is useful in the case of protecting the account balance, as it is necessary to know the current password to withdraw money from the account, not previous passwords.
- $S_{\text{no_leak_val}}$ says that if the value pointed at by the field is not externally known, then it will never be externally known. This is useful to say that certain fields are safe, and do not leak the values stored there. (julian: I'm not entirely sure that this spec is good enough though)

5.4 Bank currency constant

We define as currcy the currency of a bank, ie the sum of the balances of all accounts held by that bank.

$$S_{\text{curr_const}} \triangleq \forall b, n. \\ \{b : \text{Bank} \wedge b.\text{currcy} = n \diamond b.\text{currcy} \neq n\} \subseteq \{\text{false}\} \\ S_{\text{curr_infl}} \triangleq \forall b. \\ \{b : \text{Bank} \wedge b.\text{currcy} = n \diamond b.\text{currcy} > n\} \curvearrowright \{\text{Calls}(-, b.\text{print}(-))\}$$

5.5 DAO Bank currency consistent

As in Sect ??, we define as `currcy` the currency of a bank, ie the sum of the balances of all accounts held by that bank. And we require that the currency is the same as the money:

$$S_{\text{curr_consistent}} \triangleq \forall b, n. \{b : \text{Bank} \diamond b.\text{currcy} \neq b.\text{money}\} \subseteq \{false\}$$

The above is an invariant, ie it requires that $b.\text{currcy} = b.\text{money}$ always holds (since \diamond means 0 or more execution steps).

6 Reflections

Here we will say which function satisfy which specs, under which other specs.

7 Extensions to the Inference System of Necessity Logic

We discussed a lot of variations of inference rules. But in the end, Sophia thinks that we need no more than a way to argue that a sequence of "known" code and calls to unknown code satisfies a Hoare triple. And I now think we do not need to "go beyond" Hoare triple notation. :-)

4

7.1 Rule-1

$$\frac{\begin{array}{l} M, S \vdash \{A_1\} \text{ stms}_1 \{A_2\} \\ \mathcal{R}ev(A_2, x, \bar{z}) = A_3 \end{array} \quad \begin{array}{l} x \text{ is external} \\ \mathcal{R}et(A_3, S)_M = A_4 \end{array}}{M, S \vdash \{A_1\} \text{ stms}_1; x.m(\bar{z}) \{A_4\}}$$

In the above, the term $\mathcal{R}ev(A, \bar{z})$ returns an assertion A' which is essentially A , with the difference that all elements from \bar{z} are considered as $Outsd(\cdot)$. Moreover, $\mathcal{R}et(A, S)_M$, contains the parts of A that are preserved through S .⁶

³The following was written here, but Sophia it is not true: Note that with the deep semantics, the method `transfer_2` does not satisfy the spec $S_{\text{curr_const}}$, but with the shallow semantics, it does satisfy the spec $S_{\text{curr_const}}$. The proof will require the use of the spec itself (as in modular verification). And similar arguments about satisfaction of, and proof of adherence to, $S_{\text{curr_infl}}$.

⁴This is what was written here, but Sophia no longer agrees with it.

Note that here, the code about which we are reasoning (ie creating Hoare triples) does not need to be coming from the "safe" module. Instead, we may be reasoning about code which is a client of the "safe" module. That is, we have three "views": the code being checked, i.e., $\text{stms}_1; \text{stms}_2$ below, the "unknown" code (here the calls in stms_2), and the "safe" module, M , which satisfies the specification S . it is possible that in stms_1 we have calls to function from M , and we could reason about them using their pre- and post-conditions. ⁵

⁶needs better explanation.

7.2 Definitions

Definition 7.1. We define the function $\mathcal{R}ev(-, -)$ below:

$$\begin{aligned}
\mathcal{R}ev(A, \epsilon) &\triangleq A \\
\mathcal{R}ev(A, (z, \bar{z})) &\triangleq \mathcal{R}ev(\mathcal{R}ev(A, z), \bar{z}) \\
\mathcal{R}ev(true, z) &\triangleq true \wedge Outsds(z) \\
\mathcal{R}ev(false, z) &\triangleq false \wedge Outsds(z) \\
\mathcal{R}ev(Insds(u), z) &\triangleq (u \neq z \wedge Insds(u) \vee u = z) \wedge Outsds(z) \\
\mathcal{R}ev(Outsds(u), z) &\triangleq Outsds(u) \wedge Outsds(z) \\
\mathcal{R}ev(p = p', z) &\triangleq p = p' \wedge Outsds(z) && \text{where } p, p' \text{ are paths} \\
\mathcal{R}ev(A \wedge A', z) &\triangleq \mathcal{R}ev(A, z) \wedge \mathcal{R}ev(A', z) \\
\mathcal{R}ev(A \vee A', z) &\triangleq \mathcal{R}ev(A, z) \vee \mathcal{R}ev(A', z) \\
\mathcal{R}ev(\neg A, z) &\triangleq \mathcal{R}ev(A', z) && \text{where } A' \equiv \neg A \\
\mathcal{R}ev(\forall u. A, z) &\triangleq \forall u. \mathcal{R}ev(A, z) \\
\mathcal{R}ev(\exists u. A, z) &\triangleq \exists u. \mathcal{R}ev(A, z)^7 \\
\mathcal{R}ev(Q(u), z) &\triangleq \mathcal{R}ev(Qbody[x/u], z) && \text{where } Q \text{ is inductively defined}
\end{aligned}$$

Lemma 1. If $\mathcal{R}ev(A, z) = A'$, then $A \rightarrow A'$, and $A' \rightarrow Outsds(z)$.⁸

We will now define the preservation function:

Definition 7.2. We define $\mathcal{R}et(A, S)_M$ by cases over S and its relation to A below:

$$\begin{aligned}
\mathcal{R}et(A, \{A_1 \diamond A_2\} \subseteq \{A_3\})_M &\triangleq \begin{cases} \neg A_2 & \text{if } M \vdash A \rightarrow A_1 \wedge \neg A_3, \\ true & \text{otherwise.} \end{cases} \\
\mathcal{R}et(A, \{A_1 \diamond A_2\} \subseteq \{A_3\}; S)_M &\triangleq \mathcal{R}et(A, \{A_1 \diamond A_2\} \subseteq \{A_3\})_M \wedge \mathcal{R}et(A, S)_M \\
\mathcal{R}et(A, \{A_1 \diamond A_2\} \curvearrowright \{A_3\}; S)_M &\triangleq \mathcal{R}et(A, S)_M \\
\mathcal{R}et(A, \{A_1 \bigcirc A_2\} \subseteq \{A_3\}; S)_M &\triangleq \mathcal{R}et(A, S)_M
\end{aligned}$$

7.3 Rule-2

On Friday we thought that the rule below was not sound (because of Blackadder); **STOP PRESS:** but now SD thinks it is sound, cf discussion under Secr ??.

$$\frac{\begin{array}{c} M, S \vdash \{A_1\} stms_1 \{A_2\} \\ \mathcal{R}estrTo(A_2, x, \bar{z}) = A_3 \end{array} \quad \begin{array}{c} x \text{ is external} \\ \mathcal{R}et(A_3, S)_M = A_4 \end{array}}{M, S \vdash \{A_1\} stmts_1; x.m(\bar{z}) \{A_4\}}$$

This rule is very similar to that from ??, with the only difference that we use $\mathcal{R}estrTo(A_2, x, \bar{z})$, rather than $\mathcal{R}ev(A_2, x, \bar{z})$. $\mathcal{R}estrTo(A_2, \bar{x}, \bar{z})$ is meant to restrict visibility to only the objects that are transitively accessible from \bar{x}, \bar{z} , and then also turns the external \bar{z} s into outsiders. This assumes that we went with **From_Current**. **STOP PRESS:** I am not that sure how to define $\mathcal{R}estrTo(,)$.

⁸here we also need a M

7.4 Can we combine the holistic specs of two modules?

$M1 \models S_a$ does *not* imply that $M1 \circ M2 \models S_a$. Then, Julian tried something like $M1 \models S_a$ and $M2 \models S_a$, implies that $M1 \circ M2 \models S_a$. This approach is promising, but we still have some problems because *Outsd()*'s meaning depends on the module, and the fact that *Outsd()* can appear in both positive and negative positions.

During our meeting, we thought that such an inference would not ever be sound, because of a counterexample proposed by SD. But **STOP PRESS:**, now SD no longer thinks that this counterexample would be valid.

The counterexample had the structure that for some modules and assertions, we would have

$$\begin{aligned} Mx &\models \{A_1 \diamond A_2\} \subseteq \{false\} \\ My &\models \{A_1 \diamond A_2\} \subseteq \{false\} \\ Mx \circ My &\models \{A_1 \diamond A_2\} \subseteq \{A_3\}, \quad \text{and } A_3 \neq false. \end{aligned}$$

And the example I had given was something like "an object of class Z will perform a certain action (here $A_1 \diamond A_2$) only if it receives a special message from an object from module Mx, as well as a special message from module My". I thought that the specifications above were correctly representing that state of affairs. But I was wrong! It should be something like

$$Mx \models \text{onlyIf } A_1 A_2 Z \text{ has received both the X and Y messages}$$

8 What we can / cannot prove

Here we discuss which examples we can and which we cannot prove now

8.1 Adding external calls to the "safe module's" code

The method from below can be checked with *Rule*₂. **STOP PRESS:** But notice that the spec is too strong! as it assumes that no external entity has access to the password. In that case, nobody can call the method `transfer_3`

```

1  class Account
2
3      fld logger: Logger // external class, untrusted
4
5      void transfer_3a (Password p, Account toAcc, int amt)
6      POST:  Insd(this.passwd) → [ p==this.password) → ...
7                                     ^
8                                     ∀ a:Account.( a≠fromAcc,toAcc → ... ) ]
9      {
10         if (p==this.password)
11             this.balance -= amt
12             toAcc.balance += amt
13             logger.log(this)
14     }
15     ...

```

STOP PRESS: Should we consider "relative" inside/outside predicates, e.g. $Outsd(x, y)$ holds if we only consider objects transitively accessible from y ; that is $Outsd(x, y)$ holds if none of the external objects transitively accessible from y had access to x ?

Here with the new spec

```

1 class Account
2
3     field passwd : Password
4     field logger: Logger // external class, untrusted
5
6     void transfer_3b (Password p, Account toAcc, int amt)
7     POST: [  $\forall a:Account. Outsd(a.passwd, logger)$  ]  $\rightarrow$ 
8           [  $p == this.password$  ]  $\rightarrow \dots$ 
9            $\wedge$ 
10          [  $\forall a:Account. (a \neq fromAcc, toAcc \rightarrow \dots)$  ]
11    { ... as in transfer_3a ... }
12    ...

```

8.2 Adding external calls in "our" code – Me Bean

We now consider functions which belong to a class that is not part of M_{BA} , which use M_{BA} as well as some untrusted third party code. Here we have the module MrBean (ours), which relies on M_{BA} in order to pass its account to a third party while preserving some guarantees.

Consider method `transfer_4`, which is similar to `transfer_3`. **STOP PRESS:** I changed the spec. ⁹

```

1 module MrBean
2
3     class MrAtkinson
4
5         field logger: Logger // external class, untrusted
6
7         void transfer_4 (Account fromAcc, Password p, Account toAcc, int amt)
8         POST: [  $\forall a:Account. Outsd(a.passwd, logger)$  ]  $\rightarrow$ 
9               [  $p == fromAcc.password$  ]  $\rightarrow \dots$ 
10               $\wedge$ 
11              [  $\forall a:Account. (a \neq tfromAcc, toAcc \rightarrow \dots)$  ]
12
13    {
14        fromAcc.transfer(p, toAcc, amt);
15        logger.log(fromAcc)
16    }
17    ...

```

⁹Here some earlier thoughts on the matter in the new spec: I suppose that we were assuming that we have something like $\neg Outsd(a.passwd)$? Or that `a.passwd` is not outside the module MrBean or BankAccount? Perhaps $Outsd(_)$ should mention what it is outside of? Perhaps the module MrBean would have as invariant that all "its" Accounts keeps their Passwords "inside"?

In `transfer_4` from above, we are passing an object from the "safe" module (here `fromAcc`) to the untrusted object. What if we were to pass not only a "safe" object, but also an object from "our" module? Consider the function `transfer_5` from below.

```

1 module MrBean
2
3   class MrAtkinson
4
5     field logger: Logger // external class, untrusted
6
7     // CHALLENGE
8     void transfer_5 (Account fromAcc, Password p, Account toAcc, int amt)
9     POST: [  $\forall a:\text{Account}. \text{Outsd}(a.\text{passwd}, \text{logger}) \rightarrow$ 
10            [  $p == \text{fromAcc}.\text{password} \rightarrow \dots$ 
11               $\wedge$ 
12               $\forall a:\text{Account}. a \neq \text{this}, \text{toAcc} \rightarrow \dots$  ] ]
13
14     {
15       fromAcc.transfer(p, toAcc, amt);
16       logger.log(fromAcc, this)
17     }
18   ...

```

In `transfer_5` above, we are passing `this` to the logger. The function `transfer_5` is only satisfied if `MrAtkinson` does not leak the password, and does not make unauthorized payments. So, it seems as if proving `transfer_5` requires a holistic spec for `MrAtkinson` too.

8.3 What can be proven and how

We can prove `transfer_3a` using *Rule₁*, but as noted above, its spec is far too strong.

How could we prove the spec of `transfer_3a`, and `transfer_4a` and `transfer_4b`. There are the following solutions

STOP PRESS: The below needs revisiting:

1. Include the `Logger`'s code (or its spec) into the proof basis of `Bank`, and make a holistic spec for both of them together. This is a non-solution, because then the logger would not be "external", and "untrusted".
2. Include the `Logger`'s code (or its spec) into the proof basis of `MrBean`. This is also a non-solution, because then the logger would not be "external", and "untrusted".
3. Include `MrBean`'s code into the proof basis of `Bank`, and make a holistic spec for both of them together.
 - (a) We just prove holistically the "whole" new module. We then only need
 - (b) We were hoping that we would be able to "inherit" some of the holistic spec of the `Bank`. As we know, the meaning of *Outsd*(*_*) is delicate as it depends on modules, and appears in both positive and negative positions. But it might be possible.

4. Introduce the concept of path-accessibility, which might be $Path(o, o', A(o''))$, and would mean that any path from o to o' must go through an object o'' which satisfies property $A(o'')$

5. Adopt $Rule_2$

, **Wrong Conjecture** Sophia thought that if all the calls in the body of a function in a client module are either calls to untrusted, or to the safe module (ie no calls to the module at hand), then we should be able to verify it, without holistic spec of the module itself. But `transfer_5` proves her wrong.

Still Open How to prove `transfer_5` in a modular way?