Visible States – Design Decision Required

we can encode it!
18 September 2018

SOPHIA, zzz

For our holistic specifications to be widely applicable, we need to express in them that some intermediate states are not "visible" (not interesting) from the point of view of the particular specification. We were aware of this issue in the past, and we have been pushing it under the carper, but perhaps we should not.

When James visited in July, we discussed this problem again. In this brief document, on Monday 17.09, I outlined what the problem is, and proposed some alternative solutions. Now I think there is a fifth solution, i.e. encode it.

What do you think?

ACM Reference Format:
Sophia. 2025. Visible States – Design Decision Required we can encode it! 18 September 2018. 1, 1 (June 2025), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 THE PROBLEM – ILLUSTRATED THROUGH THE BANK EXAMPLE

1.1 The Bank

Remember the system for electronic money proposed in [?]. In Fig. 1 we outline one possible implementation, where the `Account` objects keep a reference to their `Bank`, the `Bank`object keeps a `ledger`, which is a list of pairs of accounts and balances. Fields declared `private` can only be read or written by the object itself. [1]

Now consider the objects from the diagram in Figure 2. The boxes represent objects (at a certain address and of a certain class), and the arrows represent fields pointing to other objects. For example, at address `1` we have an object of class `Bank`, at `4` and `5` we have objects of class `Node`, *e.t.c.*, and the `myBank` field from object `2` points to `1`, *e.t.c.* The grey objects, at `10`, `11`, `20` and `21` are objects of unknown provenance, but we know that `10` has a field pointing to the `Bank` at `1`, and a field pointing to `11`, and similarly for objects `20` and `21`.

1.2 The Problem

Now, we want to be able to give a formal meaning to things proposed in [?]. eg that

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 don't have.

When we are executing lines 13 and 14 from above we are temporarily modifying the currency without having access to the `Bank`. So, the states when executing the code from lines 13 and 14 should be considered "invisible", and ignored when we describe `Pol_2`. Similarly, when calling

[1]Note that the fields of class `Node` are not private; this is so because `Node` objects do not "escape" the module `Bank`/`Account` and need not be as "robust" as `Bank` objects or `Account` objects.

Author's address: Sophiazzz.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2025 Association for Computing Machinery.
XXXX-XXXX/2025/6-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

```
class Bank {
   private field ledger; // a Node

   Bank( ){  ledger = null; }

   fun makeAccount(amt){  ledger = new Node(Account(this), amt, ledger); }

   fun payFromTo(source,dest,amt){
      src=ledger.find(source);
      dst=ledger.find(dest);
      if (src!=null) && (dst!=null) && src.has(amt)
      then
         src.updateBalanceBy(-amt);  //here we decrease the currency
         dst.updateBalanceBy(amt);   //here we increase the currency
}

class Account {
   private field myBank;

   Account(aBank){ myBank = aBank;  }

   fun sprout( )
   // create Account in same Bank with 0 balance

   fun deposit(source,amnt)
   // if source and receiver are in same Bank,  and source holds enough money,
   //  then transfer amnt from source into receiver
   {  myBank.payFromTo(source,this,amt);  }
}

 class Node{
   field myBalance; // the balance, a number
   field myAccount; // the account
   field next;      // the next node

   fun find(acct){
      if (acct==myAccount) then
         this
      else
         if (next==null) then
            null
         else
            next.find(acct)
      }
   fun has(amt){  myBalance>=amt  }

   fun updateBalanceBy(amt){ myBalance += amt }

   Node(bal,acc,nxt){ myBalance=bal; myAccount=acc; next=nxt }

}
```

Fig. 1. The Bank example – outline

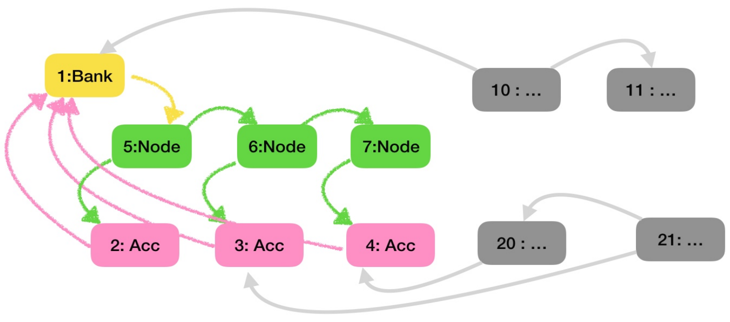

Fig. 2. Diagrammatic representation of some objects from `Bank`, `Account` *etc.*

function `updateBalance` we do not necessarily have (directly) access to the corresponding account, so you could say that the function `updateBalance` invalidates `Pol_4`. Therefore, we see that a specification should be able express which states it considers "invisible".
What we did so far: As of July, we ignored this issue, and were expressing these policies as below:

DEFINITION 1. *We define the set of objects* `o` *internal to a bank or an account* `z`*, as follows:*

$$\texttt{Internal(z)} \triangleq \begin{cases} \{\, \texttt{o} \mid \texttt{o} = \texttt{z} \ \lor\ \texttt{o}:\texttt{Account} \land \texttt{o.myBank} = \texttt{z} \\ \qquad \lor\ \exists k.\ \texttt{z.ledger.next}^k = \texttt{o} \ \}, & \textit{if } \texttt{z:Bank} \\ \{\, \texttt{o} \mid \texttt{z.myBank} : \texttt{Bank} \land \texttt{o} \in \texttt{Internal(z)} \ \}, & \textit{if } \texttt{z:Account} \\ \{\, \}, & \textit{otherwise} \end{cases}$$

Looking at the diagram from Fig. 2, the objects $\{1, 2, 3, 4, 5, 6, 7\}$ are internal to 1, and also internal to 4. The set of objects internal to 6 is empty. We can now look at the policies, and formalize as follows:

DEFINITION 2. *Policy* `Pol_2` *promises that if* `b` *was a* `Bank`*, and its* `currency` *were to change, and the set of objects involved in that change was* `S`*, then there is at least one object in* `S` *which has access to* `b`*, and which is not internal to* `b`*. And similar for* `Pol_4`*. Formally:*

$$\texttt{Pol\_2} \triangleq \forall \texttt{b}.\forall \texttt{S}.[\ \texttt{b}:\texttt{Bank} \land \texttt{this} \neq \texttt{b} \land (\Diamond \mathit{Changes}(\texttt{b.currency}))\ \text{in}\ \texttt{S} \longrightarrow \\ \exists \texttt{o}.[\ \texttt{o} \in \texttt{S} \land \mathcal{A}\mathit{ccess}(\texttt{o},\texttt{b}) \land \texttt{o} \notin \texttt{Internal(b)}\ ]\ ]$$

$$\texttt{Pol\_4} \triangleq \forall \texttt{a}.\forall \texttt{S}.[\ \texttt{a}:\texttt{Account} \land \texttt{this} \neq \texttt{a} \land (\Diamond \mathit{Changes}(\texttt{a.balance}))\ \text{in}\ \texttt{S} \longrightarrow \\ \exists \texttt{o}.[\ \texttt{o} \in \texttt{S} \land \mathcal{A}\mathit{ccess}(\texttt{o},\texttt{a}) \land \texttt{o} \notin \texttt{Internal(a)}\ ]\ ]$$

An example illustrating that the code from Fig. 1 does not satisfy `Pol_2` is when we called, say, `2.deposit(3)`. This will eventually lead to the calls `5.updateBalance(100)` and `6.updateBalance(-100)`. The set of objects involved in the first call is { 5}, which is an object that is internal to 1. The first call reduces the currency by 100, and thus we have violated `Pol_2`. However, the second call increases it by 100, and thus net effect on `currency` is none. Therefore, we want to express that for this policy all the states between the beginning of `6.updateBalance(-100)` and its end are "invisible".
[2]

2 POSSIBLE SOLUTIONS

I see the following four alternatives

[2]Note that the requirements `this` ≠ `b` and `this` ≠ `a` are annoying, but at the moment I do not know how to avoid them. Consider this an orthogonal problem.

2.1 Solution_1: Postpone

Just ignore the question for the time being

Advantage: we do not solve too many problems together. Several examples from the contracts literature can be expressed withoutthese concerns (as many simple contracts do not have auxiliary objects).

Disadvantage: there is simple code from the object capabilities literature which does not adhere to policies.

2.2 Solution_2: Visible States from the Invariant Literature.

Adapt the solutions that were developed in the invariants literature – a survey can be found at [?].

Advantage: we would be adapting known technology. Disadvantage: it is a bit inflexible, and it requires some different-style tools than what we have so far.

2.3 Solution_3: Maximal Flexibility

We add one more assertion-constructor, which says which are the states which it considers visible.

The assertion $\mathcal{V}isible(A, B)$ says that A holds in a semantics which only considers states whose receiver is in the set described by B; all other states are "skipped". That is, we have a new operational semantics, which considers some part of execution as atomic. [3].

Then, policy `Pol_2` would be

$$
\begin{array}{l}
\texttt{Pol\_2} \triangleq \forall \texttt{b}.\forall \texttt{S}.[\ \texttt{b}:\texttt{Bank} \ \longrightarrow \\
\qquad\qquad\qquad \mathcal{V}isible\ (\\
\qquad\qquad\qquad\qquad (\Diamond\, Changes(\texttt{b.currency}))\ \text{in}\ \texttt{S} \quad \longrightarrow \\
\qquad\qquad\qquad\qquad\qquad \exists \texttt{o}.\ [\ \texttt{o} \in \texttt{S}\ \wedge\ \mathcal{A}ccess(\texttt{o},\texttt{b})\ \wedge\ \texttt{o} \notin \texttt{Internal(b)}\ ], \\
\qquad\qquad\qquad\qquad \texttt{External(b)}\ )\ ]
\end{array}
$$

and where the external objects to `o` is the complement of the internals.

$\texttt{External(z,o)} \triangleq \texttt{o} \notin \texttt{Internal(z)}$

Therefore, the assertion from above says, that for all banks, if we consider as visible only the states whose receiver is not internal to a bank `b`, and if and the `currency` of `b` changes the set of objects involved in that change was `S`, then there is at least one object in `S` which has access to `b`, and which is not internal to `b`.

Advantage: flexible.

Disadvantage: it requires the satisfaction judgment to be a bit more complex, and have the form $\texttt{M}, B, \sigma \models A$. But doable.

We would define things as follows. First, define that a state σ is visible for B:

- $\texttt{M}, \sigma \Vdash B$ iff $\sigma(\texttt{this}) \in \lfloor B \rfloor_{\sigma,\texttt{M}}$

Then, define the operational semantics, which records only the states that are visible for B:

- $\texttt{M}, \sigma \leadsto_B \sigma'$
  iff
  $\texttt{M}, \sigma \Vdash B$, and $\texttt{M}, \sigma \Vdash B$, and
  $\exists n \geq 0. \exists \sigma_1, ... \sigma_n.$
  $[\ \texttt{M}, \sigma \leadsto \sigma_1\ \wedge\ \texttt{M}, \sigma_1 \leadsto \sigma_2 ... \wedge \texttt{M}, \sigma_n \leadsto \sigma'\ \wedge\ \texttt{M}, \sigma_1 \nVdash B\ \wedge\ \texttt{M}, \sigma_2 \nVdash B ... \wedge\ \texttt{M}, \sigma_n \nVdash B\ ].$

Finally, we define satisfaction by cases as follows:

- $\texttt{M}, B, \sigma \models e$ iff $\texttt{M}, \sigma \Vdash B\ \wedge\ \lfloor e \rfloor_\sigma = true$

[3]Perhaps we need a stratified system, where the set description B is not as expressive as A

- $\texttt{M}, B, \sigma \models Changes(e)$ iff $\exists \sigma'.[\ \sigma \leadsto_B^* \sigma' \ \wedge\ \lfloor e \rfloor_\sigma \neq \lfloor e \rfloor_{\sigma'}\ ]$
- $\texttt{M}, B, \sigma \models \mathcal{V}isible(A, B')$ iff $\texttt{M}, B \cap B', \sigma \models A$
- ... etc ...

2.4 Solution_4: Reduced Flexibility

Instead of making a general-purpose assertion-constructor, we expand the ones we have now, to say that a change, or a future effect is observed when considering a restricted set of visible states. So We will have:
$Changes(e, B)$ says that a change in e is observed with visible states as described in B, and
$\lozenge\ (A, B)$ says that at some future point which is observable through visible states as described in B the assertion A will hold, and
$\nabla(A, B)$ says that at some past point which was observable through visible states as described in B the assertion A held.

With these assertions, we describe `Pol_2` as follows

$$\texttt{Pol\_2} \triangleq \forall \texttt{b}.\forall \texttt{S}.[\ \texttt{b:Bank} \ \wedge\ (\lozenge Changes(\texttt{b.currency}, \texttt{External(b)}))\ \text{in}\ \texttt{S} \ \longrightarrow \ \exists \texttt{o}.[\ \texttt{o} \in \texttt{S} \ \wedge\ \mathcal{A}ccess(\texttt{o}, \texttt{b}) \ \wedge\ \texttt{o} \notin \texttt{Internal(b)}\ ]\ ]$$

Advantage: easier to make the semantics.
Disadvantage: ad hoc, and a bit repetitive. If we combine many things observed under the same visible states, then it will become ugly, eg something like $Changes(\texttt{e}, \texttt{B}) \ \to Changes(\texttt{e'}, \texttt{B})$ is less natural than $\mathcal{V}isible(Changes(\texttt{e}) \ \to Changes(\texttt{e'}), \texttt{B})$

We would define it as follows

- $\texttt{M}, \sigma \models e$ iff $\lfloor e \rfloor_\sigma = true$ // ie as in July version
- $\texttt{M}, \sigma \models Changes(e, B)$ iff $\exists \sigma'.[\ \sigma \leadsto_B^* \sigma' \ \wedge\ \lfloor e \rfloor_\sigma \neq \lfloor e \rfloor_{\sigma'}\ ]$
- $\texttt{M}, \sigma \models \lozenge\ (A, B)$ iff $\exists \sigma'.[\ \sigma \leadsto_B^* \sigma' \ \wedge\ \texttt{M}, \sigma \models A\ ]$
- ... etc ...

2.5 Solution_5: Encode it

I think we do not need new assertion-constructors, but can encode as follows:

$$\begin{array}{l}
\texttt{Pol\_2} \triangleq \forall \texttt{b}.\forall \texttt{S}.\forall \texttt{v}: \\
\quad [\ \texttt{b:Bank} \ \wedge\ \texttt{b.balance} = \texttt{v} \ \wedge\ \texttt{this} \notin \texttt{Internal(b)} \\
\qquad\qquad \wedge\ \lozenge(\texttt{b.currency} \neq \texttt{v} \ \wedge\ \texttt{this} \notin \texttt{Internal(b)}\ )\ \text{in}\ \texttt{S} \\
\quad\ \ \longrightarrow \\
\quad\ \ \exists \texttt{o}.[\ \texttt{o} \in \texttt{S} \ \wedge\ \mathcal{A}ccess(\texttt{o}, \texttt{b}) \ \wedge\ \texttt{o} \notin \texttt{Internal(b)}\ ]\ ]
\end{array}$$

The above says that if the current state is external to the bank `b` (ie the current receiver is not internal to `b`), and the current balance is `v`, and if in some future state external to `b` the balance of `b` is no longer `v`, then the set of objects used in effecting this change contains at least one object `o` external to `b` which has direct access to `b`.

Advantage: we do not need any new concepts. Also, we no longer use the concept of $Changes(\_)$. A bit verbose?
Disadvantage: is not as expressive as the previous two solutions, as it allows me to talk about any two consecutive external states, while the previous two solutions allowed me tot capture the nearest extrernal successor state. But I think it does the job for the examples we have considered so far.

3 SYNTAX

What should the assertion-constructors look like? Should they be prefix when unary and infix when binary? Should we use keywords rather than symbols, eg $\textbf{future}\,A$ rather than $\Diamond A$? And $A\,\textbf{observing}\,B$ rather than $\mathcal{V}isible(A, B)$? I think that syntax is secondary, but perhaps it affects which of the alternatives we (the design team) consider as easier to understand.

BIBLIOGRAPHY

REFERENCES

Drossopoulou, S., Francalanza, A., Müller, P., and Summers, A. J. A unified framework for verification techniques for object invariants. In *ECOOP* (2008), LNCS, Springer.

Miller, M. S., Morningstar, C., and Frantz, B. Capability-based Financial Instruments: From Object to Capabilities. In *Financial Cryptography* (2000), Springer.