Asynchronous Communication II

INF4140 - Models of concurrency

Asynchronous Communication, lecture 11

Høsten 2013

11.11.2013

- semantics: histories and trace sets
- specification: invariants over histories
  - global invariants
  - local invariants
  - the connection between local and global histories
- example: Coin machine
  - the main program
  - formulating local invariants

- Analysis of **send**/**await** statements
- Verifying local history invariants
- example: Coin Machine
  - proving loop invariants
  - the local invariant and a global invariant
- example: Mini bank

Agent/network systems (Repetition)

We consider general agent/network systems:

- Concurrent agents:
  - with self identity
  - no variables shared between agents
  - communication by message passing
- Network:
  - no channels
  - no FIFO guarantee
  - no guarantee of successful transmission

Sequential language with statements for sending and receiving:

- **send statement: send** $B : m(e)$
  means that the current agent sends message m to agent B where e is an (optional) list of actual parameters.
- **fixed receive statement: await** $B : m(w)$
  wait for a message m from a specific agent B, and receive parameters in the variable list w. We say that the message is then *consumed*.
- **open receive statement: await** $X\,?\,m(w)$
  wait for a message m from any agent X and receive parameters in w. (consuming the message). The variable X will be set to the agent that sent the message.
- We may use a **choice operator** [] to select between alternative statement lists, starting with receive statements.

Here m is a message name, B and e expressions, X and w variables.

Local reasoning by Hoare Logic (a.k.a Program Logic)

We adapt Hoare logic to reasoning about local histories in an agent A:

- Introducing a local (pseudo) variable h, initialized to empty ε
  - h represents the local history of A
- For a **send**/**await** statement, we then define the effect on h.
  - extending the h with the corresponding event
- *Local reasoning*: we do not know the global invariant
  - For **await**: do not know parameter values
  - For *open receive*: do not know the sender
- Use non-deterministic assignment

$$x := \textbf{some}$$

where variable x may be given any (type correct) value

Local invariant reasoning by Hoare Logic

- each send statement in A, say **send** $B : m$, is treated as

$$h := (h; A \uparrow B : m)$$

- each fixed receive statement in A, say **await** $B : m(w)$, where w is a list of variables, is treated as

$$w := \textbf{ some } ; h := (h; B \downarrow A : m(w))$$

  here, the usage of $w := \textbf{ some }$ expresses that A may receive any values for the receive parameters

- each open receive statement in A, from an arbitrary agent X, say **await** $X\,?\,m(w)$, is treated as

$$X := \textbf{ some } ; \textbf{await } X : m(w)$$

  where the usage of $X := \textbf{ some }$ expresses that A may receive the message from any agent

Rule for non-deterministic assignments

Non-deterministic assignments have the following rule:

$$\{\forall x \,.\, Q\}\; x := \textbf{some}\; \{Q\}$$

We may then derive rules for the introduced **send**/**await** statements.

Derived rule for send:

$$\{Q_{h \leftarrow h; A \uparrow B:m}\}\ \textbf{send}\ B : m\ \{Q\}$$

Derived rule for receive from specific agent:

$$\{\forall w\ .\ Q_{h \leftarrow h; B \downarrow A:m(w)}\}\ \textbf{await}\ B : m(w)\ \{Q\}$$

Derived rule for receive from unknown agent:

$$\{\forall w, X\ .\ Q_{h \leftarrow h; X \downarrow A:m(w)}\}\ \textbf{await}\ X\,?\,m(w)\ \{Q\}$$

As before, A is the current agent/object, and h the local history. We assume that neither B nor X occur in w, and that w is a list of distinct variables.

Remark: If there are no parameters to a fixed receive statement, say **await** $B : m$, we may simplify the Hoare Rule (message name m):

$$\{Q_{h \leftarrow h; (B \downarrow A:m)}\}\ \textbf{await}\ B : m\ \{Q\}$$

Note: No shared variables are used. Therefore, no interference, and Hoare reasoning can be done as usual in the sequential setting!

The Hoare rule for non-deterministic choice ([]) is

$$\frac{\{P_1\}\, S_1\, \{Q\} \qquad \{P_2\}\, S_2\, \{Q\}}{\{P_1 \wedge P_2\}\, (S_1 [] S_2)\, \{Q\}}$$

We may also reason backwards over **if** statements:

$$\frac{\{P_1\}\, S_1\, \{Q\} \qquad \{P_2\}\, S_2\, \{Q\}}{\{\textbf{if}\ \ b\ \ \textbf{then}\ \ P_1\ \ \textbf{else}\ \ P_2\}\, \textbf{if}\ \ b\ \ \textbf{then}\ \ S_1\ \ \textbf{else}\ \ S_2\ \ \textbf{fi}\ \{Q\}}$$

where the precondition $\textbf{if}\ \ b\ \ \textbf{then}\ \ P_1\ \ \textbf{else}\ \ P_2$ is an abbreviation for $(b \Rightarrow P_1) \wedge (\neg b \Rightarrow P_2)$

Remark: The assignment axiom is valid: $\{Q_{x \leftarrow e}\}\, x := e\, \{Q\}$

Consider an agent C which changes "5 krone" coins and "1 krone" coins into "10 krone" coins. It receives *five* and *one* messages and sends out *ten* messages as soon as possible, in the sense that the number of messages sent out should equal the total amount of kroner received divided by 10. We imagine here a fixed user agent

U, both producing the *five* and *one* messages and consuming the *ten* messages. The code of the agent C is given below, using b (*balance*) as a local variable initialized to 0.

```
loop
    while b<10 do
        (await U:five; b:=b+5)
      []
        (await U:one; b:=b+1)
    od
    send U:ten; b:=b-10
end
```

Here, the choice operator, [], selects the first enabled branch, (and makes a non-deterministic choice if both branches are enabled).

Invariants may refer to the *local history* h, which is the sequence of events visible to C that have occurred so far. The events visible to C are:

$U \downarrow C : \mathit{five}$	$--$	C consumes the message "five"
$U \downarrow C : \mathit{one}$	$--$	C consumes the message "one"
$C \uparrow U : \mathit{ten}$	$--$	C sends the message "ten"

Loop invariant for the outer loop:

$$\mathit{OUTER}: \; \mathit{sum}(h/\downarrow) = \mathit{sum}(h/\uparrow) + b \;\wedge\; 0 \leq b < 5$$

where *sum* (the sum of values in the messages) is defined as follows:

$$\begin{array}{lcl} \mathit{sum}(\varepsilon) & = & 0 \\ \mathit{sum}(h; (\ldots : \mathit{five})) & = & \mathit{sum}(h) + 5 \\ \mathit{sum}(h; (\ldots : \mathit{one})) & = & \mathit{sum}(h) + 1 \\ \mathit{sum}(h; (\ldots : \mathit{ten})) & = & \mathit{sum}(h) + 10 \end{array}$$

Loop invariant for the inner loop:

$$\mathit{INNER}: \; \mathit{sum}(h/\downarrow) = \mathit{sum}(h/\uparrow) + b \;\wedge\; 0 \leq b < 15$$

Prove that *INNER* is preserved by the body of the inner loop.

Backward construction gives:

while `b<10` **do** $\{b < 10 \land INNER\}$

$\{(INNER_{b \leftarrow (b+5)})_{h \leftarrow h;U \downarrow C:five} \land (INNER_{b \leftarrow (b+1)})_{h \leftarrow h;U \downarrow C:one}\}$

(**await** `U:five;` $\{INNER_{b \leftarrow (b+5)}\}$

`b:=b+5`)

[]

(**await** `U:one;` $\{INNER_{b \leftarrow (b+1)}\}$

`b:=b+1`)

$\{INNER\}$

od

Must prove the implication:

$$b < 10 \land INNER \Rightarrow (INNER_{b \leftarrow (b+5)})_{h \leftarrow h;U \downarrow C:five} \land (INNER_{b \leftarrow (b+1)})_{h \leftarrow h;U \downarrow C:one}$$

(details left as an exercise)

Note: From the precondition *INNER* for the loop, we have $INNER \land b \geq 10$ as the postcondition to the inner loop.

Prove that $OUTER$ is preserved by the outer loop body.
Backward construction gives:

$\{OUTER\}$
while `true` **do** $\{OUTER\}$
$\{INNER\}$
while `b<10` **do** ...**od** $\{INNER \wedge b \geq 10\}$
$\{(OUTER_{b \leftarrow (b-10)})_{h \leftarrow h; C \uparrow U:ten}\}$
send `U:ten;`
$\{OUTER_{b \leftarrow (b-10)}\}$
`b:=b-10`
$\{OUTER\}$
od

Verification conditions:

- $OUTER \Rightarrow INNER$, and
- $INNER \wedge b \geq 10 \Rightarrow (OUTER_{b \leftarrow (b-10)})_{h \leftarrow h; C \uparrow U:ten}$
- $OUTER$ holds initially since $h = \varepsilon \wedge b = 0 \Rightarrow OUTER$

Local history invariant

For each agent (A):

- Predicate $I_A(h)$ over the local communication history (h)
- Describes the interaction between A and the surrounding agents
- Must be maintained by *all* history extensions in A
- Last week: Local history invariants for the different agents may be composed, giving a global invariant

Verification idea:

- Ensure that $I_A(h)$ holds initially (i.e., with $h = \varepsilon$)
- Ensure that $I_A(h)$ holds after each **send**/**await** statement (Assuming that $I_A(h)$ holds before each such statement)

Local history invariant reasoning by Hoare logic

- may use Hoare logic to prove properties of the code in agent A
- for instance loop invariants
- the conditions may refer to the local state v (a list of variables) and the local history h, e.g., $Q(v, h)$.

The local history invariant $I_A(h)$:

- must hold after each send/receive
- if Hoare reasoning gives the condition $Q(v, h)$ immediately after a send or receive statement, we basically need to ensure:

$$Q(v, h) \Rightarrow I_A(h)$$

- we may assume that the invariant is satisfied immediately before each send/receive point.
- we may also assume that the last event of h is the send/receive event.

Proving the local history invariant

Let $I_A(h)$ be the local invariant of an agent A. The rule and comments on the previous slide can be formulated as the following verification conditions for each **send**/**await** statement in A:

send $B : m$:

$$(\ h = (h'; A{\uparrow}B : m) \wedge I_A(h') \wedge Q(v, h)\ ) \Rightarrow I_A(h)$$

- Q is the condition immediately after the send statement
- assumption $h = (h'; A{\uparrow}B : m)$: the history (after the statement) ends with the send event
- assumption $I_A(h')$: the invariant holds before the send statement

Proving the local history invariant (cont.)

await $B : m(w)$:

$$(\ h = (h'; B{\downarrow}A : m(w)) \wedge I_A(h') \wedge Q(v, h)\ ) \Rightarrow I_A(h)$$

where Q is the condition right after the receive statement.

await $X\,?\,m(w)$:

$$(\ h = (h'; X{\downarrow}A : m(w)) \wedge I_A(h') \wedge Q(v, h)\ ) \Rightarrow I_A(h)$$

where Q is the condition right after the receive statement.

Coin machine example: local history invariant

For the coin machine C, consider the local history invariant $I_C(h)$ from last week:

$$I_C(h) = 0 \leq sum(h/\downarrow) - sum(h/\uparrow) < 15$$

Consider the statement **send** $U : ten$ in C

- Hoare analysis of the outer loop gave the condition $OUTER_{b \leftarrow (b-10)}$ immediately after the statement
- The history ends with the event $C \uparrow U : ten$
- Verification condition:

$$h = h'; (C \uparrow U : ten) \land I_C(h') \land OUTER_{b \leftarrow (b-10)} \Rightarrow I_C(h)$$

Verification condition (details):

$$h = h'; (C\!\uparrow\! U : ten) \wedge I_C(h') \wedge OUTER_{\,b \leftarrow (b-10)} \Rightarrow I_C(h)$$

by definitions I_C and $OUTER$:

$$(h = h'; (C\!\uparrow\! U : ten)) \wedge (0 \leq sum(h'/\!\downarrow) - sum(h'/\!\uparrow) < 15)$$
$$\wedge\ (sum(h/\!\downarrow) = sum(h/\!\uparrow) + b - 10 \wedge\ 0 \leq b - 10 < 5)$$
$$\Rightarrow 0 \leq sum(h/\!\downarrow) - sum(h/\!\uparrow) < 15$$

by $h = h'; (C\!\uparrow\! U : ten)$ and def. of sum:

$$(0 \leq sum(h'/\!\downarrow) - sum(h'/\!\uparrow) < 15)$$
$$\wedge sum(h'/\!\downarrow) = sum(h'/\!\uparrow) + 10 + b - 10 \wedge\ 0 \leq b - 10 < 5)$$
$$\Rightarrow 0 \leq sum(h'/\!\downarrow) - sum(h'/\!\uparrow) - 10 < 15$$

now we have $b = sum(h'/\!\downarrow) - sum(h'/\!\uparrow)$:

$$0 \leq b < 15 \wedge 0 \leq b - 10 < 5 \Rightarrow 0 \leq b - 10 < 15$$

which is trivial since $b - 10 < 5 \Rightarrow b - 10 < 15$

Correctness proofs (bottom-up):

- code
- loop invariants (Hoare analysis)
- local history invariant
- verification of local history invariant based on the Hoare analysis

Note: The []-construct was useful for programming service-oriented systems, and had a simple proof rule.

Client cycle: The client C is making these messages

- put in card, give pin, give amount to withdraw, take cash, take card

Mini Bank cycle: The mini bank M is making these messages

to client: ask for pin, ask for withdrawal, give cash, return card

to central bank: request of withdrawal

Central Bank cycle: The central bank B is making these messages

to mini bank: grant a request for payment, or deny it

There may be many mini banks talking to the same central bank, and there may be many clients using each mini bank (but the mini bank must handle one client at a time).

Mini bank example: Global histories

Consider a client C, mini bank M and central bank B:

Example of successful cycle:

$[\quad C\updownarrow M : card\_in(n), M\updownarrow C : pin, C\updownarrow M : pin(x),$
$M\updownarrow C : amount, C\updownarrow M : amount(y), M\updownarrow B : request(n, x, y), B\updownarrow M : grant,$
$M\updownarrow C : cash(y), M\updownarrow C : card\_out\ ]$

where n is name, x pin code, and y cash amount, provided by clients.

Example of unsuccessful cycle:

$[\quad C\updownarrow M : card\_in(n), M\updownarrow C : pin, C\updownarrow M : pin(x),$
$M\updownarrow C : amount, C\updownarrow M : amount(y), M\updownarrow B : request(n, x, y), B\updownarrow M : deny,$
$M\updownarrow C : card\_out\ ]$

Notation: $A\updownarrow B : m$ denotes the sequence $A\uparrow B : m, A\downarrow B : m$

Mini bank example: Local histories (1)

From the global histories above, we may extract the corresponding local histories:

The successful cycle:

- Client: $[C{\uparrow}M : card\_in(n), M{\downarrow}C : pin, C{\uparrow}M : pin(x), M{\downarrow}C : amount, C{\uparrow}M : amount(y), M{\downarrow}C : cash(y), M{\downarrow}C : card\_out]$
- Mini Bank: $[C{\downarrow}M : card\_in(n), M{\uparrow}C : pin, C{\downarrow}M : pin(x), M{\uparrow}C : amount, C{\downarrow}M : amount(y), M{\uparrow}B : request(n, x, y), B{\downarrow}M : grant, M{\uparrow}C : cash(y), M{\uparrow}C : card\_out]$
- Central Bank: $[M{\downarrow}B : request(n, x, y), B{\uparrow}M : grant]$

The local histories may be used as guidelines when implementing the different agents.

Mini bank example: Local histories (2)

The unsuccessful cycle:

- Client: $[C{\uparrow}M : card\_in(n), M{\downarrow}C : pin, C{\uparrow}M : pin(x),$ $M{\downarrow}C : amount, C{\uparrow}M : amount(y), M{\downarrow}C : card\_out]$
- Mini Bank: $[C{\downarrow}M : card\_in(n), M{\uparrow}C : pin, C{\downarrow}M : pin(x),$ $M{\uparrow}C : amount, C{\downarrow}M : amount(y), M{\uparrow}B : request(n, x, y),$ $B{\downarrow}M : deny, M{\uparrow}C : card\_out]$
- Central Bank: $[M{\downarrow}B : request(n, x, y), B{\uparrow}M : deny]$

Note: many other executions possible, say when clients behaves differently, difficult to describe all at a global level (remember the formula of week 1).

Sketch of simple central bank.

Program variables:

```
pin -- array of pin codes, indexed by client names
bal -- array of account balances, indexed by client names

X : Agent, n:  Client_Name, x:  Pin_Code, y:  Natural
```

Code:

```
loop
   await X?request(n,x,y);
   if pin[n]=x and bal[n]>y
       then bal[n]:=bal[n]-y; send X:grant;
       else send X:deny
    fi
end
```

Note: the mini bank X may vary with each iteration.

Note: no absolute deadlock, but concurrent requests not allowed.

Mini bank example: Central Bank (B)

Consider the (extended) regular expression $Cycle_B$ defined by:

$[\,X{\downarrow}B : request(n,x,y), [\,B{\uparrow}X : grant \mid B{\uparrow}X : deny\,]\ \textbf{some}\ X,n,x,y\,]^*$

- with | for choice, [...]* for repetition
- Defines cycles: *request* answered with either *grant* or *deny*
- notation $[regExp\ \textbf{some}\ X,n,x,y]^*$ means that the values of X, n, x, and y are fixed in each cycle, but may vary from cycle to cycle.

Notation: Given an extended regular expression R.
Let h **is** R denote that h matches the structure described by R.
Example (for events a, b, and c):

- we have $(a; b; a; b)$ **is** $[a,b]^*$
- we have $(a; c; a; b)$ **is** $[a,[b|c]]^*$
- we do *not* have $(a; b; a)$ **is** $[a,b]^*$

Loop invariant of Central Bank (B):

Let $Cycle_B$ denote the regular expression:

$$[\, X{\downarrow}B : request(n,x,y), [\, B{\uparrow}X : grant \mid B{\uparrow}X : deny \,] \textbf{ some } X, n, x, y \,]^*$$

Loop invariant: h **is** $Cycle_B$

Proof of loop invariant (entry condition): Must prove that it is satisfied initially: ε **is** $Cycle_B$, which is trivial.

Proof of loop invariant (invariance):

```
loop  {h is Cycle_B}
   await X?request(n,x,y);
   if pin[n]=x and bal[n]>y
       then bal[n]:=bal[n]-y; send X:grant;
       else send X:deny
    fi
{h is Cycle_B}
end
```

Backward construction of a precondition for the loop body:

while `true` **do** $\{h \textbf{ is } Cycle_B\}$

$\{\forall X, n, x, y \,.\, \textbf{if } pin[n] = x \wedge bal[n] > y$
$\quad \textbf{then } (h; X{\downarrow}B : request(n, x, y); B{\uparrow}X : grant) \textbf{ is } Cycle_B$
$\quad \textbf{else } (h; X{\downarrow}B : request(n, x, y); B{\uparrow}X : deny) \textbf{ is } Cycle_B\}$

await `X?request(n,x,y);`

$\{\textbf{if } pin[n] = x \wedge bal[n] > y \textbf{ then } (h; B{\uparrow}X : grant) \textbf{ is } Cycle_B$
$\quad \textbf{else } (h; B{\uparrow}X : deny) \textbf{ is } Cycle_B\}$

if `pin[n]=x and bal[n]>y` **then**

$\{(h; B{\uparrow}X : grant) \textbf{ is } Cycle_B\}$

`bal[n]:=bal[n]-y;`

$\{(h; B{\uparrow}X : grant) \textbf{ is } Cycle_B\}$

send `X:grant;`

else

$\{(h; B{\uparrow}X : deny) \textbf{ is } Cycle_B\}$

send `X:deny`

fi

$\{h \textbf{ is } Cycle_B\}$

end

Verification condition:

$$h \textbf{ is } Cycle_B \Rightarrow \forall X, n, x, y \,.\, \textbf{if } pin[n] = x \wedge bal[n] > y$$
$$\textbf{then } (h; X{\downarrow}B : request(n,x,y); B{\uparrow}X : grant) \textbf{ is } Cycle_B$$
$$\textbf{else } (h; X{\downarrow}B : request(n,x,y); B{\uparrow}X : deny) \textbf{ is } Cycle_B$$

where $Cycle_B$ is

$$[\, X{\downarrow}B : request(n,x,y), [\, B{\uparrow}X : grant \mid B{\uparrow}X : deny \,] \textbf{ some } X, n, x, y \,]^*$$

The condition follows by the general rule (regExp R and events a and b):

$$h \textbf{ is } R^* \wedge (a; b) \textbf{ is } R \Rightarrow (h; a; b) \textbf{ is } R^*$$

since $(X{\downarrow}B : request(n,x,y); B{\uparrow}X : grant) \textbf{ is } Cycle_B$
and $(X{\downarrow}B : request(n,x,y); B{\uparrow}X : deny) \textbf{ is } Cycle_B$

$Cycle_B$ is

$$[\, X{\downarrow}B : request(n, x, y), [\, B{\uparrow}X : grant \mid B{\uparrow}X : deny \,] \textbf{ some } X, n, x, y \,]^*$$

Define the history invariant for B by:

$$h \leq Cycle_B$$

Let $h \leq R$ denote that h is a prefix of the structure described by R.

- intuition: if $h \leq R$ we may find some extension h' such that $(h; h')$ **is** R
- h **is** $R \Rightarrow h \leq R$ (but not vice versa)
- $(h; a)$ **is** $R \Rightarrow h \leq R$
- Example: $(a; b; a) \;\leq\; [a, b]^*$

$h \leq Cycle_B$

- As before, we need to ensure that the history invariant is implied after each send/receive statement.
- Here it is enough to assume the conditions after each send/receive statement in the verification of the loop invariant

This gives 2 proof conditions:

1. after send grant/deny (i.e. after **fi**)

h **is** $Cycle_B \Rightarrow h \leq Cycle_B$ which is trivial.

2. after await request

if $\ldots$ **then** $(h; B{\uparrow}X : grant)$ **is** $Cycle_B$ **else** $(h; B{\uparrow}X : deny)$ **is** $Cycle_B$

$\Rightarrow h \leq Cycle_B$ which follows from $(h; a)$ **is** $R \Rightarrow h \leq R$.

Note: We have now proved that the implementation of B satisfies the local history invariant, $h \leq Cycle_B$.

Mini bank example: Local invariant of Client (C)

$Cycle_C$:
$[\ C\!\uparrow\!X : card\_in(n)$
$\mid X\!\downarrow\!C : pin, C\!\uparrow\!X : pin(x)$
$\mid X\!\downarrow\!C : amount, C\!\uparrow\!X : amount(y')$
$\mid X\!\downarrow\!C : cash(y)$
$\mid X\!\downarrow\!C : card\_out$ **some** $X, y, y'\ ]^*$

History invariant:

$$h_C \leq Cycle_C$$

Note: The values of C, n and x are fixed from cycle to cycle.

Note: The client is willing to receive cash and cards, and give card, at any time, and will respond to *pin*, and *amount* messages from a mini bank X in a sensible way, without knowing the protocol of the particular mini bank. This is captured by $\mid$ for different choices.

$Cycle_M$:

$[\ C{\downarrow}M : card\_in(n), M{\uparrow}C : pin, C{\downarrow}M : pin(x),$
$\quad M{\uparrow}C : amount, C{\downarrow}M : amount(y),$
$\quad \textbf{if } y \leq 0 \textbf{ then } \varepsilon \textbf{ else}$
$\quad M{\uparrow}B : request(n, x, y), [B{\downarrow}M : deny \mid B{\downarrow}M : grant, M{\uparrow}C : cash(y)\ ] \textbf{ fi}\ ,$
$\quad M{\uparrow}C : card\_out \textbf{ some } C, n, x, y\ ]^*$

History invariant:

$$h_M \leq Cycle_M$$

Note: communication with a fixed central bank. The client may vary with each cycle.

Note: deadlock if a client does not respond properly.

Mini bank example: obtaining a global invariant

Consider the parallel composition of C, B, M. Global invariant:
$legal(H) \wedge H/\alpha_C \leq Cycle_C \wedge H/\alpha_M \leq Cycle_M \wedge H/\alpha_B \leq Cycle_B$

Assuming no other agents, this invariant may *almost* be formulated by:

$$
\begin{aligned}
H \leq [& C \updownarrow M : card\_in(n), M \updownarrow C : pin, C \updownarrow M : pin(x), \\
& M \updownarrow C : amount, C \updownarrow M : amount(y), \\
& \textbf{if } y \leq 0 \textbf{ then } M \updownarrow C : card\_out \\
& \textbf{else } M \updownarrow B : request(n, x, y), [B \updownarrow M : deny, M \updownarrow C : card\_out \\
& \;\; \mid B \updownarrow M : grant, M \uparrow C : cash(y), [M \downarrow C : cash(y) \; ||| \; M \updownarrow C : card\_out]] \textbf{ fi} \\
& \textbf{some } n, x, y \;]^*
\end{aligned}
$$

where ||| gives all possible interleavings. However, we have no guarantee that the cash and the card events are received by C before another cycle starts. Any next client may actually take the cash of C.

For proper clients it works OK, but improper clients may cause the Mini Bank to misbehave. Need to incorporate assumptions on the clients, or make an improved mini bank.

Improved mini bank based on a discussion of the global invariant

The analysis so far has discovered some weaknesses:

- The mini bank does not know when the client has taken his cash, and it may even start a new cycle with another client before the cash of the previous cycle is removed. This may be undesired, and we may introduce a new event, say *cash_taken* from C to M, representing the removal of cash by the client. (This will enable the mini bank to decide to take the cash back within a given amount of time.)
- A similar discussion applies to the removal of the card, and one may introduce a new event, say *card_taken* from C to M, so that the mini bank knows when a card has been removed. (This will enable the mini bank to decide to take the card back within a given amount of time.)
- A client may send improper or unexpected events. These may be lying in the network unless the mini bank receives them, and say, ignores them. For instance an old misplaced amount message may be received in (and interfere with) a later cycle. An improved mini bank could react to such message by terminating the cycle, and in between cycles it could ignore all messages (except card_in).

Summary

Concurrent agent systems, without network restrictions (need not be FIFO, message loss possible).

- Histories used for semantics, specification and reasoning
- correspondence between global and local histories, both ways
- parallel composition from local history invariants
- extension of Hoare logic with send/receive statements
- avoid interference, may reason as in the sequential setting
- Bank example, showing
  - global histories may be used to exemplify the system, from which we obtain local histories, from which we get useful coding help
  - specification of local history invariants
  - verification of local history invariants from Hoare logic + verification conditions (one for each send/receive statement)
  - composition of local history invariants to a global invariant