Asynchronous Communication I

INF4140 - Models of concurrency

Asynchronous Communication, lecture 10

Høsten 2014

7.11.2014

Asynchronous Communication: Semantics, specification and reasoning

Where are we?

- part one: shared variable systems
  - programming
  - synchronization
  - reasoning by invariants and Hoare logic
- part two: communicating systems
  - message passing
  - channels
  - rendezvous

What is the connection?

- What is the semantic understanding of message passing?
- How can we understand concurrency?
- How to understand a system by looking at each component?
- How to specify and reason about asynchronous systems?

Clarifying the semantic questions above, by means of histories:

- describing interaction
- capturing interleaving semantics for concurrent systems
- Focus: asynchronous communication systems without channels

Plan today

- histories from the outside view of components
  - describing overall understanding of a (sub)system
- Histories from the inside view of a component
  - describing local understanding of a single process
- The connection between the inside and outside view
  - the composition rule

What kind of system? Agent network systems

Two kinds of settings for concurrent systems, based on the notion of:

- processe — without self identity, but with named channels. Channels often FIFO.
- object (agent) — with self identity, but without channels, sending messages to named objects through a network. In general, a network gives no FIFO guarantee, nor guarantee of successful transmission.

We use the latter here, since it is a very general setting. The process/channel setting may be obtained by representing each combination of object and message kind as a channel.

in the following we consider agent/network systems!

New syntax statements for sending and receiving:

- *send statement:* $\texttt{send}\ B : m(e)$
  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.
- *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.

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
```

- choice operator [ ][1]
  - selects 1 *enabled* branch
  - non-deterministic choice if both branches are enabled

[1] In the literature, also + as notation can often be found.

- behavior of a concurrent system: may be described as set of executions,
- 1 execution: sequence of atomic interaction events,
- other names for it: trace, history, execution, (interaction) sequence . . . [2]

Interleaving semantics

Concurrency is expressed by the set of all possible interleavings.

- remember also: "sequential consistency" from the WMM part.
- note: for each interaction sequence, all interactions are ordered sequentially, and their "visible" concurrency

[2] message sequence (charts) in UML etc.

- very well known and widely used “format” to descibe “languages” (= sets finite “words” over given a given “alphabet”)
-

Example (Reg-Expr)

- a, b: atomic interactions.
- Assume them to "run" concurrently

$\Rightarrow$ two possible interleavings, described by

$$[[a.b] + [b.a]] \tag{1}$$

Parallel composition of a^* and b^*:

$$(a + b)^* \tag{2}$$

Remark: notation for reg-expr's

Different notations exist. E.g.: some write $a|b$ for the *alternative/non-deterministic* choice between a and b. We use $+$ instead

- to avoid confusion with parallel composition
- be consistent with common use of regexp. for describing concurrent behavior

Safety and liveness & traces

We may let each interaction sequence reflect all interactions in an execution, called the trace, and the set of all possible traces is then called the trace set.

- terminating system: finite traces[3]
- *non-terminating* systems: infinite traces
- trace set semantics in the general case: both finite and infinite traces
- 2 conceptually important classes of properties[4]
  - safety ("nothing wrong will happen")
  - liveness ("something good will happen")

[3] Be aware: typically an *infinite* set of finite traces.

[4] Safety etc. it's not a property, it's a "property/class of properties"

Safety and liveness & histories

- often: concentrate on *finite traces*
- reasons
  - conceptually/theoretically simpler
  - connection to monitoring
  - connection to checking (violations of) safety prop's
- our terminology: history = trace up to a given execution point (thus finite)
- note: In contrast to the book, histories are here finite initial parts of a trace (prefixes)
- sets of histories are

prefix closed

if a history h is in the set, then every prefix (initial part) of h is also in the set.

- sets of histories: can be used capture safety, but not liveness

Simple example: histories and trace set

Consider a system of two agents, A and B, where agent A says "hi-B" repeatedly until B replies "hi-A".

	traces	histories
a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B"	${hi_B}^{\infty} + {hi_B}^{+} hi_A$	${hi_B}^{*} + {hi_B}^{+} hi_A$
a "lazy" B will reply eventually, but there is no limit on how long A may need to wait. Thus, each trace will end with "hi_A" after finitely many "hi_B"'s.		
an "eager" B will reply within a fixed number of "hi_B"'s, for instance before A says "hi_B" three times.		

- a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B" (here comma denotes union).

Simple example: histories and trace set

Consider a system of two agents, A and B, where agent A says "hi-B" repeatedly until B replies "hi-A".

	traces	histories
a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B"	${hi_B}^\infty + {hi_B}^+ hi_A$	${hi_B}^* + {hi_B}^+ hi_A$
a "lazy" B will reply eventually, but there is no limit on how long A may need to wait. Thus, each trace will end with "hi_A" after finitely many "hi_B"'s.		
an "eager" B will reply within a fixed number of "hi_B"'s, for instance before A says "hi_B" three times.		

- a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B" (here comma denotes union).
  Trace set: $\{[hi_B]^\infty\}, \{[hi_B]^+ [hi_A]\}$

Consider a system of two agents, A and B, where agent A says "hi-B" repeatedly until B replies "hi-A".

	traces	histories
a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B"	${hi_B}^{\infty} + {hi_B}^{+} hi_A$	${hi_B}^{*} + {hi_B}^{+} hi_A$
a "lazy" B will reply eventually, but there is no limit on how long A may need to wait. Thus, each trace will end with "hi_A" after finitely many "hi_B"'s.		
an "eager" B will reply within a fixed number of "hi_B"'s, for instance before A says "hi_B" three times.		

- a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B" (here comma denotes union).
  Trace set: $\{[hi_B]^{\infty}\}, \{[hi_B]^{+}\ [hi_A]\}$
  Histories: $\{[hi_B]^{*}\}, \{[hi_B]^{+}\ [hi_A]\}$

Simple example: histories and trace set

Consider a system of two agents, A and B, where agent A says "hi-B" repeatedly until B replies "hi-A".

	traces	histories
a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B"	${hi_B}^{\infty} + {hi_B}^{+} hi_A$	${hi_B}^{*} + {hi_B}^{+} hi_A$
a "lazy" B will reply eventually, but there is no limit on how long A may need to wait. Thus, each trace will end with "hi_A" after finitely many "hi_B"'s.		
an "eager" B will reply within a fixed number of "hi_B"'s, for instance before A says "hi_B" three times.		

- a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B" (here comma denotes union).
  Trace set: $\{[hi_B]^{\infty}\}, \{[hi_B]^{+}\,[hi_A]\}$
  Histories: $\{[hi_B]^{*}\}, \{[hi_B]^{+}\,[hi_A]\}$

Simple example: histories and trace set

Consider a system of two agents, A and B, where agent A says "hi-B" repeatedly until B replies "hi-A".

	traces	histories
a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B"	$hi_B{}^{\infty} + hi_B{}^{+} hi_A$	$hi_B{}^{*} + hi_B{}^{+} hi_A$
a "lazy" B will reply eventually, but there is no limit on how long A may need to wait. Thus, each trace will end with "hi_A" after finitely many "hi_B"'s.		
an "eager" B will reply within a fixed number of "hi_B"'s, for instance before A says "hi_B" three times.		

- a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B" (here comma denotes union).
  Trace set: $\{[hi_B]^{\infty}\}, \{[hi_B]^{+}\ [hi_A]\}$
  Histories: $\{[hi_B]^{*}\}, \{[hi_B]^{+}\ [hi_A]\}$

Simple example: histories and trace set

Consider a system of two agents, A and B, where agent A says "hi-B" repeatedly until B replies "hi-A".

	traces	histories
a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B"	${hi_B}^{\infty} + {hi_B}^{+} hi_A$	${hi_B}^{*} + {hi_B}^{+} hi_A$
a "lazy" B will reply eventually, but there is no limit on how long A may need to wait. Thus, each trace will end with "hi_A" after finitely many "hi_B"'s.		
an "eager" B will reply within a fixed number of "hi_B"'s, for instance before A says "hi_B" three times.		

- a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B" (here comma denotes union).
  Trace set: $\{[hi_B]^{\infty}\}, \{[hi_B]^{+}\ [hi_A]\}$
  Histories: $\{[hi_B]^{*}\}, \{[hi_B]^{+}\ [hi_A]\}$

Simple example: histories and trace set

Consider a system of two agents, A and B, where agent A says "hi-B" repeatedly until B replies "hi-A".

	traces	histories
a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B"	$hi_B{}^{\infty} + hi_B{}^{+} hi_A$	$hi_B{}^{*} + hi_B{}^{+} hi_A$
a "lazy" B will reply eventually, but there is no limit on how long A may need to wait. Thus, each trace will end with "hi_A" after finitely many "hi_B"'s.		
an "eager" B will reply within a fixed number of "hi_B"'s, for instance before A says "hi_B" three times.		

- a "sloppy" B may or may not give a reply, in which case there will be an infinite trace with only "hi-B" (here comma denotes union).
  Trace set: $\{[hi_B]^{\infty}\}, \{[hi_B]^{+}\ [hi_A]\}$
  Histories: $\{[hi_B]^{*}\}, \{[hi_B]^{+}\ [hi_A]\}$

Let use the following conventions

- events x : *Event* is an event,
- set of events: $A : 2^{Event}$
- history h : *Hist*

A set of events is assumed to be fixed.

Definition (Histories)

Histories (over the given set of events) is given inductively over the constructors ϵ (empty history) and $\_;\_$ (appending of an event to the right of the history)

function	type			
ϵ	:	$\to$	*Hist*	the empty history (constructor)
$\_;\_$	: *Hist* $*$ *Event*	$\to$	*Hist*	append right (constructor)
$\mid \_ \mid$	: *Hist*	$\to$	*Nat*	length
$\_/\_$	: *Hist* $*$ *Set*	$\to$	*Hist*	projection by set of events
$\_ \leq \_$	: *Hist* $*$ *Hist*	$\to$	*Bool*	prefix relation
$\_ < \_$	: *Hist* $*$ *Hist*	$\to$	*Bool*	strict prefix relation

Inductive definitions (inductive wrt. ε and $\_;\_$):

$$
\begin{aligned}
&|\,\epsilon\,| && = 0 \\
&|\,(h;x)\,| && = |\,h\,| + 1 \\
&\epsilon/A && = \epsilon \\
&(h;x)/s && = \texttt{if } x \in A \texttt{ then } (h/s);x \texttt{ else } (h/s) \texttt{ fi} \\
&h \leq h' && = (h = h') \vee h < h' \\
&h < \varepsilon && = \mathit{false} \\
&h < (h';x) && = h \leq h'
\end{aligned}
$$

May use invariants to define trace sets:
A (history) invariant I is a predicate over a histories, supposed to hold at all times:

"At any point in an execution h the property $I(h)$ is satisfied"

It defines the following set:

$$\{h \mid I(h)\} \tag{3}$$

- mostly interested in *prefix-closed invariants*!
- a history invariant is historically monotonic:
$$h \leq h' \Rightarrow (I(h') \Rightarrow I(h)) \tag{4}$$
- I history-monotonic $\Rightarrow$ set from equation (3) prefix closed

Remark:
A non-monotonic predicate I may be transformed to a monotonic one I':

$$\begin{aligned} I'(\varepsilon) &= I(\varepsilon) \\ I'(h';x) &= I(h') \wedge I(h';x) \end{aligned}$$

Consider asynchronous communication by messages from one agent to another: Since message passing may take some time, the sending and receiving of a message m are semantically seen as two distinct atomic interaction events of type Event:

- $A{\uparrow}B : m$ denotes that A sends message m to B
- $A{\downarrow}B : m$ denotes that B receives (consumes) message m from A

A global history, H, is a finite sequence of such events, requiring that it is legal, i.e. each reception is preceded by a corresponding send-event.
For instance, the history

$$[(A{\uparrow}B : hi), (A{\uparrow}B : hi), (A{\downarrow}B : hi), (A{\uparrow}B : hi), (B{\uparrow}A : hi)]$$

is legal and expresses that A has sent "hi" three times and that B has received one of these and has replied "hi".

Note: a concrete message may also have parameters, say
$messagename(parameterlist)$
where the number and types of the parameters are statically checked.

Coin Machine Example: Events

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

- note all global sequences/histories "make sense"
- depens on the programming language/communciation model
- sometimes called well-definedness, well-formedness or similar
- $legal : Hist \rightarrow Bool$

Definition (Legal history)

$$
\begin{aligned}
legal(\epsilon) &= true \\
legal(h; (A{\uparrow}B : m)) &= legal(h) \\
legal(h; (A{\downarrow}B : m)) &= legal(h) \wedge \\
&\qquad |(h/\{A{\downarrow}B : m\})| < |(h/\{A{\uparrow}B : m\})|
\end{aligned}
$$

where m is message and h a history.

- should m include parameters, legality ensures that the values received are the same as those sent.

Example (coin machine C user U):

$[(U{\uparrow}C : five), (U{\uparrow}C : five), (U{\downarrow}C : five), (U{\downarrow}C : five), (C{\uparrow}U : ten)]$

Outside view: logging the global history

How to "calculate" the global history at run-time:

- introduce a global variable H,
- initialize: to empty sequence
- for each execution of a send statement in A, update H by

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

  where B is the destination and m is the l message
- for each execution of a receive statement in B, update H by

  $$H := H; (A \downarrow B : m)$$

  where m is the message and A the sender. The message must be of the kind requested by B.

Global invariant: By a predicate I on the global history, we may specify desired system behavior:

> *"at any point in an execution H the property $I(H)$ is satisfied"*

- By logging the history at run-time, as above, we may monitor an executing system. When $I(H)$ is violated we may
  - report it
  - stop the system, or
  - interact with the system (for inst. through fault handling)
- How to prove such properties by analysing the program?
- How can we monitor, or prove correctness properties, component-wise ?

Definition (Local events)

The events visible to an agent A, denoted α_A, are the events local to A, i.e.:

- $A{\uparrow}B : m$: any send-events from A. (output by A)
- $B{\downarrow}A : m$: any reception by A. (input by A)

Definition (Local history)

Given a global history: The local history of A, written h_A, is the subsequence of all events visible to A

- Conjecture: Correspondence between global and local view:

$$h_A = H/\alpha_A$$

  i.e. at any point in an execution the history observed locally in A is the projection to A -events of the history observed globally.
- Each event is visible to one, and only one, agent!

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"

The events visible to U are:

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

How to relate local and global views

From global specification to implementation:
First, set up the goal of a system: by one or more global histories. Then implement it. For each component: use the global histories to obtain a local specification, guiding the implementation work.
"construction from specifications"

From implementation to global specification:
First, make or reuse components.
Use the local knowledge for the desired components to obtain global knowledge.

Working with invariants:
The specifications may be given as invariants over the history.

- Global invariant: in terms of all events in the system
- Local invariant (for each agent): in terms of events visible to the agent

Need composition rules connecting local and global invariants.

Example revisited: Sloppy coin machine

```
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
```

interactions visible to C (i.e. those that may show up in the local history):

$$
\begin{array}{lcl}
U{\downarrow}C : \mathit{five} & -- & C \text{ consumes the message ``five''} \\
U{\downarrow}C : \mathit{one} & -- & C \text{ consumes the message ``one''} \\
C{\uparrow}U : \mathit{ten} & -- & C \text{ sends the message ``ten''}
\end{array}
$$

Loop invariant for the outer loop:

$$sum(h/\downarrow) = sum(h/\uparrow) + b \ \wedge \ 0 \leq b < 5 \tag{5}$$

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

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

Loop invariant for the inner loop:

$$sum(h/\downarrow) = sum(h/\uparrow) + b \ \wedge \ 0 \leq b < 15 \tag{6}$$

From local histories to global history: if we know all the local histories h_{Ai} in a system ($i = 1 \ldots n$), we have

$$legal(H) \wedge_i \; h_{A_i} = H/\alpha_{A_i}$$

i.e. the global history H must be legal and correspond to all the local histories. This may be used to reason about the global history.
Local invariant: a local specification of A_i is given by a predicate on the local history $I_{A_i}(h_{A_i})$ describing a property which holds before all local interaction points.

I may have the form of an implication, expressing the output events from A_i depends on a condition on its input events.
From local invariants to a global invariant:
if each agent satisfies $I_{A_i}(h_{A_i})$, the total system will satisfy:

$$legal(H) \wedge_i \; I_{A_i}(H/\alpha_{A_i})$$

Coin machine example: from local to global invariant

before each send/receive: (see eq. (6))

$$sum(h/\downarrow) = sum(h/\uparrow) + b \;\wedge\; 0 \leq b < 15$$

Local Invariant of C in terms of h alone:

$$I_C(h) = \exists b.\ (sum(h/\downarrow) = sum(h/\uparrow) + b \;\wedge\; 0 \leq b < 15) \qquad (7)$$

$$I_C(h) = 0 \leq sum(h/\downarrow) - sum(h/\uparrow) < 15 \qquad (8)$$

For a global history H ($h = H/\alpha_C$):

$$I_C(H/\alpha_C) = 0 \leq sum(H/\alpha_C/\downarrow) - sum(H/\alpha_C/\uparrow) < 15 \qquad (9)$$

Shorthand notation:
$I_C(H/\alpha_C) = 0 \leq sum(H/\downarrow C) - sum(H/C\uparrow) < 15$

- Local Invariant of a careful user U (with exact change):

$$I_U(h) = 0 \leq sum(h/\uparrow) - sum(h/\downarrow) \leq 10$$
$$I_U(H/\alpha_U) = 0 \leq sum(H/U\uparrow) - sum(H/\downarrow U) \leq 10$$

- Global Invariant of the system U and C:

$$I(H) = legal(H) \wedge I_C(H/\alpha_C) \wedge I_U(H/\alpha_U) \qquad (10)$$

implying:

Overall

$$0 \leq sum(H/U\downarrow C) - sum(H/C\uparrow U) \leq sum(H/U\uparrow C) - sum(H/C\downarrow U) \leq 10$$

since $legal(H)$ gives:
$sum(H/U\downarrow C) \leq sum(H/U\uparrow C)$ and
$sum(H/C\downarrow U) \leq sum(H/C\uparrow U)$.

So, globally, this system will have balance ≤ 10.

Loop invariant for the outer loop:

$$rec(h) = sent(h) + b \ \wedge \ 0 \leq b < 5$$

where *rec* (the total amount received) and *sent* (the total amount sent) are defined as follows:

$$\begin{array}{lcl}
rec(\varepsilon) & = & 0 \\
rec(h; (U \downarrow C : five)) & = & rec(h) + 5 \\
rec(h; (U \downarrow C : one)) & = & rec(h) + 1 \\
rec(h; (C \uparrow U : ten)) & = & rec(h) \\
sent(\varepsilon) & = & 0 \\
sent(h; (U \downarrow C : five)) & = & sent(h) \\
sent(h; (U \downarrow C : one)) & = & sent(h) \\
sent(h; (C \uparrow U : ten)) & = & sent(h) + 10
\end{array}$$

Loop invariant for the inner loop:

$$rec(h) = sent(h) + b \ \wedge \ 0 \leq b < 15$$

The above definition of legality reflects networks where you may not assume that messages sent will be delivered, and where the order of messages sent need not be the same as the order received. Perfect networks may be reflected by a stronger concept of legality (see next slide).

Remark: In "black-box" specifications, we consider observable events only, abstracting away from internal events. Then, legality of sending may be strengthened:

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

If the network delivers messages in a FIFO fashion, one could capture this by strengthening the legality-concept suitably, requiring

$$\mathit{sendevents}(h/\downarrow) \;\leq\; h/\uparrow$$

where the projections $h/\uparrow$ and $h/\downarrow$ denote the subsequence of messages sent and received, respectively, and *sendevents* converts receive events to the corresponding send events.

$$\begin{aligned}
\mathit{sendevents}(\varepsilon) &= \varepsilon \\
\mathit{sendevents}(h; (A{\uparrow}B : m)) &= \mathit{sendevents}(h) \\
\mathit{sendevents}(h; (A{\downarrow}B : m)) &= \mathit{sendevents}(h); (A{\uparrow}B : m)
\end{aligned}$$

Channel-oriented systems can be mimicked by requiring FIFO ordering of communication for each pair of agents:

$$\mathit{sendevents}(h/A\downarrow B) \leq \; h/A{\uparrow}B$$

where $A\downarrow B$ denotes the set of receive-events with A as source and B as destination, and similarly for $A{\uparrow}B$.

References I

[Andrews, 2000] Andrews, G. R. (2000).
Foundations of Multithreaded, Parallel, and Distributed Programming.
Addison-Wesley.