System Specification, Verification and Synthesis (SSVS) – CS 4830/7485, Fall 2019

6: Formal System Modeling: Asynchronous Composition

Stavros Tripakis

Northeastern University
Khoury College of Computer Sciences

ASYNCHRONOUS COMPOSITION

Basic model: interleaving and shared variables

- A bunch of shared (global) variables.
- A bunch of *processes*: each modeled as an extended state machine.
- A process can read a variable, write a variable, test a variable, ...
- Processes interleave: only one process moves at a time.

Basic model: interleaving and shared variables

Example:

```
// a small example spin model
// Peterson's solution to the mutual exclusion problem (1981)

bool turn, flag[2];         // the shared variables, booleans
byte ncrit;                 // nr of procs in critical section

active [2] proctype user()  // two processes
{
    assert(_pid == 0 || _pid == 1);
again:
    flag[_pid] = 1;
    turn = _pid;
    (flag[1 - _pid] == 0 || turn == 1 - _pid);

    ncrit++;
    assert(ncrit == 1);     // critical section
    ncrit--;

    flag[_pid] = 0;
    goto again
}
// analysis:
// $ spin -run peterson.pml
```

Subtleties

Consider this multi-threaded program:

```
Shared vars A, B: bool;
Initially A = B = 0;

Thread 1                    Thread 2
A := 1;                     B := 1;
if (B = 0)                  if (A = 0)
  print("Hello ");             print("World ");
```

What might be printed?

- "Hello "?
- "World "?
- "Hello World "?
- "World Hello "?
- Something else?
- Nothing?

Subtleties

Consider this multi-threaded program:

```
Shared vars A, B: bool;
Initially A = B = 0;

Thread 1                    Thread 2
A := 1;                     B := 1;
if (B = 0)                  if (A = 0)
  print("Hello ");             print("World ");
```

What might be printed?

- "Hello "?
- "World "?
- "Hello World "?
- "World Hello "?
- Something else?
- Nothing?

- Interleaving semantics implicitly assumes **sequential consistency**!
- But there are weaker memory models.
- **Homework**: model and verify this example in Spin.

Other subtleties

- Atomicity: are reads and writes **atomic**?
- What if Thread 1 has a statement like:

  `x := x+1;`

  where `x` is a shared variable.
- Can some other thread update the value of `x` after Thread 1 has read it, but before it has updated it?

Other subtleties

- Atomicity: are reads and writes **atomic**?
- What if Thread 1 has a statement like:

  `x := x+1;`

  where `x` is a shared variable.
- Can some other thread update the value of `x` after Thread 1 has read it, but before it has updated it?

- Careful with what you model!
- Some languages offer `atomic` constructs (e.g., Spin).

Another basic model: rendez-vous

- A bunch of processes: each modeled as an extended state machine.
- Processes mostly interleave: only one process moves at a time.
- Except for some transitions which must **synchronize**.
- Common in *process algebras*, e.g., CSP [Hoare, 1985], CCS [Milner, 1980], etc.
- In Spin this is modeled with channels of length 0.
  - Message cannot be stored in the channel queue (because queue size is 0) $\Rightarrow$ transmitter and receiver must **synchronize**.

  $\Rightarrow$ transmission and reception occurs simultaneously.
- Called **handshake** in [Baier and Katoen, 2008].

Rendez-vous communication: example

CSP notation:

$$a! \big\downarrow \quad || \quad \big\downarrow a? \quad = \quad \big\downarrow \tau$$

CCS notation:

$$a \big\downarrow \quad || \quad \big\downarrow \overline{a} \quad = \quad \big\downarrow \tau$$

τ: **silent** (or **internal**) action.

Another basic model: asynchronous message passing

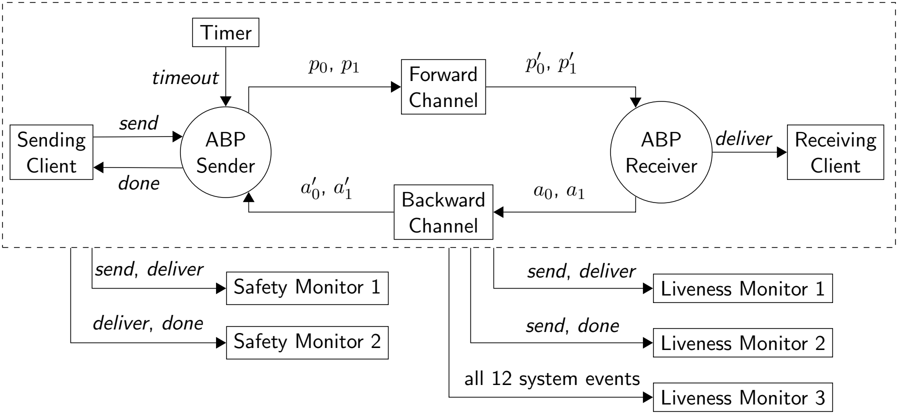
- Sender sends message to a queue.
- Receiver reads message from the queue.
- Many variants, depending on how these questions are resolved:
  - Can multiple senders write to the same queue?
  - Can multiple receivers read from the same queue?
  - Are the queues FIFO? lossy? ...
  - Are the queues of finite length?
  - If queues are finite, what happens when I try to send a message and the queue is already full?
  - What happens if I try to read and the queue is empty?
  - ...
- Some examples of models:
  - *Kahn Process Networks* [Kahn, 1974]: infinite queues, single-writer, single-reader, blocking read $\Rightarrow$ determinism!
  - Petri nets [Murata, 1989]: unordered tokens, multiple-writer, multiple-reader.
  - Spin: shared vars + rendez-vous + channels

EXAMPLE: THE ALTERNATING BIT PROTOCOL

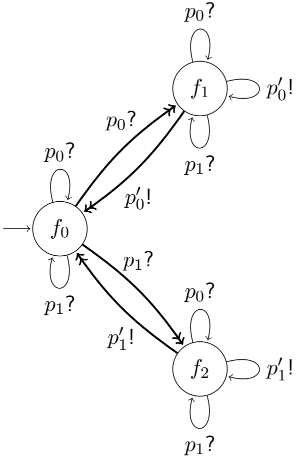
The Alternating Bit Protocol (ABP)

- A simple communication protocol: reliable transmission over unreliable channels.
- Routinely used to illustrate formal modeling and verification techniques [Holzmann, 2003, Lynch, 1996].
- Model presented here taken from [Alur and Tripakis, 2017].
- We will return to this example later.
- **Homework**: For now, you should use it as practice to form the asynchronous parallel composition of all processes in the system.
- **Homework**: How many states does the product transition system for the ABP model have in total? How many of those states are reachable?

ABP System Architecture



The channels



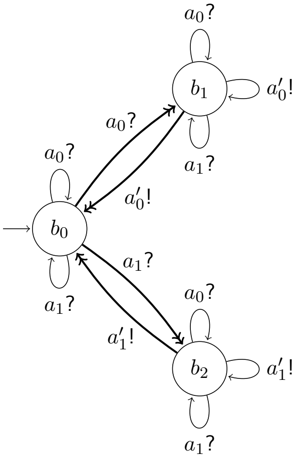


Figure: Environment processes *Forward Channel* (left) and *Backward Channel* (right). Transitions in bold lines and double arrows are strongly fair, meaning they cannot be enabled infinitely often without being taken.

The Environment Processes

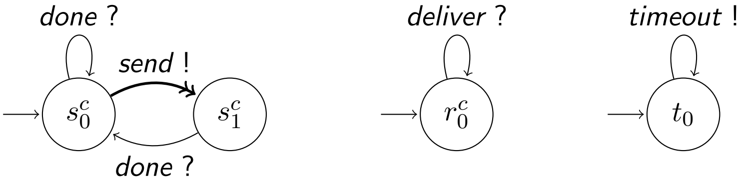
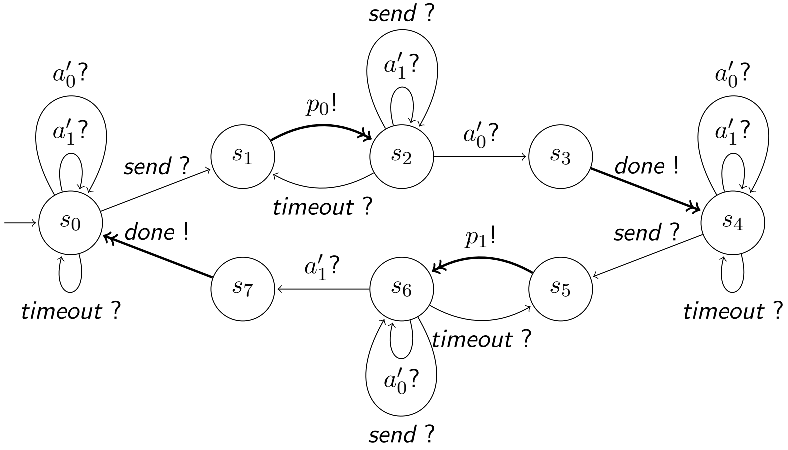


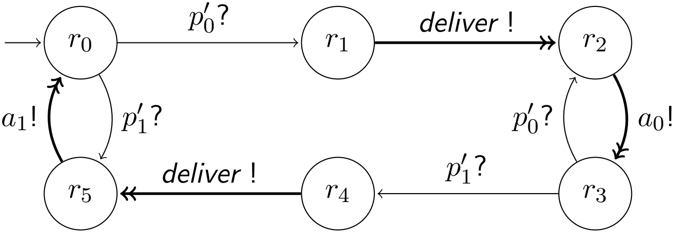
Figure: Environment processes *Sending Client* (left), *Receiving Client* (middle), and *Timer* (right).

ABP Sender



a_0'?
a_1'?
send ?
s_0
timeout ?
done !
s_1
p_0!
timeout ?
send ?
a_1'?
s_2
a_0'?
s_3
done !
a_0'?
a_1'?
s_4
send ?
timeout ?
s_7
a_1'?
s_6
p_1!
s_5
timeout ?
a_0'?
send ?

ABP Receiver



A Blocking Sender

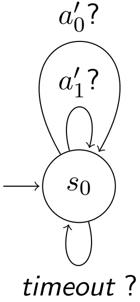


Figure: Blocking Sender: it blocks the *send* event of the Sending Client by not having any transition labeled with that event.

Bibliography

Alur, R. and Tripakis, S. (2017).
Automatic synthesis of distributed protocols.
SIGACT News, 48(1):55–90.

Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
MIT Press.

Hoare, C. (1985).
Communicating Sequential Processes.
Prentice Hall.

Holzmann, G. (2003).
The Spin Model Checker.
Addison-Wesley.

Kahn, G. (1974).
The semantics of a simple language for parallel programming.
In *Information Processing 74, Proceedings of IFIP Congress 74*. North-Holland.

Lynch, N. A. (1996).
Distributed Algorithms.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Milner, R. (1980).
A Calculus of Communicating Systems, volume 92 of *LNCS*.
Springer-Verlag.

Murata, T. (1989).
Petri nets: Properties, analysis and applications.
Proceedings of the IEEE, 77(4):541–580.