Static Analysis of Behavioural Properties of Stream Synchronisers

Anna Tikhonova

May 29, 2014

Contents

1 Mathematical Model **1**
1.1 Examples 2

1 Mathematical Model

Synchroniser $S = (\Phi,\ \Pi)$, where

$\Phi = (A,\ S,\ T)$ – nondeterministic state machine:

$A \subseteq C \times P$ – the alphabet of events, C – set of synchroniser input channels, P – set of predicates on channel messages. An event $(c, p) \in A$ represents the reception of a message on channel c that satisfies predicate p,

$S \supseteq \{s_0\}$ – set of abstract states, s_0 – start state,

$T : A \times S \to S$ – transition matrix.

$\Pi : S \times \Omega \to V$ – path functional that defines the synchroniser output:

Ω – the set of output channels,

V – the set of message values [1].

In a state s_k the functional is based on the retrospective sequence of transitions from the most recent visit to the start state s_0 to s_k:

$(s_0, c_0),\ (s_1, c_1), ...\ (s_k, c_k)$, where

s_0 – start state,

$c_i \in C,\ 0 \le i \le k$ – the channel that caused the transition from the state s_i.

Let μ_i be the message received in the transition from the state s_i. Then

$\Pi\,(s_k,\ \omega_m) = \psi_\sqcap\, \{\mu_i \mid \rho^m_{ki}\,(s_i),\ 0 \le i \le k\}$, where

ρ^m_{ki} – selection predicate that defines Π,

$\psi_\sqcap$ – the operator that coerces the messages in the operand set to their joint greatest subtype.

From the above, the synchroniser is fully defined by two functions:

1. The transition matrix T

The state machine can have a regular structure whereby many transitions can be defined at once by a formula with some limited range integer variables. For example, a machine with 8 states could have a transition matrix defined thus: $s_k \to s_{k+1 \ mod \ 8}$.

In order to be able to use regular structures, AstraKahn allows synchronisers to declare *state* variables.

2. The selection predicate ρ

   In a given state k for each output channel ω_m we note all i on which ρ^m_{ki} is true. Those message values must be stored in a previous state and recalled in state k. It is expected that the Boolean vector $\omega_i = \rho^m_{ki}$ has only very few true elements.

   Consequently the storage mechanism that AstraKahn provides for synchronisers is in the form of individual *store variables*, which are associated with input channel types.

1.1 Examples

1. Zip2. Zip2 receives messages on its input channels and sends their concatenation to the output channel. In the resulting concatenation there's exactly one message from each input channel and those messages are ordered as they received.

   The zip2 transition diagram is given in Figure 1. The message received in the current transition is referred by a keyword *this*. *ma* and *mb* are the store variables associated with the input channels *a* and *b* respectively. The statement *send* indicates a sending of a message to an output channel.

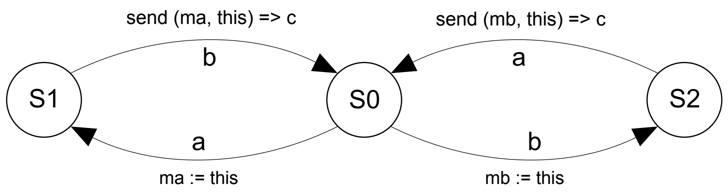

Figure 1: The transition diagram of the zip2 synchroniser.

Mathematical model $S_{zip2} = (\Phi,\ \Pi)$, where

$\Phi = (A,\ S,\ T)$,

$C = (a,\ b),\ P = (true),\ A = ((a, true),\ (b, true)),$

$S = (s_0,\ s_1,\ s_2),\ s_0$ – start state,

T:

$A \setminus S$	s_0	s_1	s_2
$(a, true)$	s_1	s_1	s_0
$(b, true)$	s_2	s_0	s_2

$\Pi : S \times \Omega \rightarrow V,$

$\Omega = (c),$

$V = ((a, b),\ (b, a))$

An output message is emitted when a transition happens either from the state s_1 or the state s_2. These states are reached in two paths:

$W_0 = ((s_0, a),\ (s_1, b))$

$\Pi\ (s_1, c) = \psi_\sqcap\ \{\mu_0 = a \mid \rho^c_{10}\ (s_0) = 1,\ \mu_1 = b \mid \rho^c_{11}\ (s_1) = 1\},\ k = 1,\ i = 0, 1$

$W_1 = ((s_0, b),\ (s_2, a))$

$\Pi\ (s_2, c) = \psi_\sqcap\ \{\mu_0 = b \mid \rho^c_{20}\ (s_0) = 1,\ \mu_2 = a \mid \rho^c_{22}\ (s_2) = 1\},\ k = 2,\ i = 0, 2$

2. Counter. Counter emits every n-th message received in its input channel to the output channel. The transition diagram for the counter synchroniser for $n = 3$ is given in Figure 2.a.

Mathematical model $S_{counter_3} = (\Phi,\ \Pi)$, where

$\Phi = (A,\ S,\ T),$

$C = (a),\ P = (true),\ A = C \times P = ((a, true)),$

$S = (s_0,\ s_1,\ s_2),\ s_0$ – start state,

T:

$A \setminus S$	s_0	s_1	s_2
$(a, true)$	s_1	s_2	s_0

$\Pi : S \times \Omega \rightarrow V,$

$\Omega = (c),$

$V = (a)$

An output message is emitted when a transition happens from the state s_2. This state is reached in a signle path:

$W_0 = ((s_0, a),\ (s_1, a),\ (s_2, a))$

$\Pi\ (s_2, c) = \psi_\sqcap\ \{\mu_0 = a|\rho^c_{10}\ (s_0) = 0, \mu_1 = a|\rho^c_{11}\ (s_1) = 0, \mu_2 = a|\rho^c_{12}\ (s_2) = 1\}, k = 1, i = 0, 1, 2$

The state machine behind the counter has a regular structure, and for this synchroniser all its transitions may be defined with a single formula: $S_{k\ mod\ 3} \rightarrow S_{k+1\ mod\ 3}$. Considering this, the transition matrix T would be:

$A \setminus S$	$S_{k\ mod\ 3}$
$(a, true)$	$S_{k+1\ mod\ 3}$

Some possible transition diagrams of the counter synchroniser are given in Figure 2. The diagram 2.a represents the unrolled regular structure of the synchroniser. However, this representation is inconvenient when $n \gg 1$. The transition diagram 2.a can be folded using state variables. Two possible variants are shown is figures 2.b and 2.c. The state variable c acts as an induction variable ia a while loop with the exit condition $c \geq 3$.

The source code for the transition diargam 2.b is given in the Listing 1.

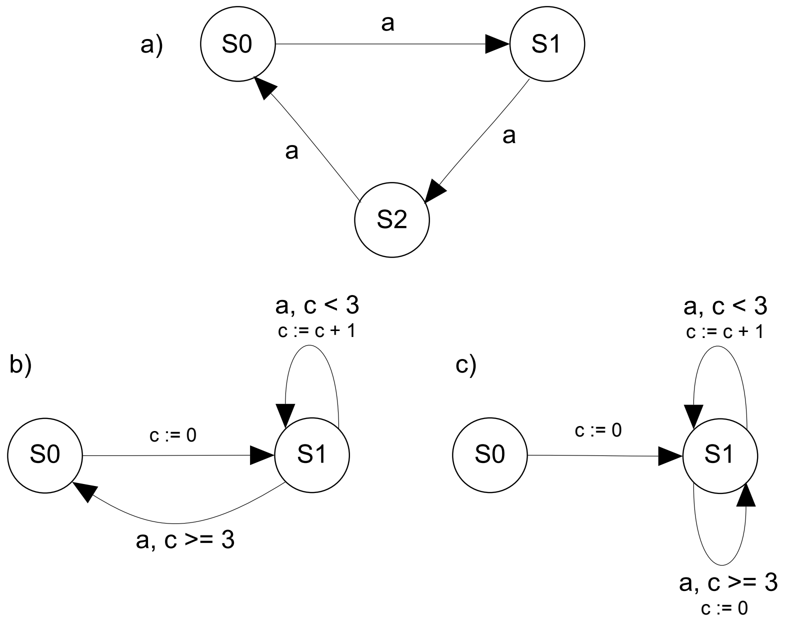

Figure 2: The transition diagrams of the counter synchroniser.

Listing 1: The source code of the counter synchroniser.

```
synch counter (a:0 | c:0)
{
  state int (8) c;

start:      do c := 0
            goto work;

work:   on a && c < 3
            do c := c + 1
            goto work;

        elseon a && c >= 3
            send this => c
            goto start;
}
```

The source code for the zip2 synchroniser (listing 2).

Listing 2: The source code of the zip2 synchroniser.

```
synch zip2 (a:0, b:0 | c:0)
{
  store ma:a, mb:b;

s0:     on a do
```

```
            ma := this
            goto s1;
        on b do
            mb := this
            goto s2;

s1:     on b  send (ma, this) => c    goto s0;
s2:     on a  send (mb, this) => c    goto s0;
}
```

- a
- b

References

[1] Alex Shafarenko. Astrakahn: A coordination language for streaming networks. *CoRR*, abs/1306.6029, 2013.