Simulations

These scripts are for the SPiM Player stochastic π-calculus simulator v1.13 [15], and Matlab 7.4.0 (ode45). The core SPiM syntax maps directly to stochastic π-calculus [17,18]. The SPiM scripts are complete and executable, and usually are a literal translation of the automata in the figures, with some additional code for plotting directives and for test signals. Figure 2 and Figure 36 instead outline the encoding of automata used in the other scripts.

Figure 2: Automata reactions

SPiM encoding of Delay at rate r from state A to state B, then running 100 automata with initial state A:

```
let A() = delay@r; B()   and B() = ...
run 100 of A()
```

Encoding of Interaction: an input ?a from state A1 to A2 and an ouput !a from state B1 to B2, over a channel of rate r, between two concurrent automata initially in states A1 and B1.

```
new a@r:chan
let A1() = ?a; A2()   and A2() = ...
let B1() = !a; B2()   and B2() = ...
run (A1() | B1())
```

SPiM encoding of multiple transitions (a delay, an input, and an output) from the same state A to three different states:

```
let A() = do delay@r; B1() or ?a; B2() or !b; B3()
```

Figure 3: Celebrity automata

```
directive sample 0.1
directive plot A(); B()

new a@1.0:chan
new b@1.0:chan

let A() = do !a; A() or ?a; B()
and B() = do !b; B() or ?b; A()

run 100 of (A() | B())
```

Figure 5: Groupie automata

```
directive sample 2.0
directive plot A(); B()

new a@1.0:chan
new b@1.0:chan

let A() = do !a; A() or ?b; B()
and B() = do !b; B() or ?a; A()

run 100 of (A() | B())
```

Figure 6: Both together

```
directive sample 10.0
directive plot Ag(); Bg(); Ac(); Bc()

new a@1.0:chan
new b@1.0:chan

let Ac() = do !a; Ac() or ?a; Bc()
and Bc() = do !b; Bc() or ?b; Ac()

let Ag() = do !a; Ag() or ?b; Bg()
and Bg() = do !b; Bg() or ?a; Ag()

run 1 of Ac()
run 100 of (Ag() | Bg())
```

Figure 7: Hysteric groupies

```
directive sample 10.0
directive plot A(); B()

new a@1.0:chan
new b@1.0:chan

let A() = do !a; A() or ?b; ?b; ?b; B()
and B() = do !b; B() or ?a; ?a; ?a; A()

let Ad() = !a; Ad()
and Bd() = !b; Bd()

run 100 of (A() | B())
run 1 of (Ad() | Bd())
```

Figure 9: Sequence of delays

```
directive sample 20.0
directive  plot  S1();  S2();  S3();  S4();  S5();  S6();
S7(); S8(); S9(); S10()

let S1() = delay@1.0; S2() and S2() = delay@1.0;S3()
and S3() = delay@1.0; S4() and S4() = delay@1.0; S5()
and S5() = delay@1.0; S6() and S6() = delay@1.0; S7()
and S7() = delay@1.0; S8() and S8() = delay@1.0; S9()
and S9() = delay@1.0; S10() and S10() = ()

run 10000 of S1()
```

Figure 11: All 3 reactions in 1 automaton

```
directive sample 0.02
directive plot A(); B()

new a@1.0:chan
new b@1.0:chan

let A() = do !a; A() or !b; A() or ?b; B()
and B() = do delay@1.0; A() or ?a; A()

run 1000 of  B()
```

Figure 12: Same behavior

```
directive sample 0.02
directive plot A(); B()

new a@1.0:chan
new b@0.5:chan

let A() = do !a; A() or !b; B() or ?b; B()
and B() = do delay@1.0; A() or ?a; A()

run 1000 of  B()
```

Figure 13: Sequence of interactions

```
directive sample 0.02
directive  plot  A1();  A2();  A3();  A4();  A5();  A6();
A7(); A8(); A9(); A10()

new a1@1.0:chan new a2@1.0:chan new a3@1.0:chan
new a4@1.0:chan new a5@1.0:chan new a6@1.0:chan
new a7@1.0:chan new a8@1.0:chan new a9@1.0:chan

let A1() = ?a1; A2() and B1() = !a1; B2()
and A2() = ?a2; A3() and B2() = !a2; B3()
and A3() = ?a3; A4() and B3() = !a3; B4()
and A4() = ?a4; A5() and B4() = !a4; B5()
and A5() = ?a5; A6() and B5() = !a5; B6()
and A6() = ?a6; A7() and B6() = !a6; B7()
and A7() = ?a7; A8() and B7() = !a7; B8()
and A8() = ?a8; A9() and B8() = !a8; B9()
and A9() = ?a9; A10() and B9() = !a9; B10()
and A10() = () and B10() = ()

run 5000 of (A1() | B1())
```

Figure 14: Zero order reactions

```
directive sample 1000.0
directive plot S(); P(); E()

new a@1.0:chan

let E() = !a; delay@1.0; E()
and S() = ?a; P()
and P() = ()

run (1 of  E() | 1000 of S())
```

Figure 15: Subtraction

```
directive sample 20.0 1000
directive plot E(); F()

new a@1.0:chan

let E() = ?a; delay@1.0; E()
and F() = !a; delay@1.0; F()

let raising(p:proc(), t:float) =
(* Produce one p() every t sec with precision dt *)
  (val dt= 100.0 run step(p, t, dt, dt))
and step(p:proc(), t:float, n:float, dt:float) =
   if n<=0.0 then (p()|step(p,t,dt,dt))
   else delay@dt/t; step(p,t,n-1.0,dt)

run 1000 of F()
```

```
run raising(E,0.01)
```

```
directive sample 20.0 1000
directive plot E(); F()

new a@1.0:chan

let E() = ?a; E()
and F() = !a; F()

let raising(p:proc(), t:float) =
  ... see code for Error! Reference source not found.

run 1000 of F()
run raising(E,0.01)
```

Figure 16: Ultrasensitivity

```
directive sample 215.0
directive plot E();F();S();P();ES();FP()

new a@1.0:chan new b@1.0:chan

let S() = ?a; P()
and P() = ?b; S()

let E() = !a; ES()
and ES() = delay@1.0; E()
and F() = !b; FP()
and FP() = delay@1.0; F()

run 1000 of S()

let raising(p:proc(), t:float) =
  ... see code for Error! Reference source not found.

run 100 of F()
run raising(E,1.0)
```

Figure 17: Positive feedback transition

```
directive sample 0.02 1000
directive plot B(); A()

val s=1.0

new b@s:chan
let A() = ?b; B()
and B() = !b;B()

run (1000 of A() | 1 of B())
```

Figure 18: Bell shape

```
directive sample 0.003 1000
directive plot B(); A(); C()

new b@1.0:chan new c@1.0:chan

let A() = ?b; B()
and B() = do !b;B() or ?c; C()
and C() = !c;C()

run ((10000 of A()) | B() | C())
```

Figure 19: Oscillator

```
directive sample 0.03 1000
directive plot A(); B(); C()

new a@1.0:chan new b@1.0:chan new c@1.0:chan
let A() = do !a;A() or ?b; B()
and B() = do !b;B() or ?c; C()
and C() = do !c;C() or ?a; A()

run (900 of A() | 500 of B() | 100 of C())
```

Figure 20: Positive two-stage feedback

```
directive sample 0.1 1000
directive plot B(); A(); A1()

val s=1.0

new c@s:chan
let A() = ?c; A1()
and A1() = ?c; B()
and B() = !c;B()

run (1000 of A() | 1 of B())
```

Figure 21: Square shape

```
directive sample 0.2 1000
directive plot B(); A(); A1(); B1(); C()

new b@1.0:chan new c@1.0:chan

let A() = ?b; A1()
and A1() = ?b; B()
and B() = do !b;B() or ?c; B1()
and B1() = ?c; C()
and C() = !c;C()

run ((1000 of A()) | B() | C())
```

Figure 22: Hysteric 3-way groupies

```
directive sample 0.5 1000
directive plot A(); B(); C()

new a@1.0:chan
new b@1.0:chan
new c@1.0:chan

let A() = do !a; A() or ?c; ?c; C()
and B() = do !b; B() or ?a; ?a; A()
and C() = do !c; C() or ?b; ?b; B()

let Ad() = !a; Ad()
and Bd() = !b; Bd()
and Cd() = !c; Cd()

run 1000 of A()
run 1 of (Ad() | Bd() | Cd())
```

Figure 23: Second-order cascade

```
directive sample 0.03
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; A()
run 100 of A()
```

Figure 24: Zero-order cascade

```
directive sample 0.01
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; Amp_hi(a,b)
  or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; delay@1.0; A()
run 100 of A()
```

```
directive sample 20.0
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; Amp_hi(a,b)
  or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; delay@1.0; A()
run 100 of A()
```

Figure 25: Zero-order transduction

```
directive sample 10.0
directive plot !a; !b

new a@1.0:chan new b@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; Amp_hi(a,b)
  or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 1000 of Amp_lo(a,b)

let A() = !a; delay@1.0; A()
run 900 of A()
```

Figure 26: Second-order double cascade

```
directive sample 0.03
directive plot !a; !b; !c
```

```
new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; ?a; Amp_hi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; A()
run 100 of A()
```

Figure 27: Zero-order double cascade

```
directive sample 0.03
directive plot !a; !b

new a@1.0:chan new b@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; Amp_hi(a,b)
  or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; ?a; Amp_hi(a,b)

run 1000 of Amp_lo(a,b)

let A() = !a; delay@1.0; A()
run 2000 of A()
```

Figure 28: Simple inverter

```
directive sample 110.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b)
  or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  delay@1.0; Inv_hi(a,b)

run 100 of (Inv_hi(a,b) | Inv_lo(b,c))

let clock(t:float, tick:chan) =
   (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
  if n<=0.0 then !tick; clock(t,tick)
  else delay@dt/t; step(tick,t,n-1.0,dt)
let S1(a:chan, tock:chan) =
  do !a; S1(a,tock) or ?tock; ()
and SN(n:int, t:float, a:chan, tick:chan, tock:chan) =
  if n=0 then clock(t, tock)
  else ?tick; (S1(a,tock) | SN(n-1,t,a,tick,tock))
let raisingfalling(a:chan, n:int, t:float) =
   (new tick:chan new tock:chan
    run (clock(t,tick) | SN(n,t,a,tick,tock)))

run raisingfalling(a,100,0.5)
```

```
directive sample 15.0 1000
directive plot !a; !b; !c; !d; !e; !f

new a@1.0:chan new b@1.0:chan new c@1.0:chan
new d@1.0:chan new e@1.0:chan new f@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b)
  or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  delay@1.0; Inv_hi(a,b)

run 100 of (Inv_lo(a,b) | Inv_lo(b,c)
| Inv_lo(c,d) | Inv_lo(d,e) | Inv_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b)
  or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  delay@1.0; Inv_hi(a,b)

run 100 of (Inv_hi(a,b) | Inv_lo(b,c) | Inv_lo(c,a))
```

Figure 29: Feedback inverter

```
directive sample 110.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b) or ?a; Inv_lo(a,b)
and Inv_lo(a:chan, b:chan) =
  do delay@1.0; Inv_hi(a,b)
  or ?b; Inv_hi(a,b)

run 100 of (Inv_hi(a,b) | Inv_lo(b,c))

let raisingfalling(a:chan, n:int, t:float) =
  ... see code for Error! Reference source not found.

run raisingfalling(a,100,0.5)
```

```
directive sample 1.0 1000
directive plot !a; !b; !c; !d; !e; !f

new a@1.0:chan new b@1.0:chan new c@1.0:chan
new d@1.0:chan new e@1.0:chan new f@1.0:chan

let Inv_hi ... and Inv_lo ...
   ... see code above

run 100 of (Inv_lo(a,b) | Inv_lo(b,c)
| Inv_lo(c,d) | Inv_lo(d,e) | Inv_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv_hi ... and Inv_lo ...
   ... see code above

run 100 of (Inv_hi(a,b) | Inv_lo(b,c) | Inv_lo(c,a))
```

Figure 30: Double-height inverter

```
directive sample 110.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b) or ?a; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
and Inv2_lo(a:chan, b:chan) =
  delay@1.0; Inv2_mi(a,b)

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c))

let raisingfalling(a:chan, n:int, t:float) =
  ... see code for Error! Reference source not found.

run raisingfalling(a,100,0.5)
```

```
directive sample 15.0 1000
directive plot !a; !b; !c; !d; !e; !f

new a@1.0:chan new b@1.0:chan new c@1.0:chan
new d@1.0:chan new e@1.0:chan new f@1.0:chan

let Inv2_hi ... and Inv2_lo ...
   ... see code above

run 100 of (Inv2_lo(a,b) | Inv2_lo(b,c)
| Inv2_lo(c,d) | Inv2_lo(d,e) | Inv2_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv2_hi ... and Inv2_lo ...
   ... see code above

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c) | Inv2_lo(c,a))
```

Figure 31: Double-height feedback inverter

```
directive sample 110.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b) or ?a; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
  or ?b; Inv2_hi(a,b)
and Inv2_lo(a:chan, b:chan) =
  do delay@1.0; Inv2_mi(a,b)
  or ?b; Inv2_mi(a,b)

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c))

let raisingfalling(a:chan, n:int, t:float) =
  ... see code for Error! Reference source not found.

run raisingfalling(a,100,0.5)
```

```
directive sample 1.0 1000
directive plot !a; !b; !c; !d; !e; !f

new a@1.0:chan new b@1.0:chan new c@1.0:chan
new d@1.0:chan new e@1.0:chan new f@1.0:chan

let Inv2_hi ... and Inv2_lo ...
   ... see code above
```

```
run 100 of (Inv2_lo(a,b) | Inv2_lo(b,c)
| Inv2_lo(c,d) | Inv2_lo(d,e) | Inv2_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Inv2_hi ... and Inv2_lo ...
    ... see code above

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c) | Inv2_lo(c,a))
```

Figure 32: Or and And

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Or_hi(a:chan, b:chan, c:chan) =
  do !c; Or_hi(a,b,c) or delay@del; Or_lo(a,b,c)
and Or_lo(a:chan, b:chan, c:chan) =
  do ?a; Or_hi(a,b,c) or ?b; Or_hi(a,b,c)

run 1000 of Or_lo(a,b,c)

let clock(t:float, tick:chan) =
   (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
   if n<=0.0 then !tick; clock(t,tick)
   else delay@dt/t; step(tick,t,n-1.0,dt)

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) =
  do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

let many(n:float, p:proc(float), nt:float) =
if n<=0.0 then () else (p(nt) | many(n-1.0, p, nt))

let BoolInputs(n:float, nt:float, m:float, mt:float) =
(run many(n, Sig_a, nt) run many(m, Sig_b, mt))
and Sig_a(nt:float) =
   (new tick:chan run (clock(nt,tick) | S_a(tick)))
and Sig_b(mt:float) =
   (new tick:chan run (clock(mt,tick) | S_b(tick)))

run BoolInputs(100.0, 4.0, 100.0, 2.0)
```

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let And_hi(a:chan, b:chan, c:chan) =
  do !c; And_hi(a,b,c) or delay@del; And_lo_a(a,b,c)
and And_lo_a(a:chan, b:chan, c:chan) =
  do ?a; And_hi(a,b,c) or delay@del; And_lo_b(a,b,c)
and And_lo_b(a:chan, b:chan, c:chan) =
  ?b; And_lo_a(a,b,c)

run 1000 of And_lo_b(a,b,c)

let BoolInputs(n:float, nt:float, m:float, mt:float) =
   ... see code for Error! Reference source not found.

run BoolInputs(100.0, 4.0, 100.0, 2.0)
```

Figure 33: Imply and Xor

```
directive sample 15.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Imply_hi_a(a:chan, b:chan, c:chan) =
  do !c; Imply_hi_a(a,b,c) or ?a; Imply_lo(a,b,c)
and Imply_hi_b(a:chan, b:chan, c:chan) =
  do !c; Imply_hi_b(a,b,c)
  or delay@del; Imply_lo(a,b,c)
and Imply_lo(a:chan, b:chan, c:chan) =
  do ?b; Imply_hi_b(a,b,c)
  or delay@del; Imply_hi_a(a,b,c)

run 1000 of Imply_lo(a,b,c)

let BoolInputs(n:float, nt:float, m:float, mt:float) =
   ... see code for Error! Reference source not found.

run BoolInputs(100.0, 4.0, 100.0, 2.0)
```

```
directive sample 20.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Xor_hi_a(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c)
  or delay@1.0; Xor_lo_a(a,b,c)
and Xor_hi_b(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c)
  or delay@1.0; Xor_lo_b(a,b,c)
and Xor_lo_a(a:chan, b:chan, c:chan) =
  do ?a; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c)
and Xor_lo_b(a:chan, b:chan, c:chan) =
  do ?b; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c)
and Xor_lo_ab(a:chan, b:chan, c:chan) =
  do delay@1.0; Xor_hi_a(a,b,c)
  or delay@1.0; Xor_hi_b(a,b,c)

run 500 of (Xor_lo_a(a,b,c) | Xor_lo_b(a,b,c))

let BoolInputs(n:float, nt:float, m:float, mt:float) =
  ... see code for Error! Reference source not found.

run BoolInputs(100.0, 8.0, 100.0, 4.0)
```

Figure 34: Memory Elements

```
(* Top Left, Top Center *)
directive sample 0.1
directive plot A(); B(); C()

new a@1.0:chan
new b@1.0:chan

let A() = do !a; A() or ?b; C()
and C() = do ?a; A() or ?b; B()
and B() = do !b; B() or ?a; C()

run 100 of (A() | B())
```

```
(* Bottom Left *)
directive sample 1.0
directive plot A(); B(); C()

new a@1.0:chan
new b@1.0:chan

let A() = do !a; A() or ?b; C()
and C() = do ?a; A() or ?b; B()
and B() = do !b; B() or ?a; C()

let Ad() = !a; Ad()
and Bd() = !b; Bd()

run 100 of (A() | B())
run 10 of (Ad() | Bd())
```

```
(* Bottom Center *)
directive sample 0.6
directive plot A(); B(); C()

new a@1.0:chan
new b@1.0:chan

let A() = do !a; A() or ?b; C()
and C() = do ?a; A() or ?b; B()
and B() = do !b; B() or ?a; C()

let Ad() = !a; Ad()

run 100 of (A() | B())
run 100 of delay@10.0; delay@10.0; delay@10.0;
           delay@10.0; delay@10.0;  Ad()
```

Figure 35: Discrete vs. Continuous Modeling

```
(* Top Left *)                  initially
(A)  dx1/dt = -(x1-x2)          x1 = 2000.0
(B)  dx2/dt = (x1-x2)           x2 = 0.0
```

```
(* Top Center *)                initially
(A)  dx1/dt=x1*x4-x3*x1-x1+x4   x1 = 2000.0
(A') dx2/dt=x3*x1-x3*x2+x1-x2   x2 = 0.0
(B)  dx3/dt=x3*x2-x1*x3-x3+x2   x3 = 0.0
(B') dx4/dt=x1*x3-x1*x4+x3-x4   x4 = 0.0
```

```
(* Top Right *)                 initially
(A)  dx1/dt=x1*x6-x3*x1-x1+x6   x1 = 2000.0
(A') dx2/dt=x3*x1-x3*x2+x1-x2   x2 = 0.0
(A") dx5/dt=x3*x2-x3*x5+x2-x5   x5 = 0.0
(B)  dx3/dt=x3*x5-x1*x3-x3+x5   x3 = 0.0
(B') dx4/dt=x1*x3-x1*x4+x3-x4   x4 = 0.0
(B") dx6/dt=x1*x4-x1*x6+x4-x6   x6 = 0.0
```

```
(* Bottom Left *)
directive sample 5.0 1000
directive plot B(); A()

new a@1.0:chan
new b@1.0:chan

let A() = do !a; A() or ?b; B()
```

```
and B() = do !b; B() or ?a; A()

let Ad() = !a; Ad()
and Bd() = !b; Bd()

run 2000 of A()
run 1 of (Ad() | Bd())
```

```
(* Bottom Center *)
Same as Bottom Left, except:
let A() = do !a; A() or ?b; ?b; B()
and B() = do !b; B() or ?a; ?a; A()
```

```
(* Bottom Right *)
Same as Bottom Left, except:
let A() = do !a; A() or ?b; ?b; ?b; B()
and B() = do !b; B() or ?a; ?a; ?a; A()
```

Figure 36: Polyautomata reactions

SPiM encoding of Association over channel $a@r_0,r_1$ of arity 1, with one automaton performing an output from state A to A1 and the other automaton performing an input from state B to B1:

```
new a@r0:chan(chan)
let A() = (new k1@r1:chan run !a(k1); A1(k1))
and B() = ?a(k1); B1(k1)
```

Encoding of Dissociation through the previously shared k1.

```
and A1(k1:chan) = !k1; A()
and B1(k1:chan) = ?k1; B()
```

More generally, for $a@r_0,...,r_{n-1}$ we declare an (n-1)-ary channel:

```
new a@r0:chan(chan,...,chan)      (*n-1 times*)
```

Association then creates n-1 shared dissociation channels:

```
let A() = (new k1@r1:chan ... new kn-1@rn-1:chan
            run !a(k1,...,kn); A1(k1,...,kn))
```

and then A1 can choose which channel to use for dissociation. Note that the constraint about not reassociating before a dissociation is not automatically enforced by this encoding.

Figure 37: Complexation/decomplexation

```
directive sample 0.005
directive plot !A_f; !A_b; !B_f; !B_b
new A_f:chan new A_b:chan new B_f:chan new B_b:chan

val mu = 1.0    val lam = 1.0
new a@mu:chan(chan)

let Af() = (new k@lam:chan run do !a(k); Ab(k) or !A_f)
and Ab(k:chan) = do !k; Af() or !A_b

let Bf() = do ?a(k); Bb(k) or !B_f
and Bb(k:chan) = do ?k; Bf() or !B_b

run (1000 of Af() | 500 of Bf())
```

Figure 38: Enzymatic reactions

```
directive sample 0.05 1000
directive plot !E_f; !E_b; !S_f; !S_b; !P_
new E_f:chan new E_b:chan
new S_f:chan new S_b:chan new P_:chan

val r0 = 1.0    val r1 = 1.0    val r2 = 100.0
new a@r0:chan(chan,chan)

let P() = !P_

let Ef() =
    (new k1@r1:chan new k2@r2:chan
     run do !a(k1,k2); Eb(k1,k2) or !E_f)
and Eb(k1:chan,k2:chan) =
  do !k1; Ef() or !k2; Ef() or !E_b

let Sf() = do ?a(k1,k2); Sb(k1,k2) or !S_f
and Sb(k1:chan,k2:chan) =
  do ?k1; Sf() or ?k2; P() or !S_b

run (1000 of Ef() | 2000 of Sf())
```

Figure 39: Homodimerization

```
directive sample 0.005 10000
directive plot !A_f; !A_i; !A_o
new A_f:chan new A_i:chan new A_o:chan

new a@1.0:chan(chan)

let Af() =
   (new k@1.0:chan
    run do ?a(k); Ai(k) or !a(k); Ao(k) or !A_f)

and Ai(k:chan) = do ?k; Af() or !A_i

and Ao(k:chan) = do !k; Af() or !A_o

run 1000 of Af()
```

Figure 40: Bidirectional polymerization

```
directive sample 1000.0
directive plot ?count

type Link = chan(chan)
type Barb = chan

val lam = 1000.0 (* set high for better counting *)
val mu = 1.0
new c@mu:chan(Link)
new enter@lam:chan(Barb)
new count@lam:Barb

let Af() =
   (new rht@lam:Link run
    do !c(rht); Ar(rht)
    or ?c(lft); Al(lft))

and Al(lft:Link) =
   (new rht@lam:Link run
    !c(rht); Ab(lft,rht))

and Ar(rht:Link) =
    ?c(lft); Ab(lft,rht)

and Ab(lft:Link, rht:Link) =
   do ?enter(barb); (?barb | !rht(barb))
   or ?lft(barb); (?barb | !rht(barb))
(* each Abound waits for a barb, exhibits it, and
passes it to the right so we can plot number of Abound
in a ring *)

let clock(t:float, tick:chan) =
   (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
   if n<=0.0 then !tick; clock(t,tick) else delay@dt/t;
step(tick,t,n-1.0,dt)

new tick:chan
let Scan() = ?tick; !enter(count); Scan()

run 1000 of Af()
run (clock(100.0, tick) | Scan())
```

Figure 42: Actin-like polymerization

```
directive sample 0.01 (* 0.25, 35.0 *) 1000
directive plot !A_f; !A_l; !A_r; !A_b
new A_f:chan new A_l:chan new A_r:chan new A_b:chan

val lam = 1.0 (* dissoc *)
val mu = 1.0 (* assoc *)
new c@mu:chan(chan)

let Af() =
   (new lft@lam:chan run
    do !c(lft); Al(lft)
    or ?c(rht); Ar(rht) or !A_f)

and Al(lft:chan) =
    do !lft; Af()
    or ?c(rht); Ab(lft,rht) or !A_l

and Ar(rht:chan) =
    do ?rht; Af() or !A_r

and Ab(lft:chan, rht:chan) =
    do !lft; Ar(rht) or !A_b

run 1000 of Af()
```