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 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@1.0; A() and B() = ...
run 100 of A()
Encoding of Interaction: an input ?a from state A1 to A2 and an output !a from state B1 to B2, over a channel of rate r, between two concurrent automata initially in states A1 and B1:
new a@1.0:chan new b@1.0: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()

directive plot A(); B(); C(); D()
directive sample 20.0
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 !b; A() and B() = do ?a; B() or 7b; 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 7a; A()
run 100 of (A() | B())
Figure 6: Both together
directive sample 10.0
directive plot A(); B(); C(); D()
new a@1.0:chan new b@1.0:chan
let Ac() = do !a; Ac() or ?a; Bc() and Bc() = do !b; Bc() or 7b; Ac() and Ag() = do !a; Ag() or ?b; Bg() and Bg() = do !b; Bg() or 7a; Ag()
run 1 of Ac()
run 100 of (Ag() | Bg())
Figure 7: Hysteric groupies
directive sample 10.0
directive plot A(); B(); C(); D()
new a@1.0:chan new b@1.0:chan
let A() = do !a; A() or ?b; B() or 7b; B() and B() = do !b; B() or 7a; 7a; A()
let Ad() = !a; Ad() and Bd() = !b; Bd()
run 100 of (Ad() | Bd())
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()
run 10000 of S1()

Figure 12: Same behavior
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 7b; B() and B() = do delay@1.0; A() or 7a; A()
run 1000 of B()
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()
and E() = !a; ES()
and ES() = delay@1.0; E()
and F() = !b; FP()
and FP() = delay@1.0; F()
run 1000 of S()
run raising(E,0.01)

Figure 17: Positive feedback transition
directive sample 0.02 1000
directive plot B(); A()
val s=1.0
new b@1.0: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 a@1.0:chan new c@1.0:chan
let A() = ?c; A()
and A1() = ?c; B()
and B() = !c;B()
run 1000 of A()
run 100 of A()

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() = !c;C()
run ((1000 of A()) | B()) | C())

Figure 20: Positive two-stage feedback
directive sample 0.1 1000
directive plot B(); A(); A1()
val s=1.0
new c@1.0:chan
let A() = !c; A()
and A1() = ?c; B()
and B() = !b;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; A()
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: Hysteretic 3-way groupies
directive sample 0.5 1000
directive plot A(); B(); C()
new a@1.0:chan new b@1.0:chan
let A() = do !a; A() or 7c; 7c; C()
and B() = do !b; B() or 7a; 7a; A()
and C() = do !c; C() or 7b; 7b; B()
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) = 7a; 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) = 7a; 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) = 7a; 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 26: Second-order double cascade
directive sample 0.03
directive plot !a; !b; !c
new a@1.0:chan new b@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) = 7a; Amp_hi(a,b)
run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))
let A() = !a; delay@1.0; A()
run 200 of A()
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 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; 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) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Inv_lo(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Inv_mi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Amp_hi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Amp_lo(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Amp_mi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Inv_hi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Inv_lo(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Inv_mi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Amp_hi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Amp_lo(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Amp_mi(a:chan, b:chan) = new a@1.0:chan new b@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 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 delay@1.0; Inv_lo(a,b)
or delay@1.0; Inv_mi(a,b)
run 1000 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) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Inv2_lo(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Inv2_mi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Amp2_hi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Amp2_lo(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Amp2_mi(a:chan, b:chan) = new a@1.0:chan new b@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 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 delay@1.0; Inv2_lo(a,b)
or delay@1.0; Inv2_mi(a,b)
or delay@1.0; Inv2_mi(a,b)
run 1000 of (Inv2_hi(a,b) | Inv2_lo(b,c) | Inv2_lo(c,a))

let raisingfalling(a:chan, n:int, t:float) = ... see code for Error! Reference source not found.
run raisingfalling(a,100,0.5)

... see code above

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

Figure 32: 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 Inv3_hi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Inv3_lo(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Inv3_mi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Amp3_hi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Amp3_lo(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Amp3_mi(a:chan, b:chan) = new a@1.0:chan new b@1.0:chan
let Inv_hi ... and Inv_lo ...

... see code above

run 100 of (Inv3_hi(a,b) | Inv3_lo(b,c) | Inv3_lo(c,a))

... see code above

run 100 of (Inv3_lo(a,b) | Inv3_lo(b,c) | Inv3_lo(c,a))

... see code above
directive sample 2.0 1000

directive plot la; lb; lc

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

let Inv 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 ?a; Or_hi (a, b, c) or delay@del; Or_lo (a, b, c)

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

run 1000 of Or_lo (a, b, c)

clock t:float, tick: chan =
  (val dt=100.0 step (tick, t, dt, dt))

let S a@float, tick: chan =
  do !a; S a (tick) or ?tick;

let S b@float, tick: chan =
  do !b; S b (tick) or ?tick;

let many n: float, p: proc (float), nt: float =
  (run many (n, S a, nt) run many (n, S b, nt))

let sig a@float, nt: float =
  (new tick: chan run (clock (nt, tick) | S_a (tick)))

let sig b@float, nt: float =
  (new tick: chan run (clock (nt, tick) | S_b (tick)))

run BoolInputs (100.0, 4.0, 100.0, 2.0)

directive sample 10.0 1000

directive plot la; lb; lc

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 !a; And_hi (a, b, c) or delay@del; And_lo (a, b, c)

let And_lo (a: chan, b: chan, c: chan) =
  do ?a; And_lo (a, b, c) or delay@del; And_hi (a, b, c)

run 1000 of And_lo (a, b, c)

clock t:float, tick: chan =
  (val dt=100.0 step (tick, t, dt, dt))

let S a@float, tick: chan =
  do !a; S a (tick) or ?tick;

let S b@float, tick: chan =
  do !b; S b (tick) or ?tick;

let many n: float, p: proc (float), nt: float =
  (run many (n, S a, nt) run many (n, S b, nt))

let sig a@float, nt: float =
  (new tick: chan run (clock (nt, tick) | S_a (tick)))

let sig b@float, nt: float =
  (new tick: chan run (clock (nt, tick) | S_b (tick)))

run BoolInputs (100.0, 4.0, 100.0, 2.0)

directive sample 15.0 1000

directive plot la; lb; lc

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 !a; And_hi (a, b, c) or delay@del; And_lo (a, b, c)

let And_lo (a: chan, b: chan, c: chan) =
  do ?a; And_lo (a, b, c) or delay@del; And_hi (a, b, c)

run 1000 of And_lo (a, b, c)

clock t:float, tick: chan =
  (val dt=100.0 step (tick, t, dt, dt))

let S a@float, tick: chan =
  do !a; S a (tick) or ?tick;

let S b@float, tick: chan =
  do !b; S b (tick) or ?tick;

let many n: float, p: proc (float), nt: float =
  (run many (n, S a, nt) run many (n, S b, nt))

let sig a@float, nt: float =
  (new tick: chan run (clock (nt, tick) | S_a (tick)))

let sig b@float, nt: float =
  (new tick: chan run (clock (nt, tick) | S_b (tick)))

run BoolInputs (100.0, 4.0, 100.0, 2.0)

directive sample 20.0 1000

directive plot la; lb; lc

new a@1.0:chan new b@1.0:chan new c@1.0:chan
let Xor_hi (a: chan, b: chan, c: chan) =
  do !a; Xor_hi (a, b, c) or delay@1.0; Xor_lo (a, b, c)

let Xor_lo (a: chan, b: chan, c: chan) =
  do ?a; Xor_lo (a, b, c) or delay@1.0; Xor_hi (a, b, c)

run 500 of (Xor_lo (a, b, c) | Xor_hi (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

directive sample 0.1

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 ?a; C()

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

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

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

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

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

and C() = do ?a; A() or ?b; B()
and B() = do !b; B() or ?a; A()
let Ad() = 1a; Ad()
and Bd() = 1b; Bd()
run 2000 of A()
run 1 of (Ad() | Bd())

(* Bottom Center *)
Same as Bottom Left, except:
let A() = do 1a; A() or 7b; 7b; B())
and B() = do 1b; B() or 7a; 7a; A())

(* Bottom Right *)
Same as Bottom Left, except:
let A() = do 1a; A() or 7b; 7b; B())
and B() = do 1b; B() or 7a; 7a; A())

Figure 36: Polyautomata reactions
SPIM encoding of Association over channel a@r1, 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)
let A() = (new k@r0:chan run !a(k1); A(k1))
and B() = (new k@r0:chan run ?a(k1); B(k1))

Encoding of Dissociation through the previously shared k1.
and Al(k1:chan) = !k1; A()
and Bl(k1:chan) = ?k1; B()

More generally, for a@r1...rn, we declare an (n-1)-ary channel:
new a@r1(chan,...,chan) (*n-1 times*)

Association then creates n-1 shared dissociation channels:
let A() = (new k@r1:chan ... new k@rn:chan run l1(a(k1),...,k1); Al(k1),...,rn)
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
let A() = (new k@lam:chan run !a(k1); A(k1))
and B() = (new k@lam:chan run ?a(k1); B(k1))

Figure 38: Enzymatic reactions
directive sample 0.005 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 F:chan
val r0 = 1.0 val r1 = 1.0 val r2 = 100.0
new a@r0(chan,chan)
let F() = !P_
let Ef() = (new k@r1:chan run !a(k1); !E_f)
and Eb(k1:chan,k2:chan) = !a(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) = ?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_b; !A_o
new A_f:chan new A_b:chan new A_o:chan
new a@1.0:chan
let A() = (new k@1.0:chan run do ?a(k1); !A(k1) or !a(k1); Ao(k1) or !A_o)
and Ai(k:chan) = do ?k; Af() or !A_i
and Ao(k:chan) = do !k; Af() or !A_o
run 1000 of Af()