

Stream semantics of synchronous block-diagrams

Application to a formally verified compiler

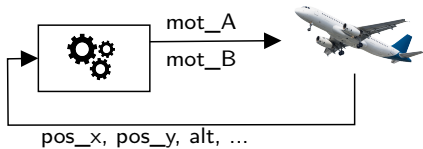
Paul Jeanmaire

Lip6, Sorbonne Université

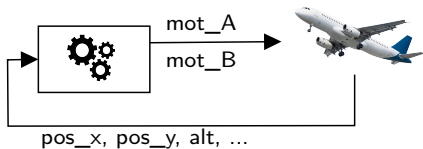
Séminaire Cambium, 14 novembre 2025



Synchronous reactive programming



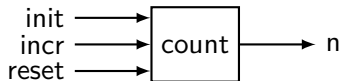
Synchronous reactive programming



every **trigger** do:
 read_inputs()
 compute()
 write_outputs()

The language Lustre

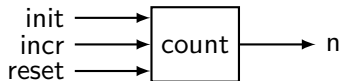
[Caspi, Pilaud and Plaice (1987), LUSTRE: a declarative language for programming synchronous systems]



```
node COUNT (init, incr: int; reset: bool)
  returns (n: int);
let
  n = init ->
    if reset then init else pre(n) + incr;
tel;
```

The language Lustre

[Caspi, Pilaud and Plaice (1987), LUSTRE: a declarative language for programming synchronous systems]



```
node COUNT (init, incr: int; reset: bool)
  returns (n: int);
let
  n = init ->
    if reset then init else pre(n) + incr;
tel;
```

init	0	0	0	0	0	0	0	...
incr	2	2	2	2	2	2	2	...
reset	F	F	F	F	T	F	F	...
n	0	2	4	6	0	2	4	...

Executable block-diagrams = "Model-Based Development"

The Scade Suite

The screenshot displays the Scade IDE interface for a project named "Wheelchair.vsw". The main workspace shows a complex control system block diagram. The diagram includes several feedback loops (FBY), gain blocks (X), summing junctions (+, -), and saturation blocks (saturate). Inputs include position (pos_l) and velocity (vref). Outputs include motor commands (v_cmd, d_cmd) and wheel velocities (v_l, v_r). The diagram is divided into two main functional blocks: "Manual" and "Automatic".

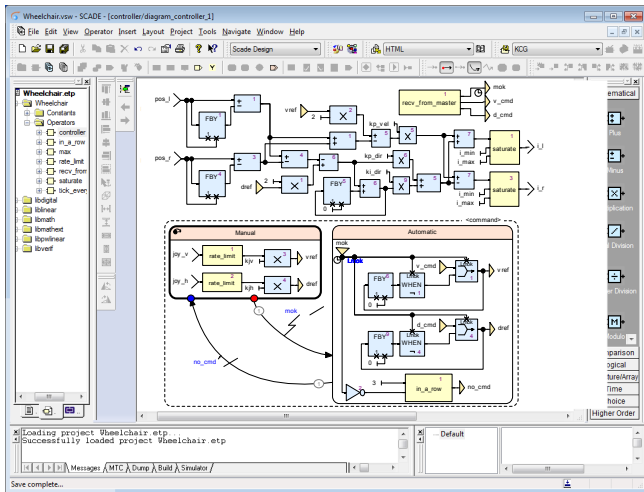
The "Manual" block contains two parallel control paths for left and right wheels, each consisting of a feedback loop (FBY) and a gain block (X). The "Automatic" block contains a more complex control system with feedback loops (FBY), gain blocks (X), and saturation blocks (saturate). It also includes a block for "in_a_row" and a "no_cmd" output.

The interface includes a menu bar (File, Edit, View, Operator, Insert, Layout, Project, Tools, Navigate, Window, Help), a toolbar, and a project tree on the left. The project tree shows the following structure:

- Wheelchair.etp
 - Constants
 - Operator
 - controller
 - in_a_row
 - max
 - rate_limt
 - recv_from
 - saturate
 - tick_ever
 - lbdigital
 - lblearn
 - lbrmath
 - lbrmathset
 - lbrpnlnear
 - lbrvref

Executable block-diagrams = “Model-Based Development”

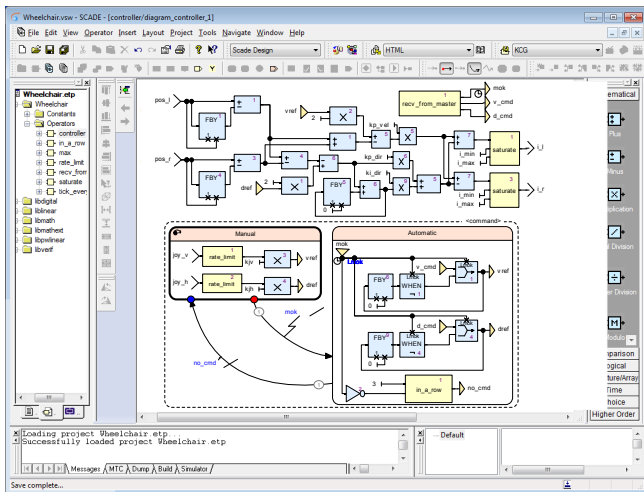
The Scade Suite



block/node = system
line = signal

Executable block-diagrams = “Model-Based Development”

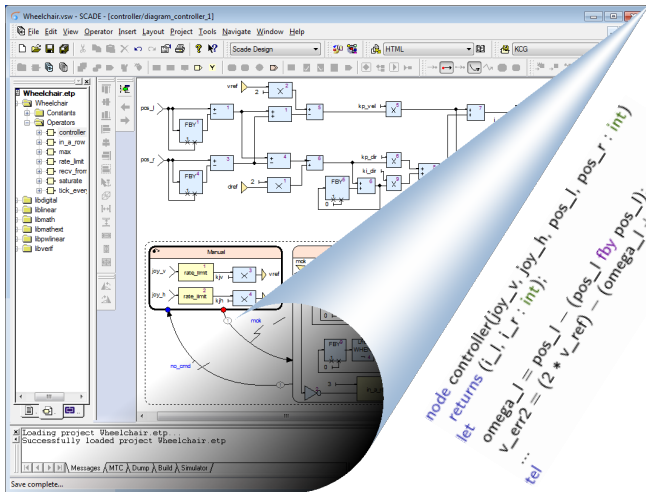
The Scade Suite



block/node = system = stream function
line = signal = stream of values

Executable block-diagrams = "Model-Based Development"

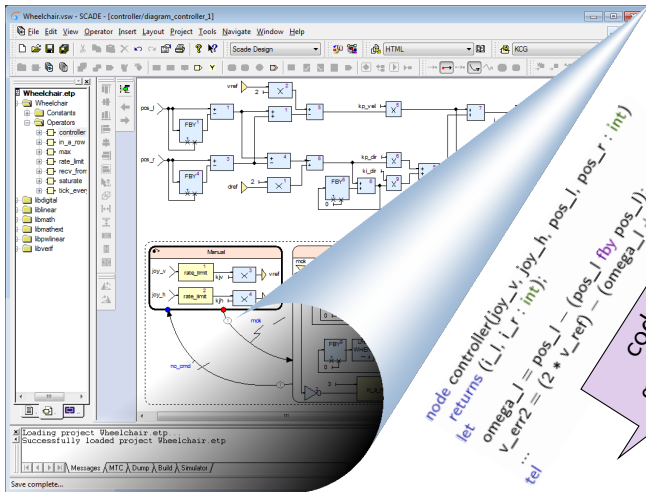
The Scade Suite



block/node = system = stream function
line = signal = stream of values

Executable block-diagrams = "Model-Based Development"

The Scade Suite



```
node controller(joy_v, joy_h, pos_l, pos_r : int)
returns (l_l, l_r : int);
let
  omega_l = pos_l - (pos_l fby pos_l);
  v_err2 = (2 * v_ref) - (omega_l + omega_r);
  ...
tel
```

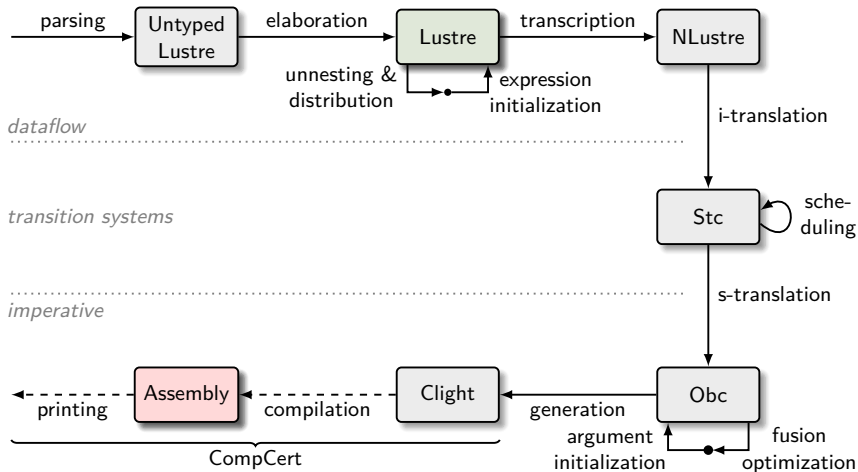
code generator
sequential program
(C, Ada, assembly)

block/node = system = stream function
line = signal = stream of values

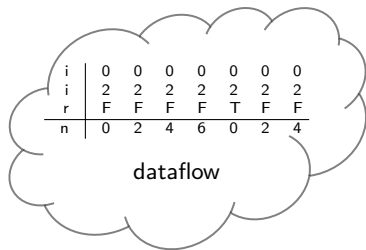
Vélus, a formally verified compiler for Lustre/Scade

<https://velus.inria.fr/>

8 people involved, 3 PhDs, 10 articles, +100k LoC



Semantic preservation



Lustre

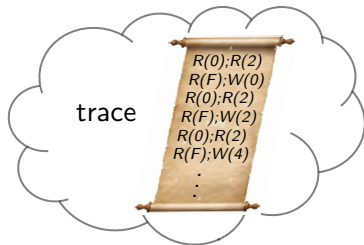
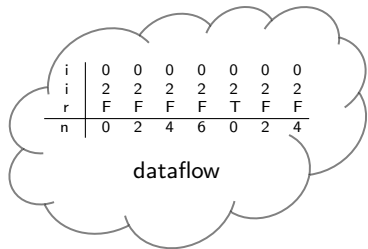
```
node WATCHDOG (set, reset: bool; delay: int)
  returns (alarm: bool);
var remaining_delay: int; deadline: bool;
set
  alarm = WATCHDOG(set, reset, deadline);
  deadline = EDGE(remaining_delay = 0);
  remaining_delay = if set then delay else
    (0 -> pre(remaining_delay) - 1);
set
node WATCHDOG2 (set, reset, time_unit: bool;
  delay: int)
  returns (alarm: bool);
var clock: bool; top
  alarm = current(WATCHDOG2
    (set, reset, delay) when clock);
  clock = true -> set or reset or time_unit;
set
```

Vélus + CompCert

Assembly

```
FunctionCompCert_alarm
  int alarm;
  int top;
  loop 00, 00, 00
  loop 10(0x0), 10(0x0)
  loop 20, 20, 20(0x0)
  loop 30, 30, 30
  loop 40(0x0), 40(0x0)
  loop 50, 50, 50
  loop 60(0x0), 60(0x0)
  loop 70, 70, 70
  loop 80, 80, 80
  loop 90(0x0), 90(0x0)
  loop 100, 100, 100
  loop 110(0x0), 110(0x0)
  loop 120, 120, 120
  loop 130, 130, 130
  loop 140(0x0), 140(0x0)
  loop 150, 150, 150
  loop 160, 160, 160
  loop 170(0x0), 170(0x0)
  loop 180, 180, 180
  loop 190(0x0), 190(0x0)
  loop 200, 200, 200
  loop 210(0x0), 210(0x0)
  loop 220, 220, 220
  loop 230(0x0), 230(0x0)
  loop 240, 240, 240
  loop 250(0x0), 250(0x0)
  loop 260, 260, 260
  loop 270(0x0), 270(0x0)
  loop 280, 280, 280
  loop 290(0x0), 290(0x0)
  loop 300, 300, 300
  loop 310(0x0), 310(0x0)
  loop 320, 320, 320
  loop 330(0x0), 330(0x0)
  loop 340, 340, 340
  loop 350(0x0), 350(0x0)
  loop 360, 360, 360
  loop 370(0x0), 370(0x0)
  loop 380, 380, 380
  loop 390(0x0), 390(0x0)
  loop 400, 400, 400
  loop 410(0x0), 410(0x0)
  loop 420, 420, 420
  loop 430(0x0), 430(0x0)
  loop 440, 440, 440
  loop 450(0x0), 450(0x0)
  loop 460, 460, 460
  loop 470(0x0), 470(0x0)
  loop 480, 480, 480
  loop 490(0x0), 490(0x0)
  loop 500, 500, 500
  loop 510(0x0), 510(0x0)
  loop 520, 520, 520
  loop 530(0x0), 530(0x0)
  loop 540, 540, 540
  loop 550(0x0), 550(0x0)
  loop 560, 560, 560
  loop 570(0x0), 570(0x0)
  loop 580, 580, 580
  loop 590(0x0), 590(0x0)
  loop 600, 600, 600
  loop 610(0x0), 610(0x0)
  loop 620, 620, 620
  loop 630(0x0), 630(0x0)
  loop 640, 640, 640
  loop 650(0x0), 650(0x0)
  loop 660, 660, 660
  loop 670(0x0), 670(0x0)
  loop 680, 680, 680
  loop 690(0x0), 690(0x0)
  loop 700, 700, 700
  loop 710(0x0), 710(0x0)
  loop 720, 720, 720
  loop 730(0x0), 730(0x0)
  loop 740, 740, 740
  loop 750(0x0), 750(0x0)
  loop 760, 760, 760
  loop 770(0x0), 770(0x0)
  loop 780, 780, 780
  loop 790(0x0), 790(0x0)
  loop 800, 800, 800
  loop 810(0x0), 810(0x0)
  loop 820, 820, 820
  loop 830(0x0), 830(0x0)
  loop 840, 840, 840
  loop 850(0x0), 850(0x0)
  loop 860, 860, 860
  loop 870(0x0), 870(0x0)
  loop 880, 880, 880
  loop 890(0x0), 890(0x0)
  loop 900, 900, 900
  loop 910(0x0), 910(0x0)
  loop 920, 920, 920
  loop 930(0x0), 930(0x0)
  loop 940, 940, 940
  loop 950(0x0), 950(0x0)
  loop 960, 960, 960
  loop 970(0x0), 970(0x0)
  loop 980, 980, 980
  loop 990(0x0), 990(0x0)
  loop 1000, 1000, 1000
  loop 1010(0x0), 1010(0x0)
  loop 1020, 1020, 1020
  loop 1030(0x0), 1030(0x0)
  loop 1040, 1040, 1040
  loop 1050(0x0), 1050(0x0)
  loop 1060, 1060, 1060
  loop 1070(0x0), 1070(0x0)
  loop 1080, 1080, 1080
  loop 1090(0x0), 1090(0x0)
  loop 1100, 1100, 1100
  loop 1110(0x0), 1110(0x0)
  loop 1120, 1120, 1120
  loop 1130(0x0), 1130(0x0)
  loop 1140, 1140, 1140
  loop 1150(0x0), 1150(0x0)
  loop 1160, 1160, 1160
  loop 1170(0x0), 1170(0x0)
  loop 1180, 1180, 1180
  loop 1190(0x0), 1190(0x0)
  loop 1200, 1200, 1200
  loop 1210(0x0), 1210(0x0)
  loop 1220, 1220, 1220
  loop 1230(0x0), 1230(0x0)
  loop 1240, 1240, 1240
  loop 1250(0x0), 1250(0x0)
  loop 1260, 1260, 1260
  loop 1270(0x0), 1270(0x0)
  loop 1280, 1280, 1280
  loop 1290(0x0), 1290(0x0)
  loop 1300, 1300, 1300
  loop 1310(0x0), 1310(0x0)
  loop 1320, 1320, 1320
  loop 1330(0x0), 1330(0x0)
  loop 1340, 1340, 1340
  loop 1350(0x0), 1350(0x0)
  loop 1360, 1360, 1360
  loop 1370(0x0), 1370(0x0)
  loop 1380, 1380, 1380
  loop 1390(0x0), 1390(0x0)
  loop 1400, 1400, 1400
  loop 1410(0x0), 1410(0x0)
  loop 1420, 1420, 1420
  loop 1430(0x0), 1430(0x0)
  loop 1440, 1440, 1440
  loop 1450(0x0), 1450(0x0)
  loop 1460, 1460, 1460
  loop 1470(0x0), 1470(0x0)
  loop 1480, 1480, 1480
  loop 1490(0x0), 1490(0x0)
  loop 1500, 1500, 1500
  loop 1510(0x0), 1510(0x0)
  loop 1520, 1520, 1520
  loop 1530(0x0), 1530(0x0)
  loop 1540, 1540, 1540
  loop 1550(0x0), 1550(0x0)
  loop 1560, 1560, 1560
  loop 1570(0x0), 1570(0x0)
  loop 1580, 1580, 1580
  loop 1590(0x0), 1590(0x0)
  loop 1600, 1600, 1600
  loop 1610(0x0), 1610(0x0)
  loop 1620, 1620, 1620
  loop 1630(0x0), 1630(0x0)
  loop 1640, 1640, 1640
  loop 1650(0x0), 1650(0x0)
  loop 1660, 1660, 1660
  loop 1670(0x0), 1670(0x0)
  loop 1680, 1680, 1680
  loop 1690(0x0), 1690(0x0)
  loop 1700, 1700, 1700
  loop 1710(0x0), 1710(0x0)
  loop 1720, 1720, 1720
  loop 1730(0x0), 1730(0x0)
  loop 1740, 1740, 1740
  loop 1750(0x0), 1750(0x0)
  loop 1760, 1760, 1760
  loop 1770(0x0), 1770(0x0)
  loop 1780, 1780, 1780
  loop 1790(0x0), 1790(0x0)
  loop 1800, 1800, 1800
  loop 1810(0x0), 1810(0x0)
  loop 1820, 1820, 1820
  loop 1830(0x0), 1830(0x0)
  loop 1840, 1840, 1840
  loop 1850(0x0), 1850(0x0)
  loop 1860, 1860, 1860
  loop 1870(0x0), 1870(0x0)
  loop 1880, 1880, 1880
  loop 1890(0x0), 1890(0x0)
  loop 1900, 1900, 1900
  loop 1910(0x0), 1910(0x0)
  loop 1920, 1920, 1920
  loop 1930(0x0), 1930(0x0)
  loop 1940, 1940, 1940
  loop 1950(0x0), 1950(0x0)
  loop 1960, 1960, 1960
  loop 1970(0x0), 1970(0x0)
  loop 1980, 1980, 1980
  loop 1990(0x0), 1990(0x0)
  loop 2000, 2000, 2000
  loop 2010(0x0), 2010(0x0)
  loop 2020, 2020, 2020
  loop 2030(0x0), 2030(0x0)
  loop 2040, 2040, 2040
  loop 2050(0x0), 2050(0x0)
  loop 2060, 2060, 2060
  loop 2070(0x0), 2070(0x0)
  loop 2080, 2080, 2080
  loop 2090(0x0), 2090(0x0)
  loop 2100, 2100, 2100
  loop 2110(0x0), 2110(0x0)
  loop 2120, 2120, 2120
  loop 2130(0x0), 2130(0x0)
  loop 2140, 2140, 2140
  loop 2150(0x0), 2150(0x0)
  loop 2160, 2160, 2160
  loop 2170(0x0), 2170(0x0)
  loop 2180, 2180, 2180
  loop 2190(0x0), 2190(0x0)
  loop 2200, 2200, 2200
  loop 2210(0x0), 2210(0x0)
  loop 2220, 2220, 2220
  loop 2230(0x0), 2230(0x0)
  loop 2240, 2240, 2240
  loop 2250(0x0), 2250(0x0)
  loop 2260, 2260, 2260
  loop 2270(0x0), 2270(0x0)
  loop 2280, 2280, 2280
  loop 2290(0x0), 2290(0x0)
  loop 2300, 2300, 2300
  loop 2310(0x0), 2310(0x0)
  loop 2320, 2320, 2320
  loop 2330(0x0), 2330(0x0)
  loop 2340, 2340, 2340
  loop 2350(0x0), 2350(0x0)
  loop 2360, 2360, 2360
  loop 2370(0x0), 2370(0x0)
  loop 2380, 2380, 2380
  loop 2390(0x0), 2390(0x0)
  loop 2400, 2400, 2400
  loop 2410(0x0), 2410(0x0)
  loop 2420, 2420, 2420
  loop 2430(0x0), 2430(0x0)
  loop 2440, 2440, 2440
  loop 2450(0x0), 2450(0x0)
  loop 2460, 2460, 2460
  loop 2470(0x0), 2470(0x0)
  loop 2480, 2480, 2480
  loop 2490(0x0), 2490(0x0)
  loop 2500, 2500, 2500
  loop 2510(0x0), 2510(0x0)
  loop 2520, 2520, 2520
  loop 2530(0x0), 2530(0x0)
  loop 2540, 2540, 2540
  loop 2550(0x0), 2550(0x0)
  loop 2560, 2560, 2560
  loop 2570(0x0), 2570(0x0)
  loop 2580, 2580, 2580
  loop 2590(0x0), 2590(0x0)
  loop 2600, 2600, 2600
  loop 2610(0x0), 2610(0x0)
  loop 2620, 2620, 2620
  loop 2630(0x0), 2630(0x0)
  loop 2640, 2640, 2640
  loop 2650(0x0), 2650(0x0)
  loop 2660, 2660, 2660
  loop 2670(0x0), 2670(0x0)
  loop 2680, 2680, 2680
  loop 2690(0x0), 2690(0x0)
  loop 2700, 2700, 2700
  loop 2710(0x0), 2710(0x0)
  loop 2720, 2720, 2720
  loop 2730(0x0), 2730(0x0)
  loop 2740, 2740, 2740
  loop 2750(0x0), 2750(0x0)
  loop 2760, 2760, 2760
  loop 2770(0x0), 2770(0x0)
  loop 2780, 2780, 2780
  loop 2790(0x0), 2790(0x0)
  loop 2800, 2800, 2800
  loop 2810(0x0), 2810(0x0)
  loop 2820, 2820, 2820
  loop 2830(0x0), 2830(0x0)
  loop 2840, 2840, 2840
  loop 2850(0x0), 2850(0x0)
  loop 2860, 2860, 2860
  loop 2870(0x0), 2870(0x0)
  loop 2880, 2880, 2880
  loop 2890(0x0), 2890(0x0)
  loop 2900, 2900, 2900
  loop 2910(0x0), 2910(0x0)
  loop 2920, 2920, 2920
  loop 2930(0x0), 2930(0x0)
  loop 2940, 2940, 2940
  loop 2950(0x0), 2950(0x0)
  loop 2960, 2960, 2960
  loop 2970(0x0), 2970(0x0)
  loop 2980, 2980, 2980
  loop 2990(0x0), 2990(0x0)
  loop 3000, 3000, 3000
  loop 3010(0x0), 3010(0x0)
  loop 3020, 3020, 3020
  loop 3030(0x0), 3030(0x0)
  loop 3040, 3040, 3040
  loop 3050(0x0), 3050(0x0)
  loop 3060, 3060, 3060
  loop 3070(0x0), 3070(0x0)
  loop 3080, 3080, 3080
  loop 3090(0x0), 3090(0x0)
  loop 3100, 3100, 3100
  loop 3110(0x0), 3110(0x0)
  loop 3120, 3120, 3120
  loop 3130(0x0), 3130(0x0)
  loop 3140, 3140, 3140
  loop 3150(0x0), 3150(0x0)
  loop 3160, 3160, 3160
  loop 3170(0x0), 3170(0x0)
  loop 3180, 3180, 3180
  loop 3190(0x0), 3190(0x0)
  loop 3200, 3200, 3200
  loop 3210(0x0), 3210(0x0)
  loop 3220, 3220, 3220
  loop 3230(0x0), 3230(0x0)
  loop 3240, 3240, 3240
  loop 3250(0x0), 3250(0x0)
  loop 3260, 3260, 3260
  loop 3270(0x0), 3270(0x0)
  loop 3280, 3280, 3280
  loop 3290(0x0), 3290(0x0)
  loop 3300, 3300, 3300
  loop 3310(0x0), 3310(0x0)
  loop 3320, 3320, 3320
  loop 3330(0x0), 3330(0x0)
  loop 3340, 3340, 3340
  loop 3350(0x0), 3350(0x0)
  loop 3360, 3360, 3360
  loop 3370(0x0), 3370(0x0)
  loop 3380, 3380, 3380
  loop 3390(0x0), 3390(0x0)
  loop 3400, 3400, 3400
  loop 3410(0x0), 3410(0x0)
  loop 3420, 3420, 3420
  loop 3430(0x0), 3430(0x0)
  loop 3440, 3440, 3440
  loop 3450(0x0), 3450(0x0)
  loop 3460, 3460, 3460
  loop 3470(0x0), 3470(0x0)
  loop 3480, 3480, 3480
  loop 3490(0x0), 3490(0x0)
  loop 3500, 3500, 3500
  loop 3510(0x0), 3510(0x0)
  loop 3520, 3520, 3520
  loop 3530(0x0), 3530(0x0)
  loop 3540, 3540, 3540
  loop 3550(0x0), 3550(0x0)
  loop 3560, 3560, 3560
  loop 3570(0x0), 3570(0x0)
  loop 3580, 3580, 3580
  loop 3590(0x0), 3590(0x0)
  loop 3600, 3600, 3600
  loop 3610(0x0), 3610(0x0)
  loop 3620, 3620, 3620
  loop 3630(0x0), 3630(0x0)
  loop 3640, 3640, 3640
  loop 3650(0x0), 3650(0x0)
  loop 3660, 3660, 3660
  loop 3670(0x0), 3670(0x0)
  loop 3680, 3680, 3680
  loop 3690(0x0), 3690(0x0)
  loop 3700, 3700, 3700
  loop 3710(0x0), 3710(0x0)
  loop 3720, 3720, 3720
  loop 3730(0x0), 3730(0x0)
  loop 3740, 3740, 3740
  loop 3750(0x0), 3750(0x0)
  loop 3760, 3760, 3760
  loop 3770(0x0), 3770(0x0)
  loop 3780, 3780, 3780
  loop 3790(0x0), 3790(0x0)
  loop 3800, 3800, 3800
  loop 3810(0x0), 3810(0x0)
  loop 3820, 3820, 3820
  loop 3830(0x0), 3830(0x0)
  loop 3840, 3840, 3840
  loop 3850(0x0), 3850(0x0)
  loop 3860, 3860, 3860
  loop 3870(0x0), 3870(0x0)
  loop 3880, 3880, 3880
  loop 3890(0x0), 3890(0x0)
  loop 3900, 3900, 3900
  loop 3910(0x0), 3910(0x0)
  loop 3920, 3920, 3920
  loop 3930(0x0), 3930(0x0)
  loop 3940, 3940, 3940
  loop 3950(0x0), 3950(0x0)
  loop 3960, 3960, 3960
  loop 3970(0x0), 3970(0x0)
  loop 3980, 3980, 3980
  loop 3990(0x0), 3990(0x0)
  loop 4000, 4000, 4000
  loop 4010(0x0), 4010(0x0)
  loop 4020, 4020, 4020
  loop 4030(0x0), 4030(0x0)
  loop 4040, 4040, 4040
  loop 4050(0x0), 4050(0x0)
  loop 4060, 4060, 4060
  loop 4070(0x0), 4070(0x0)
  loop 4080, 4080, 4080
  loop 4090(0x0), 4090(0x0)
  loop 4100, 4100, 4100
  loop 4110(0x0), 4110(0x0)
  loop 4120, 4120, 4120
  loop 4130(0x0), 4130(0x0)
  loop 4140, 4140, 4140
  loop 4150(0x0), 4150(0x0)
  loop 4160, 4160, 4160
  loop 4170(0x0), 4170(0x0)
  loop 4180, 4180, 4180
  loop 4190(0x0), 4190(0x0)
  loop 4200, 4200, 4200
  loop 4210(0x0), 4210(0x0)
  loop 4220, 4220, 4220
  loop 4230(0x0), 4230(0x0)
  loop 4240, 4240, 4240
  loop 4250(0x0), 4250(0x0)
  loop 4260, 4260, 4260
  loop 4270(0x0), 4270(0x0)
  loop 4280, 4280, 4280
  loop 4290(0x0), 4290(0x0)
  loop 4300, 4300, 4300
  loop 4310(0x0), 4310(0x0)
  loop 4320, 4320, 4320
  loop 4330(0x0), 4330(0x0)
  loop 4340, 4340, 4340
  loop 4350(0x0), 4350(0x0)
  loop 4360, 4360, 4360
  loop 4370(0x0), 4370(0x0)
  loop 4380, 4380, 4380
  loop 4390(0x0), 4390(0x0)
  loop 4400, 4400, 4400
  loop 4410(0x0), 4410(0x0)
  loop 4420, 4420, 4420
  loop 4430(0x0), 4430(0x0)
  loop 4440, 4440, 4440
  loop 4450(0x0), 4450(0x0)
  loop 4460, 4460, 4460
  loop 4470(0x0), 4470(0x0)
  loop 4480, 4480, 4480
  loop 4490(0x0), 4490(0x0)
  loop 4500, 4500, 4500
  loop 4510(0x0), 4510(0x0)
  loop 4520, 4520, 4520
  loop 4530(0x0), 4530(0x0)
  loop 4540, 4540, 4540
  loop 4550(0x0), 4550(0x0)
  loop 4560, 4560, 4560
  loop 4570(0x0), 4570(0x0)
  loop 4580, 4580, 4580
  loop 4590(0x0), 4590(0x0)
  loop 4600, 4600, 4600
  loop 4610(0x0), 4610(0x0)
  loop 4620, 4620, 4620
  loop 4630(0x0), 4630(0x0)
  loop 4640, 4640, 4640
  loop 4650(0x0), 4650(0x0)
  loop 4660, 4660, 4660
  loop 4670(0x0), 4670(0x0)
  loop 4680, 4680, 4680
  loop 4690(0x0), 4690(0x0)
  loop 4700, 4700, 4700
  loop 4710(0x0), 4710(0x0)
  loop 4720, 4720, 4720
  loop 4730(0x0), 4730(0x0)
  loop 4740, 4740, 4740
  loop 4750(0x0), 4750(0x0)
  loop 4760, 4760, 4760
  loop 4770(0x0), 4770(0x0)
  loop 4780, 4780, 4780
  loop 4790(0x0), 4790(0x0)
  loop 4800, 4800, 4800
  loop 4810(0x0), 4810(0x0)
  loop 4820, 4820, 4820
  loop 4830(0x0), 4830(0x0)
  loop 4840, 4840, 4840
  loop 4850(0x0), 4850(0x0)
  loop 4860, 4860, 4860
  loop 4870(0x0), 4870(0x0)
  loop 4880, 4880, 4880
  loop 4890(0x0), 4890(0x0)
  loop 4900, 4900, 4900
  loop 4910(0x0), 4910(0x0)
  loop 4920, 4920, 4920
  loop 4930(0x0), 4930(0x0)
  loop 4940, 4940, 4940
  loop 4950(0x0), 4950(0x0)
  loop 4960, 4960, 4960
  loop 4970(0x0), 4970(0x0)
  loop 4980, 4980, 4980
  loop 4990(0x0), 4990(0x0)
  loop 5000, 5000, 5000
  loop 5010(0x0), 5010(0x0)
  loop 5020, 5020, 5020
  loop 5030(0x0), 5030(0x0)
  loop 5040, 5040, 5040
  loop 5050(0x0), 5050(0x0)
  loop 5060, 5060, 5060
  loop 5070(0x0), 5070(0x0)
  loop 5080, 5080, 5080
  loop 5090(0x0), 5090(0x0)
  loop 5100, 5100, 5100
  loop 5110(0x0), 5110(0x0)
  loop 5120, 5120, 5120
  loop 5130(0x0), 5130(0x0)
  loop 5140, 5140, 5140
  loop 5150(0x0), 5150(0x0)
  loop 5160, 5160, 5160
  loop 5170(0x0), 5170(0x0)
  loop 5180, 5180, 5180
  loop 5190(0x0), 5190(0x0)
  loop 5200, 5200, 5200
  loop 5210(0x0), 5210(0x0)
  loop 5220, 5220, 5220
  loop 5230(0x0), 5230(0x0)
  loop 5240, 5240, 5240
  loop 5250(0x0), 5250(0x0)
  loop 5260, 5260, 5260
  loop 5270(0x0), 5270(0x0)
  loop 5280, 5280, 5280
  loop 5290(0x0), 5290(0x0)
  loop 5300, 5300, 5300
  loop 5310(0x0), 5310(0x0)
  loop 5320, 5320, 5320
  loop 5330(0x0), 5330(0x0)
  loop 5340, 5340, 5340
  loop 5350(0x0), 5350(0x0)
  loop 5360, 5360, 5360
  loop 5370(0x0), 5370(0x0)
  loop 5380, 5380, 5380
  loop 5390(0x0), 5390(0x0)
  loop 5400, 5400, 5400
  loop 5410(0x0), 5410(0x0)
  loop 5420, 5420, 5420
  loop 5430(0x0), 5430(0x0)
  loop 5440, 5440, 5440
  loop 5450(0x0), 5450(0x0)
  loop 5460, 5460, 5460
  loop 5470(0x0), 5470(0x0)
  loop 5480, 5480, 5480
  loop 5490(0x0), 5490(0x0)
  loop 5500, 5500, 5500
  loop 5510(0x0), 5510(0x0)
  loop 5520, 5520, 5520
  loop 5530(0x0), 5530(0x0)
  loop 5540, 5540, 5540
  loop 5550(0x0), 5550(0x0)
  loop 5560, 5560, 5560
  loop 5570(0x0), 5570(0x0)
  loop 5580, 5580, 5580
  loop 5590(0x0), 5590(0x0)
 
```

Semantic preservation



```
Lustre

node WATCHDOG (set, reset: bool; delay: int)
  returns (alarm: bool);
var remaining_delay: int; deadline: bool;
set
  alarm = WATCHDOG(set, reset, deadline);
  deadline = EDGE(remaining_delay = 0);
  remaining_delay = 0 or set then delay else
    (0 -> pre(remaining_delay) - 1);
set

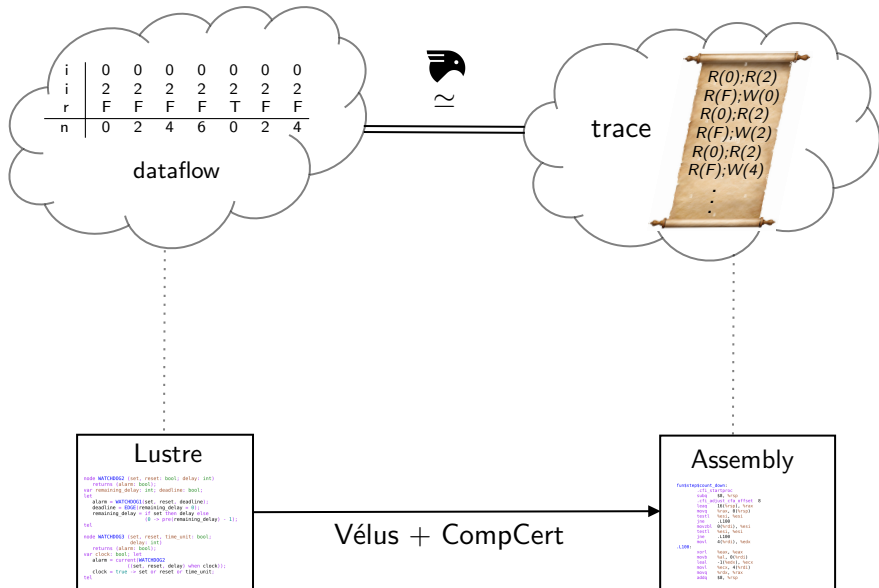
node WATCHDOG2 (set, reset, time_unit: bool;
  delay: int)
  returns (alarm: bool);
var clock: bool; len
  alarm = current(WATCHDOG2
    (set, reset, delay) when clock);
  clock = true -> set or reset or time_unit;
set
```

Vélus + CompCert

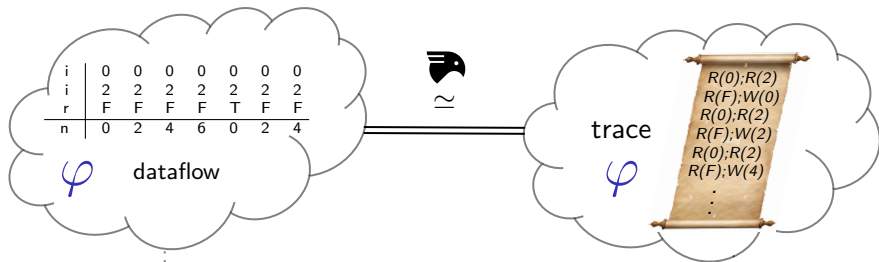
```
Assembly

funct@compcert_#mem
  cfi_startproc
  push 32, %rbp
  cfi_def_cfa_offset 8
  loop 14(0x14), loop
  movq %rax, %rax
  movl 1100, %eax
  cmovle %eax, %eax
  movl 1100, %eax
  movl 40(0x40), %eax
  jmp 1100
  movl 40(0x40), %eax
  movl %eax, %eax
  movl 1100(0x1100), %eax
  movl %eax, %eax
  movl %eax, %eax
  movl 11, %eax
  jmp 1100
```

Semantic preservation



Semantic preservation



Goal: interactive program verification

Lustre

```

node WATCHDOG (set, reset: bool; delay: int)
  returns (alarm: bool);
var remaining_delay: int; deadline: bool;
set
  alarm = WATCHDOG(set, reset, deadline);
  deadline = EDGE(remaining_delay = 0);
  remaining_delay = 0 && set then delay else
    (0 -> pre(remaining_delay) - 1);
end

node WATCHDOG2 (set, reset, time_unit: bool;
  delay: int)
  returns (alarm: bool);
var clock: bool; top
  alarm = current(WATCHDOG2
    (set, reset, delay) when clock);
  clock = true && set or reset or time_unit;
end
  
```

Vélus + CompCert

Assembly

```

FunctionExport_Atoms
  @CPU, @Hardware
  subq $8, %rdi
  loop 1f(1)rdi, %rax
  movq %rax, %rsi
  movl $1000, %ecx
  cmovq @CPU, %rsi
  movl %rsi, %edi
  jmp 1f(1)
1:
  movl $0, %eax
  movl %rsi, %eax
  movl -1000(%rsi, %ecx)
  movl %eax, %rsi
  movl %rsi, %rsi
  jmp 1f, %rax
1:
  
```

Dataflow semantics, in Rocq

```
node rer (i : bool; n : int) returns (o : bool)
var e, c : bool; d : int when c;
let
  e = i and (false fby (not i));
  c = e or (false fby o);
  d = countdown((n, e) when c);
  o = actdef(c, d) > 0;
tel;
```

i	F	T	T	T	F	T	T	...
n	2	2	2	2	2	2	2	...
e	F	T	F	F	F	T	F	...
c	F	T	T	T	F	T	T	...
d		2	1	0		2	1	...
o	F	T	T	F	F	T	T	...

Dataflow semantics, in Rocq

```
node rer (i : bool; n : int) returns (o : bool)
var e, c : bool; d : int when c;
let
  e = i and (false fby (not i));
  c = e or (false fby o);
  d = countdown((n, e) when c);
  o = actdef(c, d) > 0;
tel;
```

i	F	T	T	T	F	T	T	...
n	2	2	2	2	2	2	2	...
e	F	T	F	F	F	T	F	...
c	F	T	T	T	F	T	T	...
d	A	2	1	0	A	2	1	...
o	F	T	T	F	F	T	T	...

Dataflow semantics, in Rocq

```
node rer (i : bool; n : int) returns (o : bool)
var e, c : bool; d : int when c;
let
  e = i and (false fby (not i));
  c = e or (false fby o);
  d = countdown((n, e) when c);
  o = actdef(c, d) > 0;
tel;
```

i	F	T	T	T	F	T	T	...
n	2	2	2	2	2	2	2	...
e	F	T	F	F	F	T	F	...
c	F	T	T	T	F	T	T	...
d	A	2	1	0	A	2	1	...
o	F	T	T	F	F	T	T	...

► CoInductive Str (D:Type) := Cons : D -> Str D -> Str D

Dataflow semantics, in Rocq

```
node rer (i : bool; n : int) returns (o : bool)
var e, c : bool; d : int when c;
let
  e = i and (false fby (not i));
  c = e or (false fby o);
  d = countdown((n, e) when c);
  o = actdef(c, d) > 0;
tel;
```

i	F	T	T	T	F	T	T	...
n	2	2	2	2	2	2	2	...
e	F	T	F	F	F	T	F	...
c	F	T	T	T	F	T	T	...
d	A	2	1	0	A	2	1	...
o	F	T	T	F	F	T	T	...

- ▶ CoInductive Str (D:Type) := Cons : D -> Str D -> Str D
- ▶ History := PM.t (Str svalue) *1 variable, 1 stream*

Dataflow semantics, in Rocq

```
node rer (i : bool; n : int) returns (o : bool)
var e, c : bool; d : int when c;
let
  e = i and (false fby (not i));
  c = e or (false fby o);
  d = countdown((n, e) when c);
  o = actdef(c, d) > 0;
tel;
```

i	F	T	T	T	F	T	T	...
n	2	2	2	2	2	2	2	...
e	F	T	F	F	F	T	F	...
c	F	T	T	T	F	T	T	...
d	A	2	1	0	A	2	1	...
o	F	T	T	F	F	T	T	...

- ▶ CoInductive Str (D:Type) := Cons : D -> Str D -> Str D
- ▶ History := PM.t (Str svalue) *1 variable, 1 stream*
- ▶ Relational semantics (H : History) + constraints on H

Dataflow semantics, in Rocq

```
node rer (i : bool; n : int) returns (o : bool)
var e, c : bool; d : int when c;
let
  e = i and (false fby (not i));
  c = e or (false fby o);
  d = countdown((n, e) when c);
  o = actdef(c, d) > 0;
tel;
```

i	F	T	T	T	F	T	T	...
n	2	2	2	2	2	2	2	...
e	F	T	F	F	F	T	F	...
c	F	T	T	T	F	T	T	...
d	A	2	1	0	A	2	1	...
o	F	T	T	F	F	T	T	...

- ▶ CoInductive Str (D:Type) := Cons : D -> Str D -> Str D
- ▶ History := PM.t (Str svalue) *1 variable, 1 stream*
- ▶ Relational semantics (H : History) + constraints on H

$$\begin{aligned} H(i) &\equiv s_i \wedge H(n) \equiv s_n \\ \wedge \exists s_e, H(e) &\equiv s_e \wedge H \vdash i \text{ and } (\text{false fby } (\text{not } i)) \Downarrow s_e \\ \wedge \exists s_c, H(c) &\equiv s_c \wedge H \vdash e \text{ or } (\text{false fby } o) \Downarrow s_c \\ \wedge \dots \end{aligned}$$

Dataflow se

node rer (i : bo
var e, c : bool;
let
e = i and (fal
c = e or (false
d = countdow
o = actdef(c,
tel;

- ▶ CoInduc
- ▶ History
- ▶ Relation

```
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
End sem_branch.

Inductive sem_exp : history → Stream B → exp → list (Stream svalue) → P :=
| Sconst:
  ∀ H b c cs,
  cs = const b c →
  sem_exp H b (Econst c) [cs]
| Senum:
  ∀ H b k ty es,
  es = enum b k →
  sem_exp H b (Eenum k ty) [es]
| Svar:
  ∀ H b x s ann,
  sem_var H (Var x) s →
  sem_exp H b (Evar x ann) [s]
| Sunop:
  ∀ H b e op ty s o ann,
  sem_exp H b e [s] →
  typeof e = [ty] →
  lift1 op ty s o →
  sem_exp H b (Eunop op e ann) [o]
| Sbinop:
  ∀ H b e1 e2 op ty1 ty2 s1 s2 o ann,
  sem_exp H b e1 [s1] →
  sem_exp H b e2 [s2] →
  typeof e1 = [ty1] →
  typeof e2 = [ty2] →
  lift2 op ty1 ty2 s1 s2 o →
  sem_exp H b (Ebinop op e1 e2 ann) [o]
| Sfby:
  ∀ H b e0s es anns s0ss sss os,
  Forall2 (sem_exp H b) e0s s0ss →
  Forall2 (sem_exp H b) es sss →
  Forall3 fby (concat s0ss) (concat sss) os →
  sem_exp H b (Efby e0s es anns) os
| Swhen:
  ∀ H b x tx s k es lann ss os,
  Forall2 (sem_exp H b) es ss →
  sem_var H (Var x) s →
  Forall2 (λ s' → when k s' s) (concat ss) os →
  sem_exp H b (Ewhen es (x, tx) k lann) os
| Smerge:
  ∀ H b x tx s es lann vs os,
  sem_var H (Var x) s →
  Forall2Brs (sem_exp H b) es vs →
  Forall2 (merge s) vs os →
  sem_exp H b (EMerge (x, tx) es lann) os
```

F	T	T	...
2	2	2	...
<hr/>			
F	T	F	...
F	T	T	...
A	2	1	...
F	T	T	...

▶ Str D

eam

H

Se

Dataflow se

node rer (i : bo
var e, c : bool;
let
e = i and (fal
c = e or (false
d = countdow
o = actdef(c,
tel;

- ▶ CoInduc
- ▶ History
- ▶ Relation

```
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
End sem_branch.

Inductive sem_exp : history → Stream B → exp → list (Stream sval) → P :=
| Sconst:
  ∀ H b c cs,
  cs = const b c →
  sem_exp H b (Econst c) [cs]
| Senum:
  ∀ H b k ty es,
  es = enum b k →
  sem_exp H b (Eenum k ty) [es]
| Svar:
  ∀ H b x s ann,
  sem_var H (Var x) s →
  sem_exp H b (Evar x ann) [s]
| Sunop:
  ∀ H b e op ty s o ann,
  sem_exp H b e [s] →
  typeof e = [ty] →
  lift1 op ty s o →
  sem_exp H b (Eunop op e ann) [o]
| Sbinop:
  ∀ H b e1 e2 op ty1 ty2 s1 s2 o ann,
  sem_exp H b e1 [s1] →
  sem_exp H b e2 [s2] →
  typeof e1 = [ty1] →
  typeof e2 = [ty2] →
  lift2 op ty1 ty2 s1 s2 o →
  sem_exp H b (Ebinop op e1 e2 ann) [o]
| Sfby:
  ∀ H b e0s es anns s0ss sss os,
  Forall2 (sem_exp H b) e0s s0ss →
  Forall2 (sem_exp H b) es sss →
  Forall3 fby (concat s0ss) (concat sss) os →
  sem_exp H b (Efby e0s es anns) os
| Swhen:
  ∀ H b x tx s k es lann ss os,
  Forall2 (sem_exp H b) es ss →
  sem_var H (Var x) s →
  Forall2 (λ s' → when k s' s) (concat ss) os →
  sem_exp H b (Ewhen es (x, tx) k lann) os
| Smerge:
  ∀ H b x tx s es lann vs os,
  sem_var H (Var x) s →
  Forall2Brs (sem_exp H b) es vs →
  Forall2 (merge s) vs os →
  sem_exp H b (EMerge (x, tx) es lann) os
```

CompCert/cfrontend/Cop.v

F	T	T	...
2	2	2	...
<hr/>			
F	T	F	...
F	T	T	...
A	2	1	...
F	T	T	...

Str D

eam

H

Se

Relational synchronous semantics

Pros

- ▶ No explicit fixpoint computation
- ▶ Easy to add constraints
- ▶ No errors/undefined behaviors
- ▶ Useful A invariants

Relational synchronous semantics

Pros

- ▶ No explicit fixpoint computation
- ▶ Easy to add constraints
- ▶ No errors/undefined behaviors
- ▶ Useful A invariants

Hard for verification

Relational synchronous semantics

Pros

- ▶ No
- ▶ Easy
- ▶ No
- ▶ Use

```
xs, ys, s1, s2, s3 : Stream svalue
H : History
bs := base_of [xs]
H1 : Env.find _r H = Some xs
H2 : Env.find _up H = Some s1
H3 : Env.find _pn H = Some s3
H4 : Env.find _n H = Some ys
H5 : fby (const bs (Venum 1)) s2 s1
H6 : sem_exp H bs ((_up and _n < 5) or (not _up and _n ≤ 1)) s2
H7 : sem_exp H bs (0 fby _n) s3
H8 : sem_exp H bs (if _up then (_pn + 1) else (_pn - 1)) ys
=====
Forall_Str (λ v ⇒ match v with
| present (Vint i) ⇒
    Int.lt Int.zero i && Int.lt i (Int.repr 6) = true
| _ ⇒ ⊥
end) ys.
```

Hard for

Relational synchronous semantics

Pros

- ▶ No explicit fixpoint computation
- ▶ Easy to add constraints
- ▶ No errors/undefined behaviors
- ▶ Useful A invariants

Hard for verification

- ▶ Readability problem, no equational reasoning

Relational synchronous semantics

Pros

- ▶ No explicit fixpoint computation
- ▶ Easy to add constraints
- ▶ No errors/undefined behaviors
- ▶ Useful A invariants

Hard for verification

- ▶ Readability problem, no equational reasoning
- ▶ Hard to deal with A

Relational synchronous semantics

Pros

- ▶ No explicit fixpoint computation
- ▶ Easy to add constraints
- ▶ No errors/undefined behaviors
- ▶ Useful A invariants

Hard for verification

- ▶ Readability problem, no equational reasoning
- ▶ Hard to deal with A
- ▶ No errors/undefined behaviors (partial operators)

x		8	4	2	0	4	...
$o=8/x$		1	2	4	?		...

Relational synchronous semantics

Pros

- ▶ No explicit fixpoint computation
- ▶ Easy to add constraints
- ▶ No errors/undefined behaviors
- ▶ Useful A invariants

Hard for verification

- ▶ Readability problem, no equational reasoning
- ▶ Hard to deal with A
- ▶ No errors/undefined behaviors (partial operators)
- ▶ Existence of the semantics?

x		8	4	2	0	4	...
$o=8/x$		1	2	4	?		...

Kahn denotational semantics

$$\begin{aligned}i^\# &= i.i^\# \\op^\#(s_1, s_2) &= \epsilon \text{ if } s_1 = \epsilon \text{ or } s_2 = \epsilon \\op^\#(v_1.s_1, v_2.s_2) &= (v_1 \text{ op } v_2).op^\#(s_1, s_2) \\fby^\#(\epsilon, ys) &= \epsilon \\fby^\#(v.xs, ys) &= v.ys \\when^\#(s_1, s_2) &= \epsilon \text{ if } s_1 = \epsilon \text{ or } s_2 = \epsilon \\when^\#(v_1.s_1, true.s_2) &= v_1.(when^\#(s_1, s_2)) \\when^\#(v_1.s_1, false.s_2) &= when^\#(s_1, s_2) \\merge^\#(s_1, s_2, s_3) &= \epsilon \text{ if } s_1 = \epsilon \text{ or } s_2 = \epsilon \text{ or } s_3 = \epsilon \\merge^\#(true.s_1, v_2.s_2, s_3) &= v_2.merge^\#(s_1, s_2, s_3) \\merge^\#(false.s_1, s_2, v_3.s_3) &= v_3.merge^\#(s_1, s_2, s_3)\end{aligned}$$

Fig. 1. Data-flow Kahn semantics

- ▶ Set of streams $D^\omega = D^* \cup D^\infty$
- ▶ CPO with prefix order ($\perp = \epsilon$, the empty stream)
- ▶ Primitive stream functions are continuous
- ▶ Solution of a system: fixpoint of the equations

Kahn denotational semantics

$$\begin{aligned}i^\# &= i.i^\# \\op^\#(s_1, s_2) &= \epsilon \text{ if } s_1 = \epsilon \text{ or } s_2 = \epsilon \\op^\#(v_1.s_1, v_2.s_2) &= (v_1 \text{ op } v_2).op^\#(s_1, s_2) \\fby^\#(\epsilon, ys) &= \epsilon \\fby^\#(v.xs, ys) &= v.ys \\when^\#(s_1, s_2) &= \epsilon \text{ if } s_1 = \epsilon \text{ or } s_2 = \epsilon \\when^\#(v_1.s_1, true.s_2) &= v_1.(when^\#(s_1, s_2)) \\when^\#(v_1.s_1, false.s_2) &= when^\#(s_1, s_2) \\merge^\#(s_1, s_2, s_3) &= \epsilon \text{ if } s_1 = \epsilon \text{ or } s_2 = \epsilon \text{ or } s_3 = \epsilon \\merge^\#(true.s_1, v_2.s_2, s_3) &= v_2.merge^\#(s_1, s_2, s_3) \\merge^\#(false.s_1, s_2, v_3.s_3) &= v_3.merge^\#(s_1, s_2, s_3)\end{aligned}$$

Fig. 1. Data-flow Kahn semantics

- ▶ Set of streams $D^\omega = D^* \cup D^\infty$
- ▶ CPO with prefix order ($\perp = \epsilon$, the empty stream)
- ▶ Primitive stream functions are continuous
- ▶ Solution of a system: fixpoint of the equations

Kahn denotational semantics

$$\begin{aligned}i^\# &= i.i^\# \\op^\#(s_1, s_2) &= \epsilon \text{ if } s_1 = \epsilon \text{ or } s_2 = \epsilon \\op^\#(v_1.s_1, v_2.s_2) &= (v_1 \text{ op } v_2).op^\#(s_1, s_2) \\fby^\#(\epsilon, ys) &= \epsilon \\fby^\#(v.xs, ys) &= v.ys \\when^\#(s_1, s_2) &= \epsilon \text{ if } s_1 = \epsilon \text{ or } s_2 = \epsilon \\when^\#(v_1.s_1, true.s_2) &= v_1.(when^\#(s_1, s_2)) \\when^\#(v_1.s_1, false.s_2) &= when^\#(s_1, s_2) \\merge^\#(s_1, s_2, s_3) &= \epsilon \text{ if } s_1 = \epsilon \text{ or } s_2 = \epsilon \text{ or } s_3 = \epsilon \\merge^\#(true.s_1, v_2.s_2, s_3) &= v_2.merge^\#(s_1, s_2, s_3) \\merge^\#(false.s_1, s_2, v_3.s_3) &= v_3.merge^\#(s_1, s_2, s_3)\end{aligned}$$

Fig. 1. Data-flow Kahn semantics

- ▶ Set of streams $D^\omega = D^* \cup D^\infty$
- ▶ CPO with prefix order ($\perp = \epsilon$, the empty stream)
- ▶ Primitive stream functions are continuous
- ▶ Solution of a system: fixpoint of the equations

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

Scott domains, $(\omega\text{-})\text{CPO}$, fixpoints . . .

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

Scott domains, $(\omega\text{-})\text{CPO}$, fixpoints ...

```
(** ** Ordered type *)
Record ord : Type := mk_ord
  { tord :> Type;
    Ole : tord → tord →  $\mathbb{P}$ ;
    Ole_refl :  $\forall x : \text{tord}, \text{Ole } x \ x$ ;
    Ole_trans :  $\forall x \ y \ z : \text{tord}, \text{Ole } x \ y \rightarrow \text{Ole } y \ z \rightarrow \text{Ole } x \ z$  }.

Infix "≤" := Ole : 0_scope.

Definition monotonic (O1 O2 : ord) (f : O1 → O2) :=
   $\forall x \ y, x \leq y \rightarrow f \ x \leq f \ y$ .

Record fmono (O1 O2 : ord) : Type := mk_fmono {
  fmonot :> O1 → O2;
  fmonotonic: monotonic fmonot
}.
Infix "-m>" := fmono.

(** ** Definition of cpos *)
Record cpo : Type := mk_cpo {
  tcpo:>ord;
  D0 : tcpo;
  lub: (nat0 -m> tcpo) → tcpo;
  Dbot :  $\forall x : \text{tcpo}, D_0 \leq x$ ;
  le_lub :  $\forall (f : \text{nat0} \text{-m>} \text{tcpo}) (n : \mathbb{N}), f \ n \leq \text{lub } f$ ;
  lub_le :  $\forall (f : \text{nat0} \text{-m>} \text{tcpo}) (x : \text{tcpo}), (\forall n, f \ n \leq x) \rightarrow \text{lub } f \leq x$ 
}.
```

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

Scott domains, $(\omega\text{-})\text{CPO}$, fixpoints ...

```
(** ** Ordered type *)
Record ord : Type := mk_ord
  { tord :> Type;
    Ole : tord → tord →  $\mathbb{P}$ ;
    Ole_refl :  $\forall x : \text{tord}, \text{Ole } x \ x$ ;
    Ole_trans :  $\forall x \ y \ z : \text{tord}, \text{Ole } x \ y \rightarrow \text{Ole } y \ z \rightarrow \text{Ole } x \ z$  }.

Infix " $\leq$ " := Ole : 0_scope.

Definition monotonic (O1 O2 : ord) (f : O1 → O2) :=
   $\forall x \ y, x \leq y \rightarrow f \ x \leq f \ y$ .

Record fmono (O1 O2 : ord) : Type := mk_fmono {
  fmonot :> O1 → O2;
  fmonotonic: monotonic fmonot
}.
Infix "-m>" := fmono.

(** ** Definition of cpos *)
Record cpo : Type := mk_cpo {
  tcpo:>ord;
  D0 : tcpo;
  lub: (nat0 -m> tcpo) → tcpo;
  Dbot :  $\forall x : \text{tcpo}, D_0 \leq x$ ;
  le_lub :  $\forall (f : \text{nat0} \text{-m>} \text{tcpo}) (n : \mathbb{N}), f \ n \leq \text{lub } f$ ;
  lub_le :  $\forall (f : \text{nat0} \text{-m>} \text{tcpo}) (x : \text{tcpo}), (\forall n, f \ n \leq x) \rightarrow \text{lub } f \leq x$ 
}.
```

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$$\text{fixp} : (D \rightarrow_c D) \rightarrow_c D$$
$$\text{fixp } F \simeq_D F(\text{fixp } F)$$
$$\forall F P, \text{ admissible } P \rightarrow P \perp \rightarrow (\forall x, P x \rightarrow P(F x)) \rightarrow P(\text{fixp } F)$$

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$$\text{fixp} : (D \rightarrow_c D) \rightarrow_c D$$
$$\text{fixp } F \simeq_D F(\text{fixp } F)$$
$$\forall F P, \text{ admissible } P \rightarrow P \perp \rightarrow (\forall x, P x \rightarrow P(F x)) \rightarrow P(\text{fixp } F)$$

Building D^ω

```
CoInductive Str (D : Type) :=
```

```
  | Cons : D -> Str D -> Str D
```

```
  | Tau : Str D -> Str D.
```

```
CoFixpoint bot := Tau bot.
```

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$$\text{fixp} : (D \rightarrow_c D) \rightarrow_c D$$
$$\text{fixp } F \simeq_D F(\text{fixp } F)$$
$$\forall F P, \text{ admissible } P \rightarrow P \perp \rightarrow (\forall x, P x \rightarrow P(F x)) \rightarrow P(\text{fixp } F)$$

Building D^ω

CoInductive Str (D : Type) :=

| Cons : D -> Str D -> Str D

| Tau : Str D -> Str D.

$$\perp \stackrel{\text{def}}{=} \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdots$$

CoFixpoint bot := Tau bot.

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$$\text{fixp} : (D \rightarrow_c D) \rightarrow_c D$$
$$\text{fixp } F \simeq_D F(\text{fixp } F)$$
$$\forall F P, \text{ admissible } P \rightarrow P \perp \rightarrow (\forall x, P x \rightarrow P(F x)) \rightarrow P(\text{fixp } F)$$

Building D^ω

CoInductive Str (D : Type) :=

| Cons : D -> Str D -> Str D

| Tau : Str D -> Str D.

$$\perp \stackrel{\text{def}}{=} \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdots$$

CoFixpoint bot := Tau bot.

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$$\text{fixp} : (D \rightarrow_c D) \rightarrow_c D$$
$$\text{fixp } F \simeq_D F(\text{fixp } F)$$
$$\forall F P, \text{ admissible } P \rightarrow P \perp \rightarrow (\forall x, P x \rightarrow P(F x)) \rightarrow P(\text{fixp } F)$$

Building D^ω

```
CoInductive Str (D : Type) :=  
  | Cons : D -> Str D -> Str D  
  | Tau : Str D -> Str D.
```

$$\perp \stackrel{\text{def}}{=} \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdots$$
$$[a; b] \stackrel{\text{def}}{=} a \cdot b \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdots$$

```
CoFixpoint bot := Tau bot.
```

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$$\text{fixp} : (D \rightarrow_c D) \rightarrow_c D$$
$$\text{fixp } F \simeq_D F(\text{fixp } F)$$
$$\forall F P, \text{ admissible } P \rightarrow P \perp \rightarrow (\forall x, P x \rightarrow P(F x)) \rightarrow P(\text{fixp } F)$$

Building D^ω

CoInductive Str (D : Type) :=

| Cons : D -> Str D -> Str D

| Tau : Str D -> Str D.

CoFixpoint bot := Tau bot.

$$\perp \stackrel{\text{def}}{=} \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdots$$
$$[a; b] \stackrel{\text{def}}{=} a \cdot b \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdots$$
$$\simeq \tau \cdot \tau \cdot a \cdot \tau \cdot b \cdot \tau \cdot \tau \cdots$$

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$$\text{fixp} : (D \rightarrow_c D) \rightarrow_c D$$

$$\text{fixp } F \simeq_D F(\text{fixp } F)$$

$$\forall F P, \text{ admissible } P \rightarrow P \perp \rightarrow (\forall x, P x \rightarrow P(F x)) \rightarrow P(\text{fixp } F)$$

Building D^ω

CoInductive Str (D : Type) :=

| Cons : D -> Str D -> Str D

| Tau : Str D -> Str D.

CoFixpoint bot := Tau bot.

$$\perp \stackrel{\text{def}}{=} \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdots$$

$$[a; b] \stackrel{\text{def}}{=} a \cdot b \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdots$$

$$\simeq \tau \cdot \tau \cdot a \cdot \tau \cdot b \cdot \tau \cdot \tau \cdots$$

$$\preceq a \cdot b \cdot c \cdot \tau \cdot \tau \cdot \tau \cdot \tau \cdots$$

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$\text{filter} : (A \rightarrow \mathbb{B}) \rightarrow A^\omega \rightarrow_c A^\omega$

$\text{filter } f (a \cdot s) \simeq a \cdot \text{filter } f s \quad \text{si } (f a) = \text{T}$

$\text{filter } f (a \cdot s) \simeq \text{filter } f s \quad \text{si } (f a) = \text{F}$

$\text{filter } f \perp \simeq \perp$

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$\text{filter} : (A \rightarrow \mathbb{B}) \rightarrow A^\omega \rightarrow_c A^\omega$

$\text{filter } f (a \cdot s) \simeq a \cdot \text{filter } f s \quad \text{si } (f a) = \text{T}$

$\text{filter } f (a \cdot s) \simeq \text{filter } f s \quad \text{si } (f a) = \text{F}$

$\text{filter } f \perp \simeq \perp$

$\text{map} : (A \rightarrow B) \rightarrow A^\omega \rightarrow_c B^\omega$

$\text{map } f (a \cdot s) \simeq (f a) \cdot \text{map } f s$

$\text{map } f \perp \simeq \perp$

Kahn denotational semantics, in Rocq

C. Paulin-Mohring, *A denotational semantics for Kahn networks in Coq*, 2009

$\text{filter} : (A \rightarrow \mathbb{B}) \rightarrow A^\omega \rightarrow_c A^\omega$

$\text{filter } f (a \cdot s) \simeq a \cdot \text{filter } f s \quad \text{si } (f a) = \text{T}$

$\text{filter } f (a \cdot s) \simeq \text{filter } f s \quad \text{si } (f a) = \text{F}$

$\text{filter } f \perp \simeq \perp$

$\text{map} : (A \rightarrow B) \rightarrow A^\omega \rightarrow_c B^\omega$

$\text{map } f (a \cdot s) \simeq (f a) \cdot \text{map } f s$

$\text{map } f \perp \simeq \perp$

$\text{zip} : (A \rightarrow B \rightarrow C) \rightarrow A^\omega \rightarrow_c B^\omega \rightarrow_c C^\omega$

$\text{zip } f (a \cdot s_1) (b \cdot s_2) \simeq (f a b) \cdot \text{zip } f s_1 s_2$

$\text{zip } f s \perp \simeq \text{zip } f \perp s \simeq \perp$

Kahn denotat

C. Paulin-Mohring, A

oq, 2009

$\text{filter} : (A \rightarrow \mathbb{B})$

$\text{filter } f (a \cdot s) \simeq$

$\text{filter } f (a \cdot s) \simeq$

$\text{filter } f \perp \simeq \perp$

$\text{map} : (A \rightarrow B)$

$\text{map } f (a \cdot s) \simeq$

$\text{map } f \perp \simeq \perp$

$\text{zip} : (A \rightarrow B \rightarrow$

$\text{zip } f (a \cdot s_1) (b$

$\text{zip } f s \perp \simeq \text{zip}$

```
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
(** ** Mapping a function on a stream *)

Section MapStream.
Variable D D' : Type.
Variable F : D -> D'.

Definition mapf : (DS D -C-> DS D') -m> D -0-> DS D -C-> DS D'.
exists (lambda (f : DS D -C-> DS D') (a:D) => CONS (F a) @_ f).
red; intros f g Hle a.
apply (fcont_le_intro (D1:=DS D) (D2:=DS D')); intro s.
repeat rewrite fcont_comp_simpl.
auto.
Defined.

Lemma mapf_simpl : forall f, mapf f = lambda a => CONS (F a) @_ f.
trivial.
Qed.

Definition Mapf : (DS D -C-> DS D') -c> D-0-> DS D -C->DS D'.
exists mapf.
red; intros h a.
rewrite mapf_simpl.
rewrite fcpo_lub_simpl.
apply (fcont_le_intro (D1:=DS D) (D2:=DS D')); intro s.
rewrite fcont_lub_simpl.
repeat rewrite fcont_comp_simpl; repeat rewrite fcont_lub_simpl; intros.
rewrite (fcontinuous (CONS (F a)) (h <_> s)).
apply lub_le_compat; intro n; auto.
Defined.

Definition MAP : DS D -C-> DS D' := FIXP (DS D-C->DS D') (DSCASE D D' @_ Mapf).

Lemma MAP_eq : MAP == DSCASE D D' (Mapf MAP).
exact (FIXP_eq (DSCASE D D' @_ Mapf)).
Qed.

Definition map (s : DS D) := MAP s.

Lemma map_bot : map 0 == 0.
unfold map.
rewrite (fcont_eq_elim MAP_eq 0).
rewrite DSCASE_simpl; auto.
Qed.

Lemma map_eq_cons : forall a s,
  map (cons a s) == cons (F a) (map s).
intros; unfold map at 1.
rewrite (fcont_eq_elim MAP_eq (cons a s)).
rewrite DSCASE_simpl.
rewrite DSCASE_cons; auto.
Qed.
```

Kahn denotat

C. Paulin-Mohring, A

oq, 2009

filter : $(A \rightarrow \mathbb{B})$

filter f $(a \cdot s) \simeq$

filter f $(a \cdot s) \simeq$

filter $f \perp \simeq \perp$

map : $(A \rightarrow B)$

map f $(a \cdot s) \simeq$

map $f \perp \simeq \perp$

zip : $(A \rightarrow B \rightarrow$

zip f $(a \cdot s_1)$ $(b$

zip f $s \perp \simeq$ zip

```
File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
(** ** Mapping a function on a stream **)

Section MapStream.
Variable D D' : Type.
Variable F : D -> D'.

Definition mapf : (DS D -C-> DS D') -m> D -0-> DS D -C-> DS D'.
exists (lambda (f : DS D -C-> DS D') (a:D) => CONS (F a) @_ f).
red; intros f g Hle a.
apply (fcont_le_intro (D1:=DS D) (D2:=DS D')); intro s.
repeat rewrite fcont_comp_simpl.
auto.
Defined.

Lemma mapf_simpl : forall f, mapf f = lambda a => CONS (F a) @_ f.
trivial.
Qed.

Definition Mapf : (DS D -C-> DS D') -c> D-0-> DS D -C->DS D'.
exists mapf.
red; intros h a.
rewrite mapf_simpl.
rewrite fcpo_lub_simpl.
apply (fcont_le_intro (D1:=DS D) (D2:=DS D')); intro s.
rewrite fcont_lub_simpl.
repeat rewrite fcont_comp_simpl; repeat rewrite fcont_lub_simpl; intros.
rewrite (fcontinuous (CONS (F a)) (h <_> s)).
apply lub_le_compat; intro n; auto.
Defined.

Definition MAP : DS D -C-> DS D' := FIXP (DS D-C->DS D') (DSCASE D D' @_ Mapf).

Lemma MAP_eq : MAP == DSCASE D D' (Mapf MAP).
exact (FIXP_eq (DSCASE D D' @_ Mapf)).
Qed.

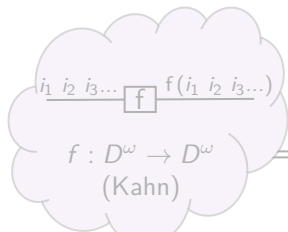
Definition map (s : DS D) := MAP s.

Lemma map_bot : map 0 == 0.
unfold map.
rewrite (fcont_eq_elim MAP_eq 0).
rewrite DSCASE_simpl; auto.
Qed.

Lemma map_eq_cons : forall a s,
  map (cons a s) == cons (F a) (map s).
intros; unfold map at 1.
rewrite (fcont_eq_elim MAP_eq (cons a s)).
rewrite DSCASE_simpl.
rewrite DSCASE_cons; auto.
Qed.
```

could be more practical...

Roadmap



deep
embedding

```

Lustre

code METACOMB (set, reset, head, delay: int)
  returns (alarm, bool)
var remaining_delay: int; deadline: bool;
set
alarm = METACOMB(set, reset, deadline);
deadline = (set && remaining_delay = 0);
remaining_delay = if set then delay else
  remaining_delay - 1;
end

code METACOMB (set, reset, time_slot: bool,
  delay: int)
  returns (alarm, bool)
var clock: bool; slot
alarm = METACOMB(set, reset, time_slot);
clock = time_slot && remaining_delay <= delay;
end
  
```

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show

```

Inductive sem_exp
  : history -> Stream B -> exp -> list (Stream svalue) -> P :=
| Sconst:
  V H b c cs,
  cs = const b c ->
  sem_exp H b (Econst c) [cs]
| Senum:
  V H b k ty es,
  es = enum b k ->
  sem_exp H b (Eenum k ty) [es]
| Svar:
  V H b x s ann,
  sem_var H (Var x) s ->
  sem_exp H b (Evar x ann) [s]
| Slast:
  V H b x s ann,
  sem_var H (Last x) s ->
  sem_exp H b (Elast x ann) [s]
| Sunop:
  V H b e op ty s o ann,
  sem_exp H b e [s] ->
  typeof e = [ty] ->
  lift1 op ty s o ->
  sem_exp H b (Eunop op e ann) [o]
| Sbinop:
  V H b e1 e2 op ty1 ty2 s1 s2 o ann,
  sem_exp H b e1 [s1] ->
  sem_exp H b e2 [s2] ->
  typeof e1 = [ty1] ->
  typeof e2 = [ty2] ->
  lift2 op ty1 ty2 s1 s2 o ->
  sem_exp H b (Ebinop op e1 e2 ann) [o]
| Sextcall:
  V H b f es tyout ck tyins ss vs,
  Forall2 (lambda ty cty -> ty = Tprimitive cty) (typesof es) tyins ->
  Forall (sem_exp H b) es ss ->
  
```



Roadmap

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help

```
Definition denot_exp (ins : list ident) (e : exp) :  
  FEnv -C- SEnv -C- SEnv -C- nprod (numstreams e).
```

Lemma denot_exp_eq :

```
  ∀ ins e envG envI env,  
  denot_exp ins e envG envI env =  
  match e return nprod (numstreams e) with  
  | Econst c _ => sconst (Vscalar (sem_cconst c) (bss ins envI))  
  | Eenum c _ => sconst (Venum c) (bss ins envI)  
  | Evar x _ => denot_var ins envI env x  
  | Eunop op e an =>  
    let se := denot_exp ins e envG envI env in  
    match numstreams e as n return nprod n → nprod 1 with  
    | 1 => λ se =>  
      match typeof e with  
      | [ty] => sunop (λ v => sem_unop op v ty) se  
      | _ => errTy  
      end  
    | _ => λ _ => errTy  
    end se  
  | Ebinop op e1 e2 an =>  
    let se1 := denot_exp ins e1 envG envI env in  
    let se2 := denot_exp ins e2 envG envI env in  
    match numstreams e1 as n1, numstreams e2 as n2  
    return nprod n1 → nprod n2 → nprod 1 with  
    | 1,1 => λ se1 se2 =>  
      match typeof e1, typeof e2 with  
      | [ty1],[ty2] => sbinop (λ v1 v2 => sem_binop op v1 ty1 v2 ty2) se1  
      | _,_ => errTy  
      end  
    | _,_ => λ _ _ => errTy  
    end se1 se2  
  | Efby e0s es an =>  
    let s0s := denot_exps ins e0s envG envI env in  
    let ss := denot_exps ins es envG envI env in  
    let n := (list_sum (List.map numstreams e0s)) in
```

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show

Inductive sem_exp

: history → Stream B → exp → list (Stream svalue) → P :=

```
| Sconst:  
  ∀ H b c cs,  
  cs = const b c →  
  sem_exp H b (Econst c) [cs]  
| Senum:  
  ∀ H b k ty es,  
  es = enum b k →  
  sem_exp H b (Eenum k ty) [es]  
| Svar:  
  ∀ H b x s ann,  
  sem_var H (Var x) s →  
  sem_exp H b (Evar x ann) [s]  
| Slast:  
  ∀ H b x s ann,  
  sem_var H (Last x) s →  
  sem_exp H b (Elast x ann) [s]  
| Sunop:  
  ∀ H b e op ty s o ann,  
  sem_exp H b e [s] →  
  typeof e = [ty] →  
  lift1 op ty s o →  
  sem_exp H b (Eunop op e ann) [o]  
| Sbinop:  
  ∀ H b e1 e2 op ty1 ty2 s1 s2 o ann,  
  sem_exp H b e1 [s1] →  
  sem_exp H b e2 [s2] →  
  typeof e1 = [ty1] →  
  typeof e2 = [ty2] →  
  lift2 op ty1 ty2 s1 s2 o →  
  sem_exp H b (Ebinop op e1 e2 ann) [o]  
| Sextcall:  
  ∀ H b f es tyout ck tyins ss vs, █  
  Forall2 (λ ty cty → ty = Tprimitive cty) (typesof es) tyins →  
  Forall1 (sem_exp H b) es ss →
```

Roadmap

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help

```
Definition denot_exp (ins : list ident) (e : exp) :
  FEnv -C- SEnv -C- SEnv -C- nprod (numstreams e).

Lemma denot_exp_eq :
  ∀ ins e envG envI env,
  denot_exp ins e envG envI env =
  match e return nprod (numstreams e) with
  | Econst c _ => sconst (Vscalar (sem_cconst c)) (bss ins envI)
  | Eenum c _ => sconst (Venum c) (bss ins envI)
  | Evar x _ => denot_var ins envI env x
  | Eunop op e an =>
    let se := denot_exp ins e envG envI env in
    match numstreams e as n return nprod n → nprod 1 with
    | 1 => λ se =>
      match typeof e with
      | [ty] => sunop (λ v => sem_unop op v ty) se
      | _ => errTy
      end
    | _ => λ _ => errTy
    end se
  | Ebinop op e1 e2 an =>
    let se1 := denot_exp ins e1 envG envI env in
    let se2 := denot_exp ins e2 envG envI env in
    match numstreams e1 as n1, numstreams e2 as n2
    return nprod n1 → nprod n2 → nprod 1 with
    | 1,1 => λ se1 se2 =>
      match typeof e1, typeof e2 with
      | [ty1],[ty2] => sbinop (λ v1 v2 => sem_binop op v1 ty1 v2 ty2) se1
      | _ , _ => errTy
      end
    | _ , _ => λ _ _ => errTy
    end se1 se2
  | Efbv e0s es an =>
    let s0s := denot_exps ins e0s envG envI env in
    let ss := denot_exps ins es envG envI env in
    let n := (list_sum (List.map numstreams e0s)) in
```

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show

```
Inductive sem_exp
  : history → Stream B → exp → list (Stream svalue) → P :=
| Sconst:
  ∀ H b c cs,
  cs = const b c →
  sem_exp H b (Econst c) [cs]
| Senum:
```

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet

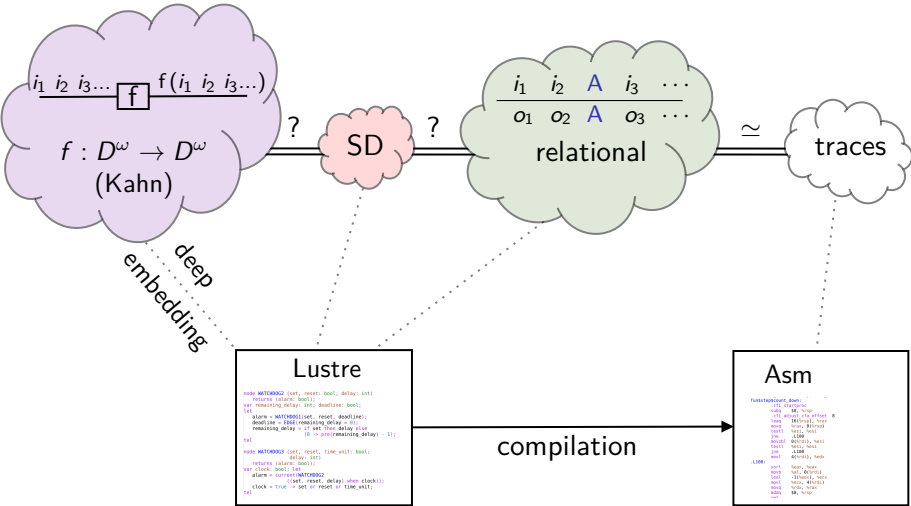
```
with sem_block: history → Stream B → block → P :=
| Sbeq:
  ∀ H b eq,
  sem_equation H b eq →
  sem_block H b (Beq eq)

with sem_node: ident → list (Stream svalue) → list (Stream svalue) → P :=
| Snode:
  ∀ f ss os n H,
  find_node f G = Some n →
  Forall2 (λ x → sem_var H (Var x)) (List.map fst n.(n_in)) ss →
  Forall2 (λ x → sem_var H (Var x)) (List.map fst n.(n_out)) os →
  let bs := clocks_of ss in
  sem_block H bs n.(n_block) →
  sem_node f ss os.

sem_exp H b e2 [s2] →
typeof e1 = [ty1] →
typeof e2 = [ty2] →
lift2 op ty1 ty2 s1 s2 0 →
sem_exp H b (Ebinop op e1 e2 ann) [o]

| Sextcall:
  ∀ H b f es tyout ck tyins ss vs,
  Forall2 (λ ty cty → ty = Tprimitive cty) (typesof es) tyins →
  Forall (sem_exn H b) es es →
```


Roadmap



Three semantic models

	Kahn	Synchronous dénot.	Relational
streams			
mechanics			

Three semantic models

	Kahn	Synchronous dénot.	Relational
streams			D^∞ $D := A \mid v$
mechanics			

Three semantic models

	Kahn	Synchronous dénot.	Relational
streams	$D^\omega = D^* \cup D^\infty$ $D := v \mid err$		D^∞ $D := A \mid v$
mechanics			

Three semantic models

	Kahn	Synchronous dénot.	Relational
streams	$D^\omega = D^* \cup D^\infty$ $D := v \mid err$	$D^\omega = D^* \cup D^\infty$ $D := A \mid v \mid err$	D^∞ $D := A \mid v$
mechanics			

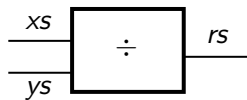
Three semantic models

	Kahn	Synchronous dénot.	Relational
streams	$D^\omega = D^* \cup D^\infty$ $D := v \mid err$	$D^\omega = D^* \cup D^\infty$ $D := A \mid v \mid err$	D^∞ $D := A \mid v$
mechanics			relations/predicates conjunctions existence supposed

Three semantic models

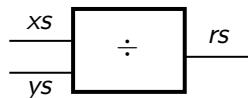
	Kahn	Synchronous dénot.	Relational
streams	$D^\omega = D^* \cup D^\infty$ $D := v \mid err$	$D^\omega = D^* \cup D^\infty$ $D := A \mid v \mid err$	D^∞ $D := A \mid v$
mechanics	denot, prefix order, $\perp = \epsilon$ continuous and blocking functions fixpoint computation		relations/predicates conjunctions existence supposed

Combinatorial operator



xs	A	10	10	A	10	10	...
ys	A	10	5	A	2	1	...
rs	A	1	2	A	5	10	...

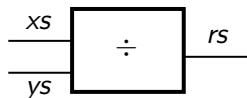
Combinatorial operator



<i>xs</i>	A	10	10	A	10	10	...
<i>ys</i>	A	10	5	A	2	1	...
<i>rs</i>	A	1	2	A	5	10	...

$$\frac{\text{LIFT } xs \ ys \ rs}{\text{LIFT } (A \cdot xs) \ (A \cdot ys) \ (A \cdot rs)}$$
$$\frac{\text{LIFT } xs \ ys \ rs \quad v_1 \div v_2 = \text{Some } r}{\text{LIFT } (v_1 \cdot xs) \ (v_2 \cdot ys) \ (r \cdot rs)}$$

Combinatorial operator

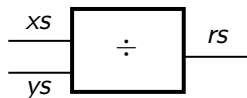


<i>xs</i>	A	10	10	A	10	10	...
<i>ys</i>	A	10	5	A	2	1	...
<i>rs</i>	A	1	2	A	5	10	...

lift : Str \rightarrow_c Str \rightarrow_c Str :=
zip

$$\frac{\text{LIFT } xs \ ys \ rs}{\text{LIFT } (A \cdot xs) \ (A \cdot ys) \ (A \cdot rs)}$$
$$\frac{\text{LIFT } xs \ ys \ rs \quad v_1 \div v_2 = \text{Some } r}{\text{LIFT } (v_1 \cdot xs) \ (v_2 \cdot ys) \ (r \cdot rs)}$$

Combinatorial operator



<i>xs</i>	A	10	10	A	10	10	...
<i>ys</i>	A	10	5	A	2	1	...
<i>rs</i>	A	1	2	A	5	10	...

`lift : Str →c Str →c Str :=`

`zip (λ A, A → A`

)

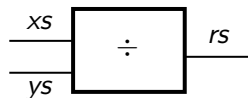
LIFT *xs ys rs*

LIFT (*A · xs*) (*A · ys*) (*A · rs*)

LIFT *xs ys rs* $v_1 \div v_2 = \text{Some } r$

LIFT ($v_1 \cdot xs$) ($v_2 \cdot ys$) ($r \cdot rs$)

Combinatorial operator

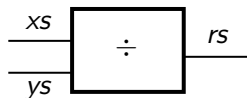


<i>xs</i>	A	10	10	A	10	10	...
<i>ys</i>	A	10	5	A	2	1	...
<i>rs</i>	A	1	2	A	5	10	...

```
lift : Str →c Str →c Str :=  
zip (λ A, A → A  
    | v1, v2 → (match v1 ÷ v2 with  
                  | Some r → r  
                  | None → errrt)  
    )
```

$$\frac{\text{LIFT } xs \ ys \ rs}{\text{LIFT } (A \cdot xs) \ (A \cdot ys) \ (A \cdot rs)}$$
$$\frac{\text{LIFT } xs \ ys \ rs \quad v_1 \div v_2 = \text{Some } r}{\text{LIFT } (v_1 \cdot xs) \ (v_2 \cdot ys) \ (r \cdot rs)}$$

Combinatorial operator



<i>xs</i>	A	10	10	A	10	10	...
<i>ys</i>	A	10	5	A	2	1	...
<i>rs</i>	A	1	2	A	5	10	...

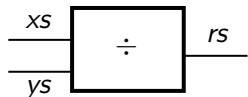
```

lift : Str →c Str →c Str :=
zip (λ A, A → A
    | v1, v2 → (match v1 ÷ v2 with
                  | Some r → r
                  | None → errrt)
    | v, A | A, v → errsync
)
  
```

$$\frac{\text{LIFT } xs \ ys \ rs}{\text{LIFT } (A \cdot xs) \ (A \cdot ys) \ (A \cdot rs)}$$

$$\frac{\text{LIFT } xs \ ys \ rs \quad v_1 \div v_2 = \text{Some } r}{\text{LIFT } (v_1 \cdot xs) \ (v_2 \cdot ys) \ (r \cdot rs)}$$

Combinatorial operator

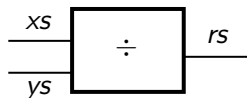


<i>xs</i>	A	10	10	A	10	10	...
<i>ys</i>	A	10	5	A	2	1	...
<i>rs</i>	A	1	2	A	5	10	...

```
lift : Str →c Str →c Str :=  
zip (λ A, A → A  
    | v1, v2 → (match v1 ÷ v2 with  
                    | Some r → r  
                    | None → errrt)  
    | v, A | A, v → errsync  
    | err, _ | _, err → err)
```

$$\frac{\text{LIFT } xs \ ys \ rs}{\text{LIFT } (A \cdot xs) \ (A \cdot ys) \ (A \cdot rs)}$$
$$\frac{\text{LIFT } xs \ ys \ rs \quad v_1 \div v_2 = \text{Some } r}{\text{LIFT } (v_1 \cdot xs) \ (v_2 \cdot ys) \ (r \cdot rs)}$$

Combinatorial operator




<i>xs</i>	A	10	10	A	10	10	...
<i>ys</i>	A	10	5	A	2	1	...
<i>rs</i>	A	1	2	A	5	10	...

$\text{lift} : \text{Str} \rightarrow_c \text{Str} \rightarrow_c \text{Str} :=$

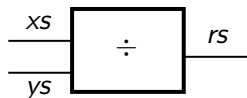
$\text{zip}(\lambda A, A \rightarrow A$
 $\quad | v_1, v_2 \rightarrow (\text{match } v_1 \div v_2 \text{ with}$
 $\quad \quad | \text{Some } r \rightarrow r$
 $\quad \quad | \text{None} \rightarrow \text{err}_{\text{rt}})$
 $\quad | v, A \mid A, v \rightarrow \text{err}_{\text{sync}}$
 $\quad | \text{err}, _ \mid _, \text{err} \rightarrow \text{err})$

$$\frac{\text{LIFT } xs \ ys \ rs}{\text{LIFT } (A \cdot xs) \ (A \cdot ys) \ (A \cdot rs)}$$

$$\frac{\text{LIFT } xs \ ys \ rs \quad v_1 \div v_2 = \text{Some } r}{\text{LIFT } (v_1 \cdot xs) \ (v_2 \cdot ys) \ (r \cdot rs)}$$

 **Theorem:** if $\text{lift } xs \ ys$ is error-free, then $\text{LIFT } xs \ ys \ (\text{lift } xs \ ys)$

Combinatorial operator



xs	A	10	10	A	10	10	...
ys	A	10	5	A	2	1	...
rs	A	1	2	A	5	10	...

$\text{lift}^\# : \text{Str} \rightarrow_c \text{Str} \rightarrow_c \text{Str} :=$

$\text{zip}(\lambda A, A \rightarrow A$

| $v_1, v_2 \rightarrow (\text{match } v_1 \div v_2 \text{ with}$
| $\text{Some } r \rightarrow r$
| $\text{None} \rightarrow \text{err}_{\text{rt}}$)

| $v, A \mid A, v \rightarrow \text{err}_{\text{sync}}$

| $\text{err}, _ \mid _, \text{err} \rightarrow \text{err}$)

$\text{LIFT } xs \text{ } ys \text{ } rs$

$\text{LIFT } (A \cdot xs) \text{ } (A \cdot ys) \text{ } (A \cdot rs)$

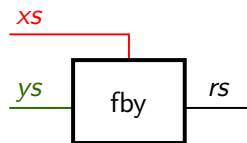
$\text{LIFT } xs \text{ } ys \text{ } rs \quad v_1 \div v_2 = \text{Some } r$

$\text{LIFT } (v_1 \cdot xs) \text{ } (v_2 \cdot ys) \text{ } (r \cdot rs)$

Theorem: if $\text{lift } xs \text{ } ys$ is error-free, then $\text{LIFT } xs \text{ } ys \text{ } (\text{lift } xs \text{ } ys)$

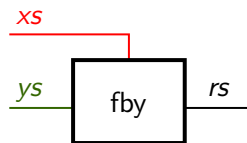
Theorem: also, $[\text{lift } xs \text{ } ys]_A \simeq \text{lift}^\# [xs]_A [ys]_A$

Initialized delay



<i>xs</i>	5	5	5	5	5	...
<i>ys</i>	6	7	8	9	10	...
<i>rs</i>	5	6	7	8	9	...

Initialized delay



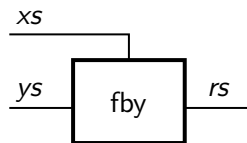
<i>xs</i>	5	5	5	5	5	...
<i>ys</i>	6	7	8	9	10	...
<i>rs</i>	5	6	7	8	9	...

$\text{fby}^\# : \text{Str} \rightarrow_c \text{Str} \rightarrow_c \text{Str}$

$\text{fby}^\# \perp ys \simeq \perp$

$\text{fby}^\# (v \cdot xs) ys \simeq v \cdot ys$

Initialized delay



<i>xs</i>	A	5	A	5	5	5	5	...
<i>ys</i>	A	6	A	7	8	9	10	...
<i>rs</i>	A	5	A	6	7	8	9	...

$\text{fby}^\# : \text{Str} \rightarrow_c \text{Str} \rightarrow_c \text{Str}$

$\text{fby}^\# \perp \text{ys} \simeq \perp$

$\text{fby}^\# (v \cdot \text{xs}) \text{ys} \simeq v \cdot \text{ys}$

$\text{FBY } \text{xs } \text{ys } \text{rs}$

$\text{FBY } (A \cdot \text{xs}) (A \cdot \text{ys}) (A \cdot \text{rs})$

$\text{FBY}_1 v_2 \text{xs } \text{ys } \text{rs}$

$\text{FBY } (v_1 \cdot \text{xs}) (v_2 \cdot \text{ys}) (v_1 \cdot \text{rs})$

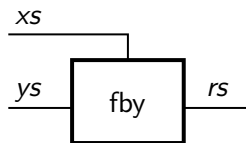
$\text{FBY}_1 v \text{xs } \text{ys } \text{rs}$

$\text{FBY}_1 v (A \cdot \text{xs}) (A \cdot \text{ys}) (A \cdot \text{rs})$

$\text{FBY}_1 v_2 \text{xs } \text{ys } \text{rs}$

$\text{FBY}_1 v (v_1 \cdot \text{xs}) (v_2 \cdot \text{ys}) (v \cdot \text{rs})$

Initialized delay



xs	A	5	A	5	5	5	5	...
ys	A	6	A	7	8	9	10	...
rs	A	5	A	6	7	8	9	...

$\text{fby}^\# : \text{Str} \rightarrow_c \text{Str} \rightarrow_c \text{Str}$

$\text{fby}^\# \perp ys \simeq \perp$

$\text{fby}^\# (v \cdot xs) ys \simeq v \cdot ys$

- ▶ sync errors?
- ▶ $x = 0$ $\text{fby} (x + 1)$?

$\text{FBY } xs \ ys \ rs$

$\text{FBY} (A \cdot xs) (A \cdot ys) (A \cdot rs)$

$\text{FBY}_1 \ v_2 \ xs \ ys \ rs$

$\text{FBY} (v_1 \cdot xs) (v_2 \cdot ys) (v_1 \cdot rs)$

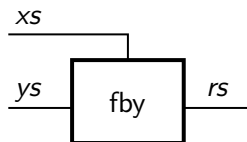
$\text{FBY}_1 \ v \ xs \ ys \ rs$

$\text{FBY}_1 \ v (A \cdot xs) (A \cdot ys) (A \cdot rs)$

$\text{FBY}_1 \ v_2 \ xs \ ys \ rs$

$\text{FBY}_1 \ v (v_1 \cdot xs) (v_2 \cdot ys) (v \cdot rs)$

Initialized delay



<i>xs</i>	A	5	A	5	5	5	5	...
<i>ys</i>	A	6	A	7	8	9	10	...
<i>rs</i>	A	5	A	6	7	8	9	...

$\text{fby } (A \cdot xs) \text{ } ys := A \cdot \text{fby}_A \text{ } xs \text{ } ys$

$\text{fby } (v \cdot xs) \text{ } ys := v \cdot \text{fby}'_1 \text{ } \text{None } xs \text{ } ys$

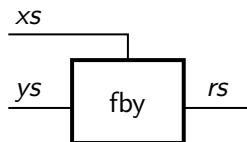
$\text{fby}_A \text{ } xs (A \cdot ys) := \text{fby } xs \text{ } ys$

$\text{fby}'_1 \text{ } \text{None } xs (v \cdot ys) := \text{fby}_1 \text{ } v \text{ } xs \text{ } ys$

$\text{fby}_1 \text{ } v (A \cdot xs) \text{ } ys := A \cdot \text{fby}'_1 \text{ } (\text{Some } v) \text{ } xs \text{ } ys$

$\text{fby}_1 \text{ } v (v_x \cdot xs) \text{ } ys := v \cdot \text{fby}'_1 \text{ } \text{None } xs \text{ } ys$

Initialized delay



xs	A	5	A	5	5	5	5	...
ys	A	6	A	7	8	9	10	...
rs	A	5	A	6	7	8	9	...

$$\text{fby } (A \cdot xs) \text{ } ys := A \cdot \text{fby}_A \text{ } xs \text{ } ys$$

$$\text{fby } (v \cdot xs) \text{ } ys := v \cdot \text{fby}'_1 \text{ None } xs \text{ } ys$$

$$\text{fby } (\text{err} \cdot xs) \text{ } ys := \text{err} \cdot \text{map } (\lambda x. \text{err}) \text{ } xs$$

$$\text{fby}_A \text{ } xs \text{ } (A \cdot ys) := \text{fby } xs \text{ } ys$$

$$\text{fby}_A \text{ } xs \text{ } (\text{err} \cdot ys) := \text{map } (\lambda x. \text{err}) \text{ } xs$$

$$\text{fby}_A \text{ } xs \text{ } (v \cdot ys) := \text{map } (\lambda x. \text{err}_{\text{sync}}) \text{ } xs$$

$$\text{fby}'_1 \text{ None } xs \text{ } (v \cdot ys) := \text{fby}_1 \text{ } v \text{ } xs \text{ } ys$$

$$\text{fby}'_1 \text{ (Some } v) \text{ } xs \text{ } (A \cdot ys) := \text{fby}_1 \text{ } v \text{ } xs \text{ } ys$$

$$\text{fby}'_1 \text{ } _ \text{ } xs \text{ } (\text{err} \cdot ys) := \text{map } (\lambda x. \text{err}) \text{ } xs$$

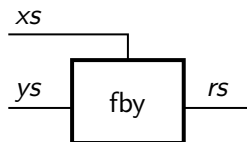
$$\text{fby}'_1 \text{ } _ \text{ } xs \text{ } (_ \cdot ys) := \text{map } (\lambda x. \text{err}_{\text{sync}}) \text{ } xs$$

$$\text{fby}_1 \text{ } v \text{ } (A \cdot xs) \text{ } ys := A \cdot \text{fby}'_1 \text{ (Some } v) \text{ } xs \text{ } ys$$

$$\text{fby}_1 \text{ } v \text{ } (v_x \cdot xs) \text{ } ys := v \cdot \text{fby}'_1 \text{ None } xs \text{ } ys$$

$$\text{fby}_1 \text{ } v \text{ } (\text{err} \cdot xs) \text{ } ys := \text{err} \cdot \text{map } (\lambda x. \text{err}) \text{ } xs$$

Initialized delay



xs	A	5	A	5	5	5	5	...
ys	A	6	A	7	8	9	10	...
rs	A	5	A	6	7	8	9	...

$$\text{fby } (A \cdot xs) \text{ } ys := A \cdot \text{fby}_A \text{ } xs \text{ } ys$$

$$\text{fby } (v \cdot xs) \text{ } ys := v \cdot \text{fby}'_1 \text{ None } xs \text{ } ys$$

$$\text{fby } (\text{err} \cdot xs) \text{ } ys := \text{err} \cdot \text{map } (\lambda x. \text{err}) \text{ } xs$$

$$\text{fby}_A \text{ } xs (A \cdot ys) := \text{fby } xs \text{ } ys$$

$$\text{fby}_A \text{ } xs (\text{err} \cdot ys) := \text{map } (\lambda x. \text{err}) \text{ } xs$$

$$\text{fby}_A \text{ } xs (v \cdot ys) := \text{map } (\lambda x. \text{err}_{\text{sync}}) \text{ } xs$$

$$\text{fby}'_1 \text{ None } xs (v \cdot ys) := \text{fby}_1 \text{ } v \text{ } xs \text{ } ys$$

$$\text{fby}'_1 (\text{Some } v) \text{ } xs (A \cdot ys) := \text{fby}_1 \text{ } v \text{ } xs \text{ } ys$$


$$\text{fby}'_1 \text{ } _ \text{ } xs (\text{err} \cdot ys) := \text{map } (\lambda x. \text{err}) \text{ } xs$$

$$\text{fby}'_1 \text{ } _ \text{ } xs (_ \cdot ys) := \text{map } (\lambda x. \text{err}_{\text{sync}}) \text{ } xs$$

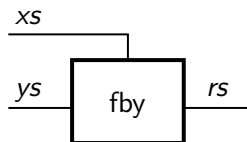
$$\text{fby}_1 \text{ } v (A \cdot xs) \text{ } ys := A \cdot \text{fby}'_1 (\text{Some } v) \text{ } xs \text{ } ys$$

$$\text{fby}_1 \text{ } v (v_x \cdot xs) \text{ } ys := v \cdot \text{fby}'_1 \text{ None } xs \text{ } ys$$

$$\text{fby}_1 \text{ } v (\text{err} \cdot xs) \text{ } ys := \text{err} \cdot \text{map } (\lambda x. \text{err}) \text{ } xs$$

 **Theorem:** if $\text{fby } xs \text{ } ys$ is error-free, then $\text{FBY } xs \text{ } ys \text{ } (\text{fby } xs \text{ } ys)$

Initialized delay



xs	A	5	A	5	5	5	5	...
ys	A	6	A	7	8	9	10	...
rs	A	5	A	6	7	8	9	...

$$\text{fby } (A \cdot xs) \text{ } ys := A \cdot \text{fby}_A \text{ } xs \text{ } ys$$

$$\text{fby } (v \cdot xs) \text{ } ys := v \cdot \text{fby}'_1 \text{ None } xs \text{ } ys$$

$$\text{fby } (\text{err} \cdot xs) \text{ } ys := \text{err} \cdot \text{map } (\lambda x. \text{err}) \text{ } xs$$

$$\text{fby}_A \text{ } xs \text{ } (A \cdot ys) := \text{fby } xs \text{ } ys$$

$$\text{fby}_A \text{ } xs \text{ } (\text{err} \cdot ys) := \text{map } (\lambda x. \text{err}) \text{ } xs$$

$$\text{fby}_A \text{ } xs \text{ } (v \cdot ys) := \text{map } (\lambda x. \text{err}_{\text{sync}}) \text{ } xs$$

$$\text{fby}'_1 \text{ None } xs \text{ } (v \cdot ys) := \text{fby}_1 \text{ } v \text{ } xs \text{ } ys$$

$$\text{fby}'_1 \text{ (Some } v) \text{ } xs \text{ } (A \cdot ys) := \text{fby}_1 \text{ } v \text{ } xs \text{ } ys$$


$$\text{fby}'_1 \text{ } _ \text{ } xs \text{ } (\text{err} \cdot ys) := \text{map } (\lambda x. \text{err}) \text{ } xs$$


$$\text{fby}'_1 \text{ } _ \text{ } xs \text{ } (_ \cdot ys) := \text{map } (\lambda x. \text{err}_{\text{sync}}) \text{ } xs$$

$$\text{fby}_1 \text{ } v \text{ } (A \cdot xs) \text{ } ys := A \cdot \text{fby}'_1 \text{ (Some } v) \text{ } xs \text{ } ys$$

$$\text{fby}_1 \text{ } v \text{ } (v_x \cdot xs) \text{ } ys := v \cdot \text{fby}'_1 \text{ None } xs \text{ } ys$$

$$\text{fby}_1 \text{ } v \text{ } (\text{err} \cdot xs) \text{ } ys := \text{err} \cdot \text{map } (\lambda x. \text{err}) \text{ } xs$$

 **Theorem:** if $\text{fby } xs \text{ } ys$ is error-free, then $\text{FBY } xs \text{ } ys \text{ } (\text{fby } xs \text{ } ys)$

 **Theorem:** also, $\lfloor \text{fby } xs \text{ } ys \rfloor_A \preceq \text{fby}^\# \lfloor xs \rfloor_A \lfloor ys \rfloor_A$

Modular resetting of nodes

```
node count (init, incr : int)
  returns (n: int);
let
  n = init fby (n + incr);
tel
```

init	0	0	0	0	0	0	0 ...
incr	1	1	1	1	1	1	1 ...
n	0	1	2	3	4	5	6 ...

Modular resetting of nodes

```
node count (init, incr : int; r : bool)
  returns (n: int);
let
n = if r then init
    else (init fby (n + incr));
tel
```

init	0	0	0	0	0	0	0 ...
incr	1	1	1	1	1	1	1 ...
r	F	F	T	F	F	T	F ...
n	0	1	0	1	2	0	1 ...

Modular resetting of nodes

[Hamon, Pouzet, 2001]

reset f rs xs :=

let $cs = \text{true-until } rs$ **in**

merge cs (f (when cs xs))

(reset f (whenot cs rs) (whenot cs xs))

init	0	0	0	0	0	0	0 ...
incr	1	1	1	1	1	1	1 ...
r	F	F	T	F	F	T	F ...
n	0	1	0	1	2	0	1 ...

Modular resetting of nodes

[Hamon, Pouzet, 2001]

```
reset  $f$   $rs$   $xs$  :=
```

```
let  $cs$  = true-until  $rs$  in
```

```
merge  $cs$  ( $f$  (when  $cs$   $xs$ ))
```

```
  (reset  $f$  (whenot  $cs$   $rs$ ) (whenot  $cs$   $xs$ ))
```

init	0	0	0	0	0	0	0 ...
incr	1	1	1	1	1	1	1 ...
r	F	F	T	F	F	T	F ...
n	0	1	0	1	2	0	1 ...

c = reset counter(**init**, incr) every r;

Modular resetting of nodes

[Hamon, Pouzet, 2001]

```
reset  $f$   $rs$   $xs$  :=
```

```
let  $cs = \text{true-until } rs$  in
```

```
merge  $cs$  ( $f$  (when  $cs$   $xs$ ))
```

```
(reset  $f$  (whenot  $cs$   $rs$ ) (whenot  $cs$   $xs$ ))
```

```
 $c = \text{reset counter}(\text{init}, \text{incr}) \text{ every } r;$ 
```

xs	x_1	x_2	x_3	x_4	x_5	x_6	x_7	\dots
rs	F	F	T	F	F	T	F	\dots
ys	y_1	y_2	y_3	y_4	y_5	y_6	y_7	\dots

[Bourke, Brun, Pouzet, 2020]

$$f \vdash [x_1 \cdot x_2] \Downarrow [y_1 \cdot y_2]$$
$$\wedge f \vdash [x_4 \cdot x_5 \cdot x_6] \Downarrow [y_4 \cdot y_5 \cdot y_6]$$
$$\wedge f \vdash [x_6 \cdot x_7] \Downarrow [y_6 \cdot y_7]$$
$$\wedge \dots$$

Modular resetting of nodes

[Hamon, Pouzet, 2001]

reset f rs xs :=

let $cs = \text{true-until } rs$ **in**

merge cs (f (when cs xs))

(reset f (whenot cs rs) (whenot cs xs))

xs	x_1	x_2	x_3	x_4	x_5	x_6	x_7	\dots
rs	F	F	T	F	F	T	F	\dots
ys	y_1	y_2	y_3	y_4	y_5	y_6	y_7	\dots

[Gérard, 2013]

$sreset_f rs xs$	$:= sreset'_f rs xs (f xs)$
$sreset'_f (T \cdot rs) xs ys$	$\simeq sreset'_f (F \cdot rs) xs (f xs)$
$sreset'_f (F \cdot rs) (x \cdot xs) (y \cdot ys)$	$\simeq y \cdot (sreset'_f rs xs ys)$
$sreset'_f (A \cdot rs) (x \cdot xs) (y \cdot ys)$	$\simeq y \cdot (sreset'_f rs xs ys)$

Modular resetting of nodes

[Hamon, Pouzet, 2001]

```
reset  $f$   $rs$   $xs$  :=
```

```
let  $cs = \text{true-until } rs$  in
```


```
merge  $cs$  ( $f$  (when  $cs$   $xs$ ))
```

```
(reset  $f$  (whenot  $cs$   $rs$ ) (whenot  $cs$   $xs$ ))
```

xs	x_1	x_2	x_3	x_4	x_5	x_6	x_7	\dots
rs	F	F	T	F	F	T	F	\dots
ys	y_1	y_2	y_3	y_4	y_5	y_6	y_7	\dots

[Gérard, 2013]

```
sreset $_f$   $rs$   $xs$  := sreset' $_f$   $rs$   $xs$  ( $f$   $xs$ )  
sreset' $_f$  ( $T \cdot rs$ )  $xs$   $ys$   $\simeq$  sreset' $_f$  ( $F \cdot rs$ )  $xs$  ( $f$   $xs$ )  
sreset' $_f$  ( $F \cdot rs$ ) ( $x \cdot xs$ ) ( $y \cdot ys$ )  $\simeq$   $y \cdot$  (sreset' $_f$   $rs$   $xs$   $ys$ )  
sreset' $_f$  ( $A \cdot rs$ ) ( $x \cdot xs$ ) ( $y \cdot ys$ )  $\simeq$   $y \cdot$  (sreset' $_f$   $rs$   $xs$   $ys$ )
```

 **Theorem:** if $\text{clock}(rs) \simeq \text{clock}(xs)$ then $\text{reset}_f rs xs \simeq \text{sreset}_f rs xs$

Modular resetting of nodes

[Hamon, Pouzet, 2001]

reset f rs xs :=

let $cs = \text{true-until } rs$ **in**


merge cs (f (when cs xs))

(reset f (whenot cs rs) (whenot cs xs))

xs	x_1	x_2	x_3	x_4	x_5	x_6	x_7	\dots
rs	F	F	T	F	F	T	F	\dots
ys	y_1	y_2	y_3	y_4	y_5	y_6	y_7	\dots

[Gérard, 2013]

$$\begin{aligned} \text{sreset}_f rs xs &:= \text{sreset}'_f rs xs (f xs) \\ \text{sreset}'_f (T \cdot rs) xs ys &\simeq \text{sreset}'_f (F \cdot rs) xs (f xs) \\ \text{sreset}'_f (F \cdot rs) (x \cdot xs) (y \cdot ys) &\simeq y \cdot (\text{sreset}'_f rs xs ys) \\ \text{sreset}'_f (A \cdot rs) (x \cdot xs) (y \cdot ys) &\simeq y \cdot (\text{sreset}'_f rs xs ys) \end{aligned}$$

 **Theorem:** if $\text{clock}(rs) \simeq \text{clock}(xs)$ then $\text{reset}_f rs xs \simeq \text{sreset}_f rs xs$

For all f satisfying:

► $\forall xs, f(A \cdot xs) \simeq A \cdot f(xs)$

Modular resetting of nodes

[Hamon, Pouzet, 2001]

reset f rs xs :=

let $cs = \text{true-until } rs$ in


merge cs (f (when cs xs))

(reset f (whenot cs rs) (whenot cs xs))

xs	x_1	x_2	x_3	x_4	x_5	x_6	x_7	\dots
rs	F	F	T	F	F	T	F	\dots
ys	y_1	y_2	y_3	y_4	y_5	y_6	y_7	\dots

[Gérard, 2013]

$$\begin{aligned} \text{sreset}_f rs xs &:= \text{sreset}'_f rs xs (f xs) \\ \text{sreset}'_f (T \cdot rs) xs ys &\simeq \text{sreset}'_f (F \cdot rs) xs (f xs) \\ \text{sreset}'_f (F \cdot rs) (x \cdot xs) (y \cdot ys) &\simeq y \cdot (\text{sreset}'_f rs xs ys) \\ \text{sreset}'_f (A \cdot rs) (x \cdot xs) (y \cdot ys) &\simeq y \cdot (\text{sreset}'_f rs xs ys) \end{aligned}$$

 **Theorem:** if $\text{clock}(rs) \simeq \text{clock}(xs)$ then $\text{reset}_f rs xs \simeq \text{sreset}_f rs xs$

For all f satisfying:

- ▶ $\forall xs, f(A \cdot xs) \simeq A \cdot f(xs)$
- ▶ $\forall n xs, f(\text{take } n xs) \simeq \text{take } n f(xs)$

Modular resetting of nodes

[Hamon, Pouzet, 2001]

reset f rs xs :=

let $cs = \text{true-until } rs$ **in**


merge cs (f (when cs xs))

(reset f (whenot cs rs) (whenot cs xs))

xs	x_1	x_2	x_3	x_4	x_5	x_6	x_7	\dots
rs	F	F	T	F	F	T	F	\dots
ys	y_1	y_2	y_3	y_4	y_5	y_6	y_7	\dots

[Gérard, 2013]

$$\begin{aligned} \text{sreset}_f rs xs &:= \text{sreset}'_f rs xs (f xs) \\ \text{sreset}'_f (T \cdot rs) xs ys &\simeq \text{sreset}'_f (F \cdot rs) xs (f xs) \\ \text{sreset}'_f (F \cdot rs) (x \cdot xs) (y \cdot ys) &\simeq y \cdot (\text{sreset}'_f rs xs ys) \\ \text{sreset}'_f (A \cdot rs) (x \cdot xs) (y \cdot ys) &\simeq y \cdot (\text{sreset}'_f rs xs ys) \end{aligned}$$

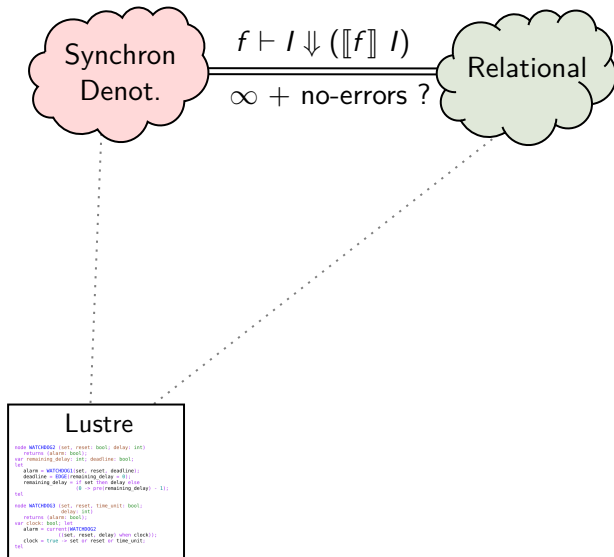
 **Theorem:** if $\text{clock}(rs) \simeq \text{clock}(xs)$ then $\text{reset}_f rs xs \simeq \text{sreset}_f rs xs$

For all f satisfying:

- ▶ $\forall xs, f(A \cdot xs) \simeq A \cdot f(xs)$
- ▶ $\forall n xs, f(\text{take } n xs) \simeq \text{take } n f(xs)$
- ▶ $\forall n xs, \text{if } \text{tl}^n xs \preceq A^\omega \text{ then } \text{tl}^n(f(xs)) \preceq A^\omega$

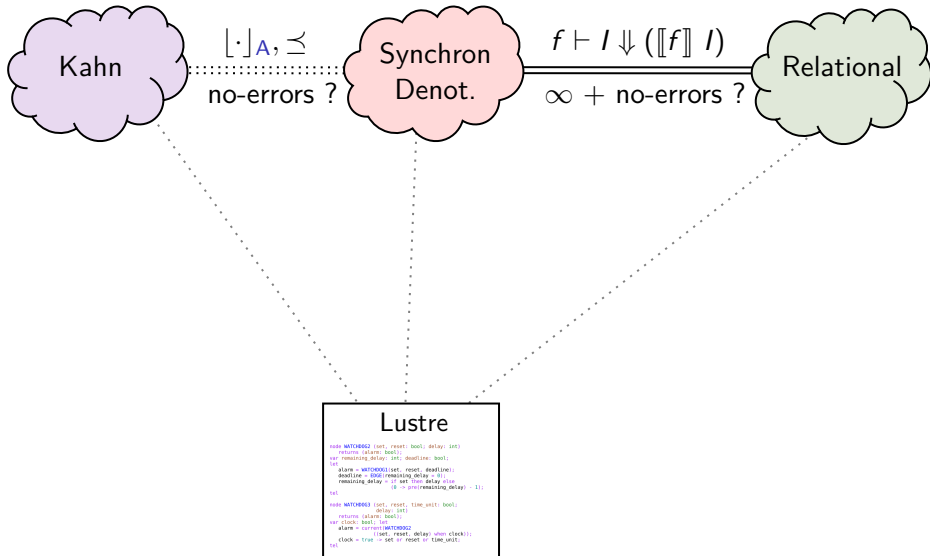
The plan

Error handling



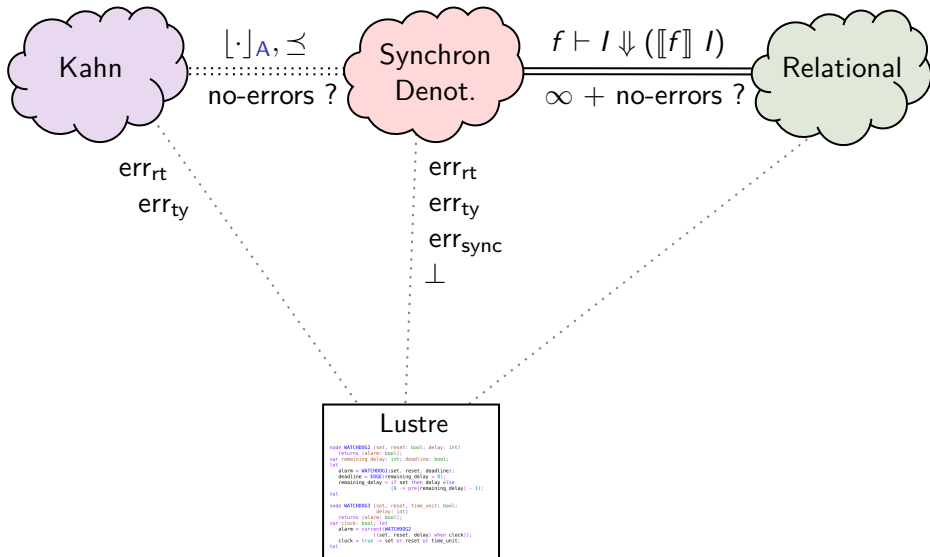
The plan

Error handling



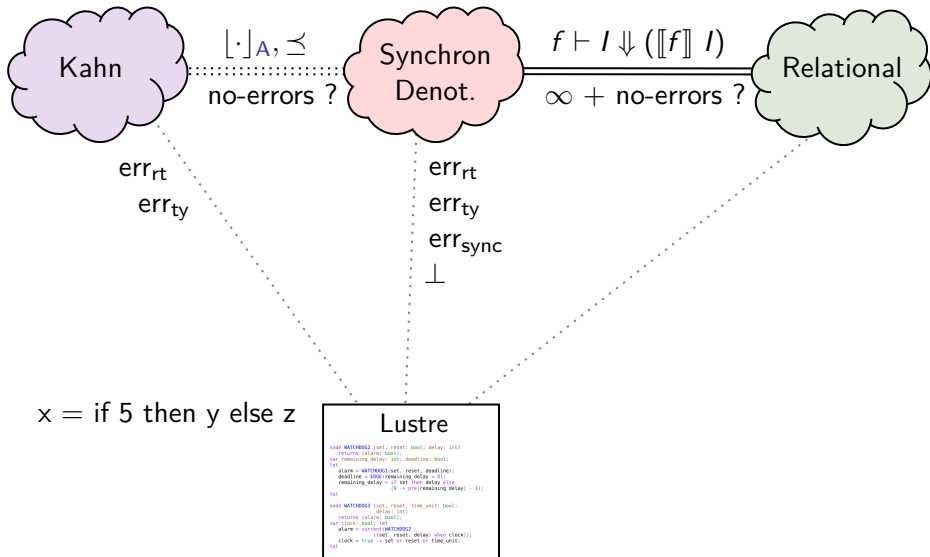
The plan

Error handling



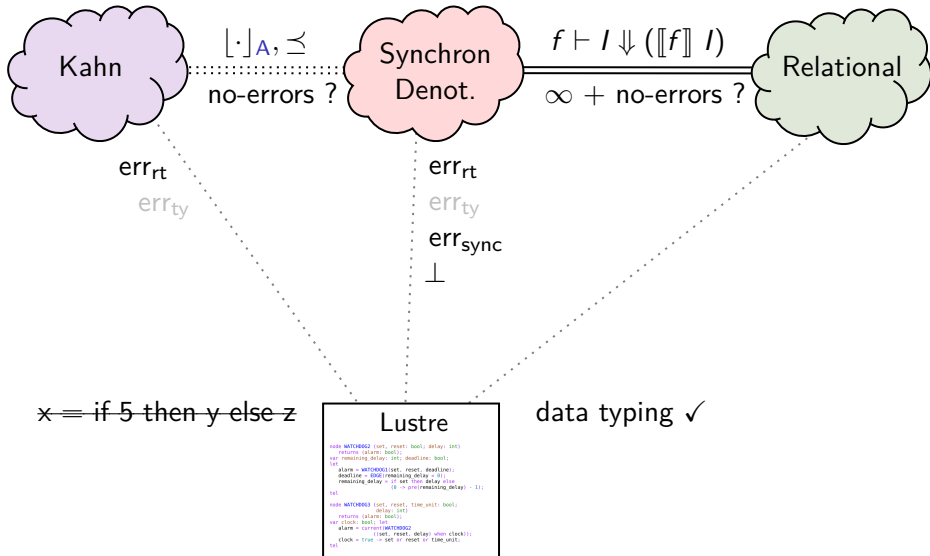
The plan

Error handling



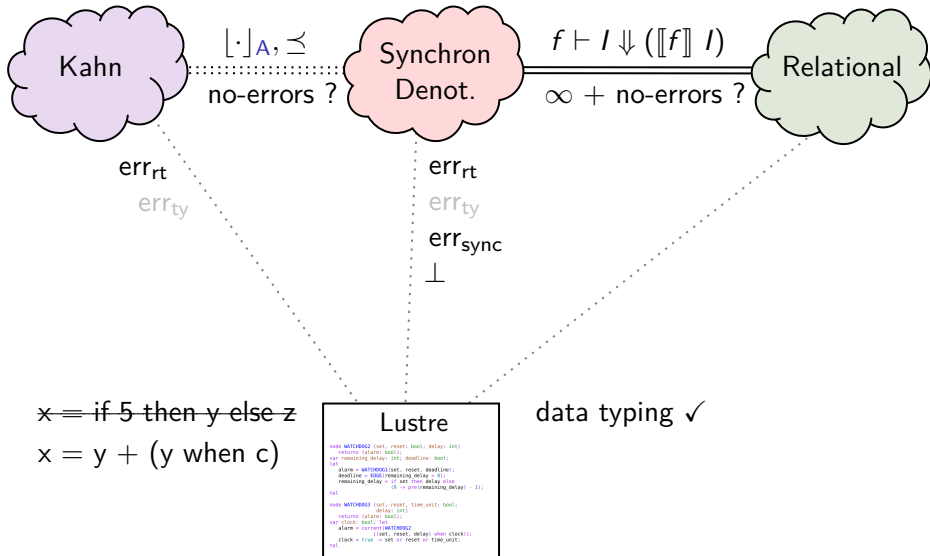
The plan

Error handling



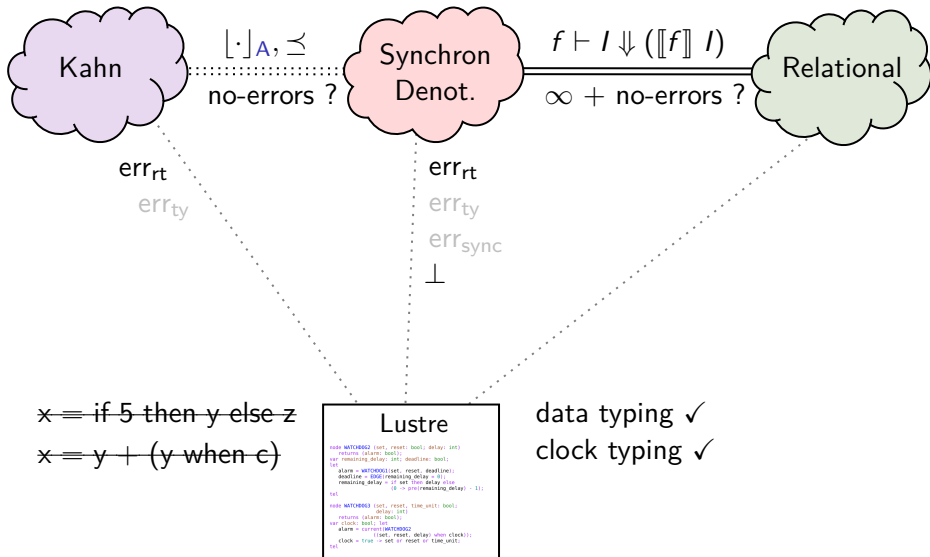
The plan

Error handling



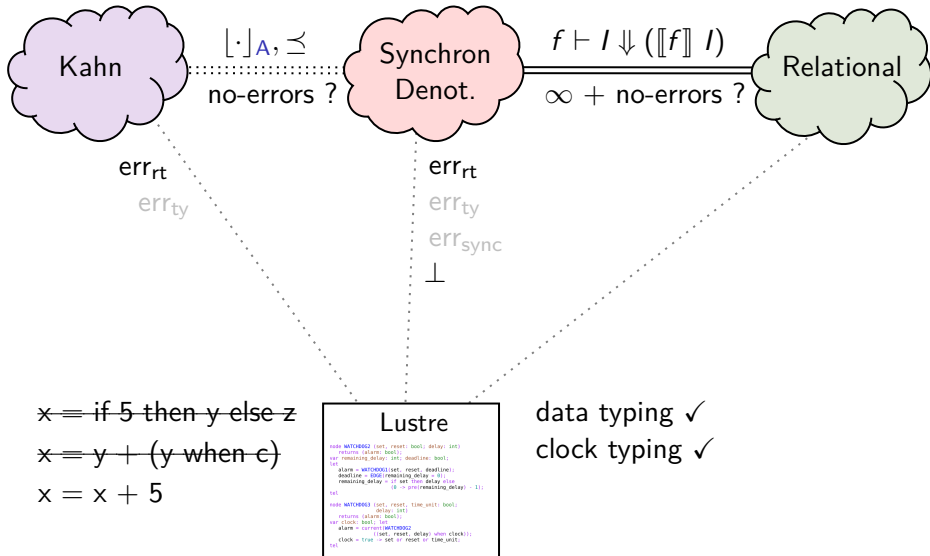
The plan

Error handling



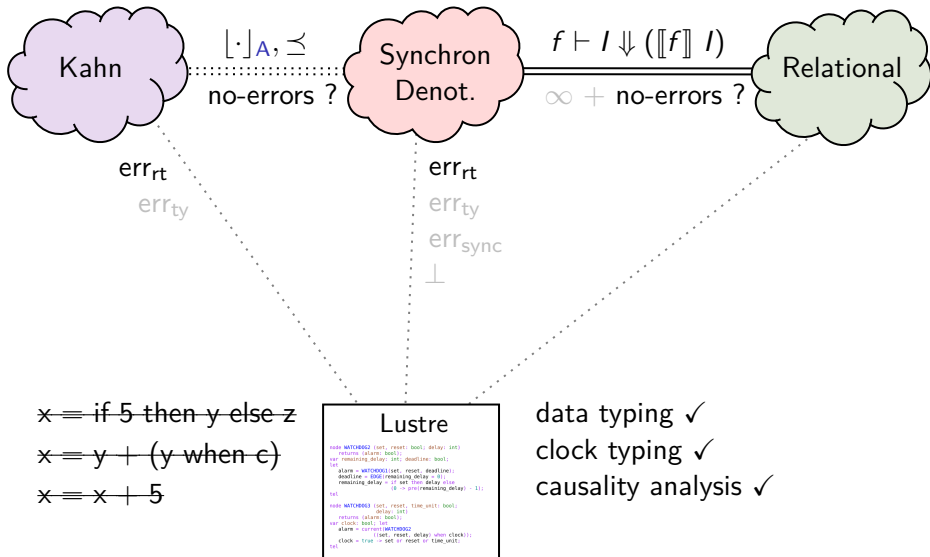
The plan

Error handling



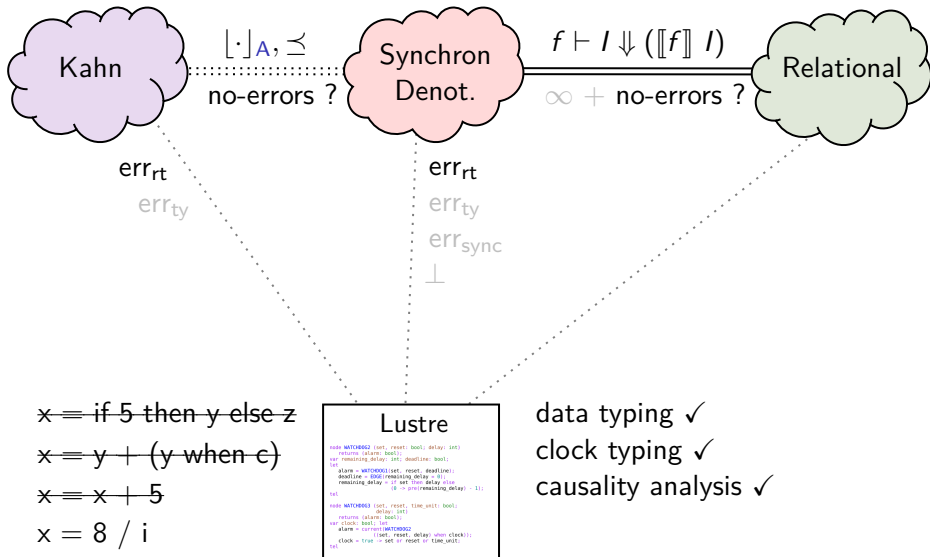
The plan

Error handling



The plan

Error handling



$x \equiv \text{if } 5 \text{ then } y \text{ else } z$

$x \equiv y + (y \text{ when } c)$

$x \equiv x + 5$

$x = 8 / i$

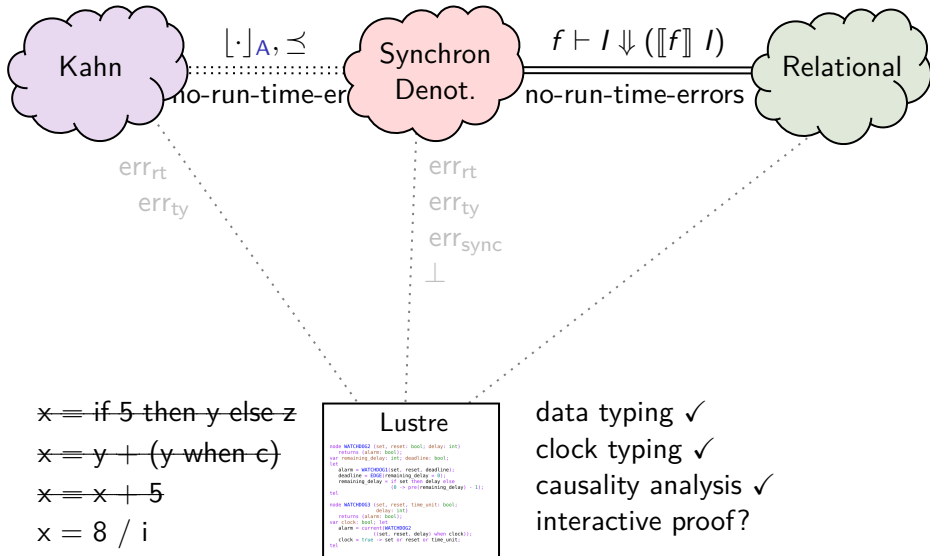
data typing ✓

clock typing ✓

causality analysis ✓

The plan

Error handling



Static analysis

Rcall: only few well-typed operators can fail
(cf. CompCert/cfrontend/Cop.v)

Static analysis

Rcall: only few well-typed operators can fail
(cf. CompCert/cfrontend/Cop.v)

`n/0`

`MIN_INT / -1`

`(int)NaN`

`n%0`

`MIN_INT % -1`

`(int)Infty`

`n >> 64`

`(int)2^63-1.0`

`n << 64`

`(int)-2^63.0`

Static analysis

Rcall: only few well-typed operators can fail
(cf. CompCert/cfrontend/Cop.v)

$n/0$	$n\%0$	$n \gg 64$	$n \ll 64$
<code>MIN_INT / -1</code>	<code>MIN_INT % -1</code>		
<code>(int)NaN</code>	<code>(int)Infty</code>	<code>(int)2^63-1.0</code>	<code>(int)-2^63.0</code>

Procedure: prohibit these operations when the operand is unknown

- ▶ `check-ops (4 % x) = F`
- ▶ `check-ops (x / 2) = T`
- ▶ ...


Static analysis

Rcall: only few well-typed operators can fail
(cf. CompCert/cfrontend/Cop.v)

$n/0$	$n\%0$	$n \gg 64$	$n \ll 64$
$\text{MIN_INT} / -1$	$\text{MIN_INT} \% -1$		
$(\text{int})\text{NaN}$	$(\text{int})\text{Infty}$	$(\text{int})2^{63}-1.0$	$(\text{int})-2^{63}.0$

Procedure: prohibit these operations when the operand is unknown

- ▶ $\text{check-ops}(4 \% x) = \text{F}$
- ▶ $\text{check-ops}(x / 2) = \text{T}$
- ▶ ...

 **Theorem:** $\forall x s f$, if $\text{check-ops}(f) = \text{T}$ then no-run-time-errors f $x s$

Conclusion

Reinforcement of the correctness theorem

Theorem (before¹)

if compile $f = \text{OK}$ asm

and welltyped-inputs f xs

and $f \vdash xs \Downarrow ys$

then $asm \Downarrow \langle \text{Load}(xs(i)) \cdot \text{Store}(ys(i)) \rangle_{i=0}^{\infty}$

¹with state machines

Conclusion

Reinforcement of the correctness theorem

Theorem (before¹)

if compile $f = \text{OK}$ asm
and welltyped-inputs f xs
and $f \vdash xs \Downarrow ys$
then $asm \Downarrow \langle \text{Load}(xs(i)) \cdot \text{Store}(ys(i)) \rangle_{i=0}^{\infty}$

Theorem (after²)

if compile $f = \text{OK}$ asm
and welltyped-inputs f xs
and no-run-time-errors f xs
then $\exists ys, f \vdash xs \Downarrow ys \wedge asm \Downarrow \langle \text{Load}(xs(i)) \cdot \text{Store}(ys(i)) \rangle_{i=0}^{\infty}$

¹with state machines

²without state machines

Conclusion

Reinforcement of the correctness theorem

Theorem (before¹)

if compile $f = \text{OK}$ asm
and welltyped-inputs f xs
and $f \vdash xs \Downarrow ys$
then $asm \Downarrow \langle \text{Load}(xs(i)) \cdot \text{Store}(ys(i)) \rangle_{i=0}^{\infty}$

Theorem (after²)

if compile $f = \text{OK}$ asm
and welltyped-inputs f xs
and check-ops (f) = T
then $\exists ys, f \vdash xs \Downarrow ys \wedge asm \Downarrow \langle \text{Load}(xs(i)) \cdot \text{Store}(ys(i)) \rangle_{i=0}^{\infty}$

¹with state machines

²without state machines

Conclusion

We have defined:

Conclusion

We have defined:

- ▶ a synchronous denotational semantics for a dataflow language with modular reinitialization within Vélus

Conclusion

We have defined:

- ▶ a synchronous denotational semantics for a dataflow language with modular reinitialization within Vélus
- ▶ an accurate error modeling in the context of fixed-point evaluation in Rocq

Conclusion

We have defined:

- ▶ a synchronous denotational semantics for a dataflow language with modular reinitialization within Vélus
- ▶ an accurate error modeling in the context of fixed-point evaluation in Rocq
- ▶ criteria for its correspondence with Kahn's model

Conclusion

We have defined:

- ▶ a synchronous denotational semantics for a dataflow language with modular reinitialization within Vélus
- ▶ an accurate error modeling in the context of fixed-point evaluation in Rocq
- ▶ criteria for its correspondence with Kahn's model
- ▶ a static analysis to validate some executions

Conclusion

We have defined:

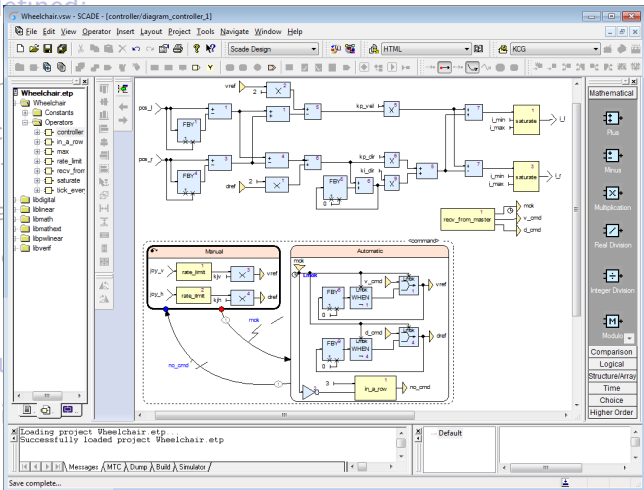
- ▶ a synchronous denotational semantics for a dataflow language with modular reinitialization within Vélus
- ▶ an accurate error modeling in the context of fixed-point evaluation in Rocq
- ▶ criteria for its correspondence with Kahn's model
- ▶ a static analysis to validate some executions

(not so) Future work:

Conclusion

We have defined:

- ▶ a sync
- with m
- ▶ an acc
- evalua
- ▶ criteria
- ▶ a stati
- (not so) Fu
- ▶ extens



ge

Conclusion

We have defined:

- ▶ a synchronous denotational semantics for a dataflow language with modular reinitialization within Vélus
- ▶ an accurate error modeling in the context of fixed-point evaluation in Rocq
- ▶ criteria for its correspondence with Kahn's model
- ▶ a static analysis to validate some executions

(not so) Future work:

- ▶ extension to hierarchical states machines
- ▶ more sophisticated static analysis

Conclusion

We have defined:

- ▶ a synchronous denotational semantics for a dataflow language with modular reinitialization within Vélus
- ▶ an accurate error modeling in the context of fixed-point evaluation in Rocq
- ▶ criteria for its correspondence with Kahn's model
- ▶ a static analysis to validate some executions

(not so) Future work:

- ▶ extension to hierarchical states machines
- ▶ more sophisticated static analysis
- ▶ more verification principles