Towards a denotational semantics of streams for a verified Lustre compiler

Timothy Bourke **Paul Jeanmaire** Marc Pouzet

ENS, Inria, équipe Parkas

TYPES'22, 21 juin 2022

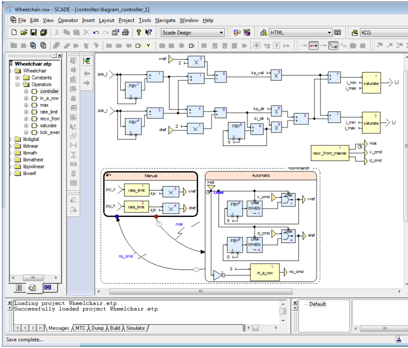
Synchronous programming

Model-based development

- block/node = system = **function of streams**
- line = signal = **stream of values**

Lustre

- Language of dataflow equations
- Compiled to C



Scade 6 suite

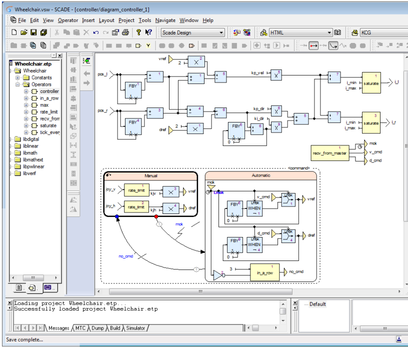
Synchronous programming

Model-based development

- block/node = system = **function of streams**
- line = signal = **stream of values**

Lustre

- Language of dataflow equations
- Compiled to C



Scade 6 suite

```
node f (c : bool) returns (o : int);
var (x y : int);
let
  x = 0 → pre y;
  y = x + 1;
  o = x when c;
tel;
```

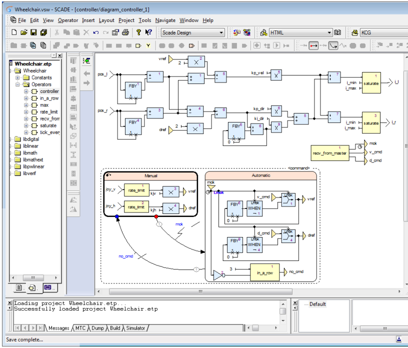
Synchronous programming

Model-based development

- block/node = system = **function of streams**
- line = signal = **stream of values**

Lustre

- Language of dataflow equations
- Compiled to C



Scade 6 suite

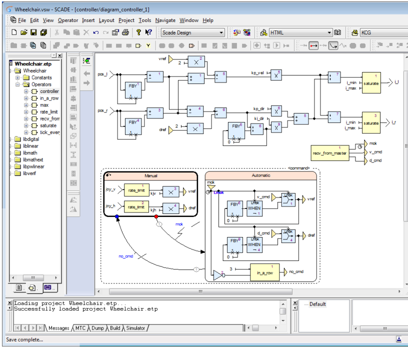
```
node f (c : bool) returns (o : int);
var (x y : int);
let
  x = 0 → pre y;
  y = x + 1;
  o = x when c;
tel;
```

c	T	T	T	T	T	T	T	T	T
x	0	1	2	3	4	5	6	7	8
y	1	2	3	4	5	6	7	8	9

Synchronous programming

Model-based development

- block/node = system = **function of streams**
- line = signal = **stream of values**



Scade 6 suite

Lustre

- Language of dataflow equations
- Compiled to C

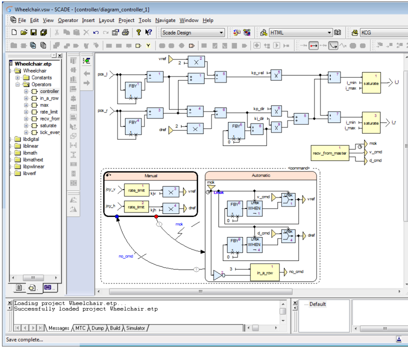
```
node f (c : bool) returns (o : int);
var (x y : int);
let
  x = 0 → pre y;
  y = x + 1;
  o = x when c;
tel;
```

c	T	T	T	T	T	T	T	T	T
x	0	1	2	3	4	5	6	7	8
y	1	2	3	4	5	6	7	8	9
o	0	1	2	3	4	5	6	7	8

Synchronous programming

Model-based development

- block/node = system = **function of streams**
- line = signal = **stream of values**



Scade 6 suite

Lustre

- Language of dataflow equations
- Compiled to C

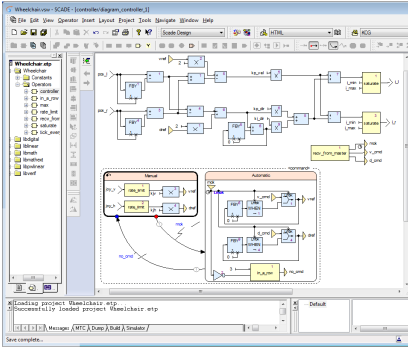
```
node f (c : bool) returns (o : int);
var (x y : int);
let
  x = 0 -> pre y;
  y = x + 1;
  o = x when c;
tel;
```

c	T	**F**	T	T	**F**	T	T	T	T
x	0	1	2	3	4	5	6	7	8
y	1	2	3	4	5	6	7	8	9
o	0		2	3		5	6	7	8

Synchronous programming

Model-based development

- block/node = system = **function of streams**
- line = signal = **stream of values**



Scade 6 suite

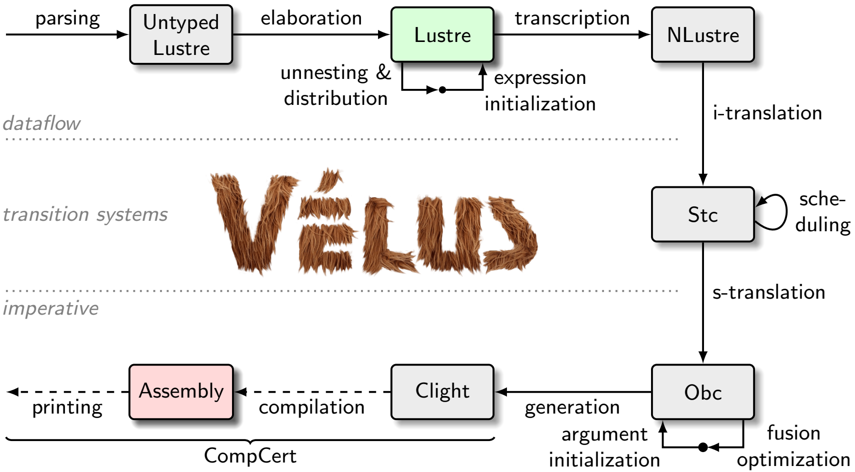
Lustre

- Language of dataflow equations
- Compiled to C

```
node f (c : bool) returns (o : int);
var (x y : int);
let
  x = 0 → pre y;
  y = x + 1;
  o = x when c;
tel;
```

c	T	**F**	T	T	**F**	T	T	T	T
x	0	1	2	3	4	5	6	7	8
y	1	2	3	4	5	6	7	8	9
o	0	**A**	2	3	**A**	5	6	7	8

A formally verified compiler for Lustre



A formally verified compiler for Lustre

Correctness theorem (simplified)

$$\forall f, xs, ys,\ \text{sem\_node}\ f\ xs\ ys \implies \exists T \in \text{Traces}\,(\mathit{compile}\ f),\ T \simeq (xs, ys)$$

A formally verified compiler for Lustre

Correctness theorem (simplified)

$$\forall f, xs, ys,\ \text{sem\_node}\ f\ \ xs\ \ ys \implies$$
$$\exists T \in \text{Traces}\,(\mathit{compile}\ \ f),\ T \simeq (xs, ys)$$

Relational semantics (simplified too)

```
Inductive sem_node: node → list (Stream D) → list (Stream D) → ℙ :=
  Snode: ∀ H f b xs ys,
      Forall2 (sem_var H) f.(n_in) xs →
      Forall2 (sem_var H) f.(n_out) ys →
      b = clocks_of xs →
      Forall (sem_equation H b) n.(n_eqs) →
      sem_node f xs ys

with sem_equation: history → Stream 𝔹 → equation → ℙ :=
  Seq: ∀ H b xs es ss,
      Forall2 (sem_exp H b) es ss →
      Forall2 (sem_var H) xs (concat ss) →
      sem_equation H b (xs, es)

with sem_exp : history → Stream 𝔹 → exp → list (Stream svalue) → ℙ :=
  ...
```

A denotational semantics for Lustre

In terms of Kahn networks

$$
\begin{aligned}
\texttt{v} &= v.\texttt{v} \\[4pt]
\epsilon \;\texttt{+}\; xs &= \epsilon \\
xs \;\texttt{+}\; \epsilon &= \epsilon \\
x.xs \;\texttt{+}\; y.ys &= (x+y).(xs \;\texttt{+}\; ys) \\[4pt]
\epsilon \;\texttt{->}\; \texttt{pre}\; ys &= \epsilon \\
x.xs \;\texttt{->}\; \texttt{pre}\; \epsilon &= x.\epsilon \\
x.xs \;\texttt{->}\; \texttt{pre}\; y.ys &= x.(y.ys \;\texttt{->}\; \texttt{pre}\; ys) \\
&= x.ys \\[4pt]
\epsilon \;\texttt{when}\; cs &= \epsilon \\
xs \;\texttt{when}\; \epsilon &= \epsilon \\
x.xs \;\texttt{when}\; true.cs &= x.(xs \;\texttt{when}\; cs) \\
x.xs \;\texttt{when}\; false.cs &= xs \;\texttt{when}\; cs \\[4pt]
\texttt{current}(v, xs, \epsilon) &= \epsilon \\
\texttt{current}(v, xs, false.cs) &= v.\texttt{current}(v, xs, cs) \\
\texttt{current}(v, \epsilon, true.cs) &= \epsilon \\
\texttt{current}(v, x.xs, true.cs) &= x.\texttt{current}(x, xs, cs)
\end{aligned}
$$

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\perp$ (or ϵ) is the empty stream

A denotational semantics for Lustre

In terms of Kahn networks

$$
\begin{aligned}
\texttt{v} &= v.\texttt{v} \\[4pt]
\epsilon \;\texttt{+}\; xs &= \epsilon \\
xs \;\texttt{+}\; \epsilon &= \epsilon \\
x.xs \;\texttt{+}\; y.ys &= (x + y).(xs \;\texttt{+}\; ys) \\[4pt]
\epsilon \;\texttt{->}\; \texttt{pre}\; ys &= \epsilon \\
x.xs \;\texttt{->}\; \texttt{pre}\; \epsilon &= x.\epsilon \\
x.xs \;\texttt{->}\; \texttt{pre}\; y.ys &= x.(y.ys \;\texttt{->}\; \texttt{pre}\; ys) \\
&= x.ys \\[4pt]
\epsilon \;\texttt{when}\; cs &= \epsilon \\
xs \;\texttt{when}\; \epsilon &= \epsilon \\
x.xs \;\texttt{when}\; true.cs &= x.(xs \;\texttt{when}\; cs) \\
x.xs \;\texttt{when}\; false.cs &= xs \;\texttt{when}\; cs \\[4pt]
\texttt{current}(v, xs, \epsilon) &= \epsilon \\
\texttt{current}(v, xs, false.cs) &= v.\texttt{current}(v, xs, cs) \\
\texttt{current}(v, \epsilon, true.cs) &= \epsilon \\
\texttt{current}(v, x.xs, true.cs) &= x.\texttt{current}(x, xs, cs)
\end{aligned}
$$

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\perp$ (or ϵ) is the empty stream

$$
\begin{aligned}
x &= 0 \rightarrow \mathrm{pre}\ y; \\
y &= x + 1;
\end{aligned}
$$

x	$\perp$
y	$\perp$

A denotational semantics for Lustre

In terms of Kahn networks

$$
\begin{array}{|ll|}
\hline
\mathtt{v} & = v.\mathtt{v} \\
 & \\
\epsilon\ \mathtt{+}\ xs & = \epsilon \\
xs\ \mathtt{+}\ \epsilon & = \epsilon \\
x.xs\ \mathtt{+}\ y.ys & = (x+y).(xs\ \mathtt{+}\ ys) \\
 & \\
\epsilon\ \mathtt{->}\ \mathtt{pre}\ ys & = \epsilon \\
x.xs\ \mathtt{->}\ \mathtt{pre}\ \epsilon & = x.\epsilon \\
x.xs\ \mathtt{->}\ \mathtt{pre}\ y.ys & = x.(y.ys\ \mathtt{->}\ \mathtt{pre}\ ys) \\
 & = x.ys \\
 & \\
\epsilon\ \mathtt{when}\ cs & = \epsilon \\
xs\ \mathtt{when}\ \epsilon & = \epsilon \\
x.xs\ \mathtt{when}\ true.cs & = x.(xs\ \mathtt{when}\ cs) \\
x.xs\ \mathtt{when}\ false.cs & = xs\ \mathtt{when}\ cs \\
 & \\
\mathtt{current}(v, xs, \epsilon) & = \epsilon \\
\mathtt{current}(v, xs, false.cs) & = v.\mathtt{current}(v, xs, cs) \\
\mathtt{current}(v, \epsilon, true.cs) & = \epsilon \\
\mathtt{current}(v, x.xs, true.cs) & = x.\mathtt{current}(x, xs, cs) \\
\hline
\end{array}
$$

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\perp$ (or ϵ) is the empty stream

$$
\begin{array}{l}
x = 0 \rightarrow \mathrm{pre}\ y; \\
y = x + 1;
\end{array}
\qquad
\begin{array}{c|l}
\mathrm{x} & 0\perp \\
\mathrm{y} & \perp
\end{array}
$$

A denotational semantics for Lustre

In terms of Kahn networks

$$
\begin{array}{ll}
\texttt{v} & = v.\texttt{v} \\[6pt]
\epsilon \;\texttt{+}\; xs & = \epsilon \\
xs \;\texttt{+}\; \epsilon & = \epsilon \\
x.xs \;\texttt{+}\; y.ys & = (x+y).(xs \;\texttt{+}\; ys) \\[6pt]
\epsilon \;\texttt{->}\; \texttt{pre}\; ys & = \epsilon \\
x.xs \;\texttt{->}\; \texttt{pre}\; \epsilon & = x.\epsilon \\
x.xs \;\texttt{->}\; \texttt{pre}\; y.ys & = x.(y.ys \;\texttt{->}\; \texttt{pre}\; ys) \\
 & = x.ys \\[6pt]
\epsilon \;\texttt{when}\; cs & = \epsilon \\
xs \;\texttt{when}\; \epsilon & = \epsilon \\
x.xs \;\texttt{when}\; true.cs & = x.(xs \;\texttt{when}\; cs) \\
x.xs \;\texttt{when}\; false.cs & = xs \;\texttt{when}\; cs \\[6pt]
\texttt{current}(v, xs, \epsilon) & = \epsilon \\
\texttt{current}(v, xs, false.cs) & = v.\texttt{current}(v, xs, cs) \\
\texttt{current}(v, \epsilon, true.cs) & = \epsilon \\
\texttt{current}(v, x.xs, true.cs) & = x.\texttt{current}(x, xs, cs)
\end{array}
$$

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\perp$ (or ϵ) is the empty stream

$$
\begin{array}{l}
x = 0 \rightarrow \mathrm{pre}\; y; \\
y = x + 1;
\end{array}
\qquad
\begin{array}{c|l}
x & 0\perp \\
y & 1\perp
\end{array}
$$

A denotational semantics for Lustre

In terms of Kahn networks

$$
\begin{array}{ll}
\mathtt{v} & = v.\mathtt{v} \\
\\
\epsilon \;\mathtt{+}\; xs & = \epsilon \\
xs \;\mathtt{+}\; \epsilon & = \epsilon \\
x.xs \;\mathtt{+}\; y.ys & = (x+y).(xs \;\mathtt{+}\; ys) \\
\\
\epsilon \;\mathtt{->}\; \mathtt{pre}\; ys & = \epsilon \\
x.xs \;\mathtt{->}\; \mathtt{pre}\; \epsilon & = x.\epsilon \\
x.xs \;\mathtt{->}\; \mathtt{pre}\; y.ys & = x.(y.ys \;\mathtt{->}\; \mathtt{pre}\; ys) \\
 & = x.ys \\
\\
\epsilon \;\mathtt{when}\; cs & = \epsilon \\
xs \;\mathtt{when}\; \epsilon & = \epsilon \\
x.xs \;\mathtt{when}\; true.cs & = x.(xs \;\mathtt{when}\; cs) \\
x.xs \;\mathtt{when}\; false.cs & = xs \;\mathtt{when}\; cs \\
\\
\mathtt{current}(v, xs, \epsilon) & = \epsilon \\
\mathtt{current}(v, xs, false.cs) & = v.\mathtt{current}(v, xs, cs) \\
\mathtt{current}(v, \epsilon, true.cs) & = \epsilon \\
\mathtt{current}(v, x.xs, true.cs) & = x.\mathtt{current}(x, xs, cs)
\end{array}
$$

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\bot$ (or ϵ) is the empty stream

$$
\begin{array}{l}
x = 0 \rightarrow \mathrm{pre}\ y; \\
y = x + 1;
\end{array}
\qquad
\begin{array}{c|l}
\mathrm{x} & 01\bot \\
\mathrm{y} & 1\bot
\end{array}
$$

A denotational semantics for Lustre

In terms of Kahn networks

$$
\begin{array}{ll}
\texttt{v} & = v.\texttt{v} \\
\\
\epsilon \;\texttt{+}\; xs & = \epsilon \\
xs \;\texttt{+}\; \epsilon & = \epsilon \\
x.xs \;\texttt{+}\; y.ys & = (x+y).(xs \;\texttt{+}\; ys) \\
\\
\epsilon \;\texttt{->}\; \texttt{pre}\; ys & = \epsilon \\
x.xs \;\texttt{->}\; \texttt{pre}\; \epsilon & = x.\epsilon \\
x.xs \;\texttt{->}\; \texttt{pre}\; y.ys & = x.(y.ys \;\texttt{->}\; \texttt{pre}\; ys) \\
 & = x.ys \\
\\
\epsilon \;\texttt{when}\; cs & = \epsilon \\
xs \;\texttt{when}\; \epsilon & = \epsilon \\
x.xs \;\texttt{when}\; true.cs & = x.(xs \;\texttt{when}\; cs) \\
x.xs \;\texttt{when}\; false.cs & = xs \;\texttt{when}\; cs \\
\\
\texttt{current}(v, xs, \epsilon) & = \epsilon \\
\texttt{current}(v, xs, false.cs) & = v.\texttt{current}(v, xs, cs) \\
\texttt{current}(v, \epsilon, true.cs) & = \epsilon \\
\texttt{current}(v, x.xs, true.cs) & = x.\texttt{current}(x, xs, cs)
\end{array}
$$

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\perp$ (or ϵ) is the empty stream

$$
\begin{array}{l}
x = 0 \rightarrow \mathrm{pre}\; y; \\
y = x + 1;
\end{array}
\qquad
\begin{array}{c|l}
x & 01\perp \\
y & 12\perp
\end{array}
$$

A denotational semantics for Lustre

In terms of Kahn networks

$\mathtt{v}$	$= v.\mathtt{v}$
$\epsilon\ \mathtt{+}\ xs$	$= \epsilon$
$xs\ \mathtt{+}\ \epsilon$	$= \epsilon$
$x.xs\ \mathtt{+}\ y.ys$	$= (x + y).(xs\ \mathtt{+}\ ys)$
$\epsilon\ \mathtt{->}\ \mathtt{pre}\ ys$	$= \epsilon$
$x.xs\ \mathtt{->}\ \mathtt{pre}\ \epsilon$	$= x.\epsilon$
$x.xs\ \mathtt{->}\ \mathtt{pre}\ y.ys$	$= x.(y.ys\ \mathtt{->}\ \mathtt{pre}\ ys)$
	$= x.ys$
$\epsilon\ \mathtt{when}\ cs$	$= \epsilon$
$xs\ \mathtt{when}\ \epsilon$	$= \epsilon$
$x.xs\ \mathtt{when}\ true.cs$	$= x.(xs\ \mathtt{when}\ cs)$
$x.xs\ \mathtt{when}\ false.cs$	$= xs\ \mathtt{when}\ cs$
$\mathtt{current}(v, xs, \epsilon)$	$= \epsilon$
$\mathtt{current}(v, xs, false.cs)$	$= v.\mathtt{current}(v, xs, cs)$
$\mathtt{current}(v, \epsilon, true.cs)$	$= \epsilon$
$\mathtt{current}(v, x.xs, true.cs)$	$= x.\mathtt{current}(x, xs, cs)$

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\perp$ (or ϵ) is the empty stream

$$x = 0 \rightarrow \mathrm{pre}\ y;$$
$$y = x + 1;$$

x	$012\perp$
y	$12\perp$

A denotational semantics for Lustre

In terms of Kahn networks

$\mathtt{v}$	$= v.\mathtt{v}$
ϵ `+` xs	$= \epsilon$
xs `+` ϵ	$= \epsilon$
$x.xs$ `+` $y.ys$	$= (x+y).(xs$ `+` $ys)$
ϵ `-> pre` ys	$= \epsilon$
$x.xs$ `-> pre` ϵ	$= x.\epsilon$
$x.xs$ `-> pre` $y.ys$	$= x.(y.ys$ `-> pre` $ys)$
	$= x.ys$
ϵ `when` cs	$= \epsilon$
xs `when` ϵ	$= \epsilon$
$x.xs$ `when` $true.cs$	$= x.(xs$ `when` $cs)$
$x.xs$ `when` $false.cs$	$= xs$ `when` cs
$\mathtt{current}(v, xs, \epsilon)$	$= \epsilon$
$\mathtt{current}(v, xs, false.cs)$	$= v.\mathtt{current}(v, xs, cs)$
$\mathtt{current}(v, \epsilon, true.cs)$	$= \epsilon$
$\mathtt{current}(v, x.xs, true.cs)$	$= x.\mathtt{current}(x, xs, cs)$

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\perp$ (or ϵ) is the empty stream

$$x = 0 \rightarrow \mathrm{pre}\ y;$$
$$y = x + 1;$$

x	$012\perp$
y	$123\perp$

A denotational semantics for Lustre

In terms of Kahn networks

`v`	$= v.$`v`
ϵ `+` xs	$= \epsilon$
xs `+` ϵ	$= \epsilon$
$x.xs$ `+` $y.ys$	$= (x + y).(xs$ `+` $ys)$
ϵ `-> pre` ys	$= \epsilon$
$x.xs$ `-> pre` ϵ	$= x.\epsilon$
$x.xs$ `-> pre` $y.ys$	$= x.(y.ys$ `-> pre` $ys)$
	$= x.ys$
ϵ `when` cs	$= \epsilon$
xs `when` ϵ	$= \epsilon$
$x.xs$ `when` $true.cs$	$= x.(xs$ `when` $cs)$
$x.xs$ `when` $false.cs$	$= xs$ `when` cs
`current`(v, xs, ϵ)	$= \epsilon$
`current`$(v, xs, false.cs)$	$= v.$`current`(v, xs, cs)
`current`$(v, \epsilon, true.cs)$	$= \epsilon$
`current`$(v, x.xs, true.cs)$	$= x.$`current`(x, xs, cs)

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\perp$ (or ϵ) is the empty stream

$$x = 0 \rightarrow \mathrm{pre}\ y;$$
$$y = x + 1;$$

x	0123456789101
y	1234567891011

A denotational semantics for Lustre

In terms of Kahn networks

v	$= v.\mathtt{v}$
ϵ + xs	$= \epsilon$
xs + ϵ	$= \epsilon$
$x.xs$ + $y.ys$	$= (x+y).(xs$ + $ys)$
ϵ -> pre ys	$= \epsilon$
$x.xs$ -> pre ϵ	$= x.\epsilon$
$x.xs$ -> pre $y.ys$	$= x.(y.ys$ -> pre $ys)$
	$= x.ys$
ϵ when cs	$= \epsilon$
xs when ϵ	$= \epsilon$
$x.xs$ when $true.cs$	$= x.(xs$ when $cs)$
$x.xs$ when $false.cs$	$= xs$ when cs
current(v, xs, ϵ)	$= \epsilon$
current$(v, xs, false.cs)$	$= v.$current(v, xs, cs)
current$(v, \epsilon, true.cs)$	$= \epsilon$
current$(v, x.xs, true.cs)$	$= x.$current(x, xs, cs)

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\bot$ (or ϵ) is the empty stream

$$x = 0 \rightarrow \mathrm{pre}\ y;$$
$$y = x + 1;$$

x	0123456789101
y	1234567891011

- Suitable for verification

A denotational semantics for Lustre

In terms of Kahn networks

$$
\begin{array}{ll}
\mathtt{v} & = v.\mathtt{v} \\
\\
\epsilon \ \mathtt{+}\ xs & = \epsilon \\
xs \ \mathtt{+}\ \epsilon & = \epsilon \\
x.xs \ \mathtt{+}\ y.ys & = (x+y).(xs \ \mathtt{+}\ ys) \\
\\
\epsilon \ \mathtt{->}\ \mathtt{pre}\ ys & = \epsilon \\
x.xs \ \mathtt{->}\ \mathtt{pre}\ \epsilon & = x.\epsilon \\
x.xs \ \mathtt{->}\ \mathtt{pre}\ y.ys & = x.(y.ys \ \mathtt{->}\ \mathtt{pre}\ ys) \\
 & = x.ys \\
\\
\epsilon \ \mathtt{when}\ cs & = \epsilon \\
xs \ \mathtt{when}\ \epsilon & = \epsilon \\
x.xs \ \mathtt{when}\ true.cs & = x.(xs \ \mathtt{when}\ cs) \\
x.xs \ \mathtt{when}\ false.cs & = xs \ \mathtt{when}\ cs \\
\\
\mathtt{current}(v, xs, \epsilon) & = \epsilon \\
\mathtt{current}(v, xs, false.cs) & = v.\mathtt{current}(v, xs, cs) \\
\mathtt{current}(v, \epsilon, true.cs) & = \epsilon \\
\mathtt{current}(v, x.xs, true.cs) & = x.\mathtt{current}(x, xs, cs)
\end{array}
$$

- Set of streams: $D^* \cup D^\omega$
- CPO with prefix order
- $\bot$ (or ϵ) is the empty stream

$$
\begin{aligned}
x &= 0 \rightarrow \mathrm{pre}\ y; \\
y &= x + 1;
\end{aligned}
$$

x	0123456789101
y	1234567891011

- Suitable for verification
- Witness of sem_node

A denotational semantics for Vélus

Coq implementation

- Thanks to a generic CPO library[1]
- Guardedness by using a transparent "not-yet" element

```
CoInductive Str {D} :=
| Eps : Str → Str
| Con : D → Str → Str.
```

```
CoFixpoint Str_bot :=
  (* empty stream *)
  Eps Str_bot.
```

[1] Christine Paulin-Mohring, *A constructive denotational semantics for Kahn networks in Coq*, From semantics to CS, 2007

A denotational semantics for Vélus

Coq implementation

- Thanks to a generic CPO library[1]
- Guardedness by using a transparent "not-yet" element

```
CoInductive Str {D} :=
| Eps : Str → Str
| Con : D → Str → Str.
```

```
CoFixpoint Str_bot :=
  (* empty stream *)
  Eps Str_bot.
```

Language semantics

```
Definition denot_exp (e : exp) :
 (* inputs *)                (* environment *)
 Str_prod SI —C→ Str 𝔹 —C→ Str_prod SI —C→ prod (numstreams e).

Definition denot_equation (equ : equation) :
 Str_prod SI —C→ Str 𝔹 —C→ Str_prod SI —C→ Str_prod SI.

Definition denot equ envI bs := FIXP (denot_equation equ envI bs).
```

[1] Christine Paulin-Mohring, *A constructive denotational semantics for Kahn networks in Coq*, From semantics to CS, 2007

A denotational semantics for Vélus

Coq implementation

- Thanks to a generic CPO library[1]
- Guardedness by using a transparent "not-yet" element

```
CoInductive Str {D} :=
| Eps
| Con
```

```
CoFixpoint Str_bot :=
```

Conjecture

```
Lemma ok_denot :
  ∀ equ envI bs,
   let env := FIXP (denot_equation equ envI bs)
   in sem_equation env bs equ.
```

Languag

```
Definition
 (* inputs
 Str_prod S                                    e).
```

```
Definition denot_equation (equ : equation) :
 Str_prod SI —C→ Str 𝔹 —C→ Str_prod SI —C→ Str_prod SI.
```

```
Definition denot equ envI bs := FIXP (denot_equation equ envI bs).
```

[1] Christine Paulin-Mohring, *A constructive denotational semantics for Kahn networks in Coq*, From semantics to CS, 2007

Integration into the compilation chain

Which type for streams?

Kahn

$D^* \cup D^\omega$

```
CoInductive Str {D} :=
| Eps : Str → Str
| Con : D → Str → Str.
```

x	**1**	2	**3**	**4**	5	6	7	8
c	**T**	F	**T**	**T**	F	F	F	F
x when c	**1**	**3**	**4**	ϵ	ϵ	ϵ	ϵ	ϵ

Relational $\dashrightarrow$ Vélus

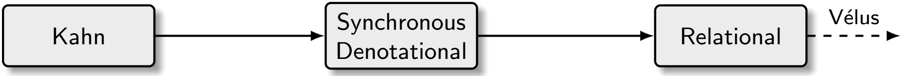
$(\text{sampl } D)^\omega$

```
CoInductive Str {D} :=
| Con : sampl D → Str → Str.
```

x	**1**	2	**3**	**4**	5	6	7	8
c	**T**	F	**T**	**T**	F	F	F	F
x when c	**1**	A	**3**	**4**	A	A	A	A

Integration into the compilation chain

Which type for streams?



$D^* \cup D^\omega$

```
CoInductive Str {D} :=
| Eps : Str → Str
| Con : D → Str → Str.
```

x	1	2	3	4	5	6	7	8
c	T	F	T	T	F	F	F	F
x when c	1	3	4	ϵ	ϵ	ϵ	ϵ	ϵ

$(\text{sampl } D)^* \cup (\text{sampl } D)^\omega$

```
CoInductive Str {D} :=
| Eps : Str → Str
| Con : sampl D → Str → Str.
```

x	1	2	3	4	5	6	7	8
c	T	F	T	T	F	F	F	F
x when c	1	A	3	4	A	ϵ	ϵ	ϵ

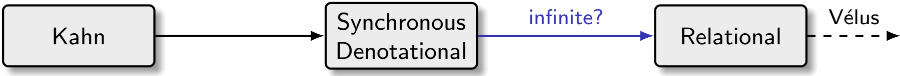
$(\text{sampl } D)^\omega$

```
CoInductive Str {D} :=
| Con : sampl D → Str → Str.
```

x	1	2	3	4	5	6	7	8
c	T	F	T	T	F	F	F	F
x when c	1	A	3	4	A	A	A	A

Integration into the compilation chain

Which type for streams?



$D^* \cup D^\omega$

$(\text{sampl } D)^* \cup (\text{sampl } D)^\omega$

$(\text{sampl } D)^\omega$

```
CoInductive Str {D} :=
| Eps : Str → Str
| Con : D → Str → Str.
```

```
CoInductive Str {D} :=
| Eps : Str → Str
| Con : sampl D → Str → Str.
```

```
CoInductive Str {D} :=
| Con : sampl D → Str → Str.
```

x	1	2	3	4	5	6	7	8
c	T	F	T	T	F	F	F	F
x when c	1	3	4	ϵ	ϵ	ϵ	ϵ	ϵ

x	1	2	3	4	5	6	7	8
c	T	F	T	T	F	F	F	F
x when c	1	A	3	4	A	ϵ	ϵ	ϵ

x	1	2	3	4	5	6	7	8
c	T	F	T	T	F	F	F	F
x when c	1	A	3	4	A	A	A	A

Integration into the compilation chain

Which type for streams?

Kahn → Synchronous Denotational → (infinite?) → Relational - - - (Vélus) →

$D^* \cup D^\omega$ … $\ldots\text{mpl } D)^\omega$

Stream conversion

```
(* infinite number of [Con] *)
Definition tr_Str (s : SD.Str) : infinite s → R.Str.

Conjecture ok_exp_inf :
  ∀ e envI bs env,
    let xs := denot_exp e envI bs env in
    ∀ (xsi : infinite xs), (* to prove! *)
    sem_exp env bs e (tr_Str xs xsi).
```