Analyse de dépendance vérifiée pour un langage synchrone à flot de données

Timothy Bourke[1,2], Basile Pesin[1,2], Marc Pouzet[2,1]

[1] Inria Paris, France

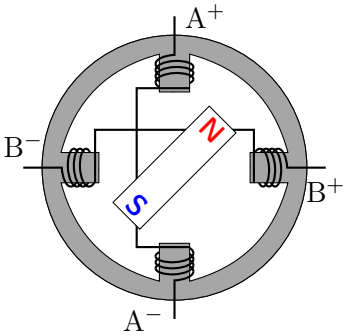
[2] Département d'informatique de l'École normale supérieure, CNRS, PSL University, Paris, France

Journées Francophones des Langages Applicatifs
3 Février 2023

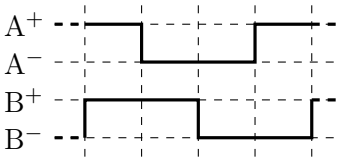
Un petit système embarqué : la porte d'une télécabine

- La porte s'ouvre quand la cabine arrive en station, se ferme quand elle sort
- moteur pas-à-pas : 4 bobines, controllées par `motorA` et `motorB`

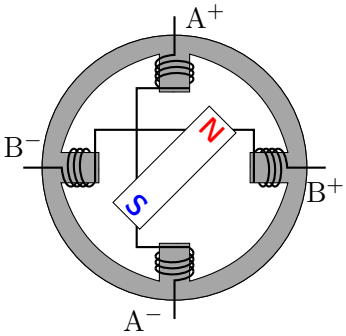
crédit : Téléphérique de Rochebrune



Séquence de contrôle moteur

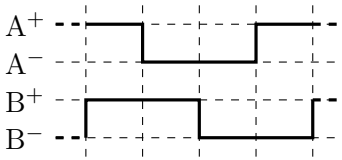


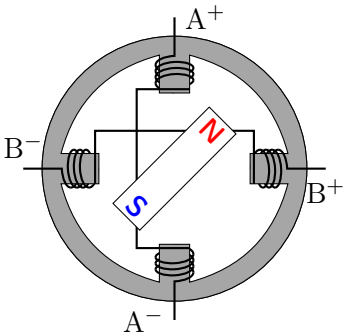

```
moteurA = true fby (not moteurB);
moteurB = true fby moteurA;
```



Séquence de contrôle moteur

```
node sequence_controle()
returns (moteurA, moteurB : bool)
let
    moteurA = true fby (not moteurB);
    moteurB = true fby moteurA;
tel
```





Compilation vérifiée d'un langage synchrone à flots de données

- Vélus : formalisation du langage Lustre, sa sémantique, et sa compilation
  [Bourke, Jeanmaire, Pesin et Pouzet (2021): Verified Lustre Normalization with Node Subsampling]

Compilation vérifiée d'un langage synchrone à flots de données

- Vélus : formalisation du langage Lustre, sa sémantique, et sa compilation

  [Bourke, Jeanmaire, Pesin et Pouzet (2021): Verified Lustre Normalization with Node Subsampling]

- Lucid Synchrone et Scade 6 introduisent des structures d'activations de plus haut niveau

  [Colaço, Pagano et Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines ; Colaço, Hamon et Pouzet (2006): Mixing Signals and Modes in Synchronous Data-flow Systems ; Colaço, Pagano et Pouzet (2017): Scade 6: A Formal Language for Embedded Critical Software Development]

Compilation vérifiée d'un langage synchrone à flots de données

- Vélus : formalisation du langage Lustre, sa sémantique, et sa compilation

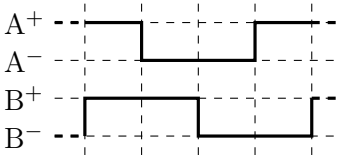
  [Bourke, Jeanmaire, Pesin et Pouzet (2021): Verified Lustre Normalization with Node Subsampling]

- Lucid Synchrone et Scade 6 introduisent des structures d'activations de plus haut niveau

  [Colaço, Pagano et Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines ; Colaço, Hamon et Pouzet (2006): Mixing Signals and Modes in Synchronous Data-flow Systems ; Colaço, Pagano et Pouzet (2017): Scade 6: A Formal Language for Embedded Critical Software Development]

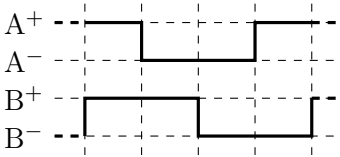
- extension de Vélus avec certaines de ces constructions : `switch` et `last`
- extension de l'analyse de dépendance de Vélus pour traiter ces nouvelles constructions
- utilisation de l'analyse pour prouver des propriétés du modèle sémantique

Implémentation du contrôle moteur



Implémentation du contrôle moteur

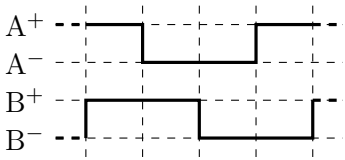
```
moteurA = true fby (not moteurB);
moteurB = true fby moteurA;
```



Implémentation du contrôle moteur

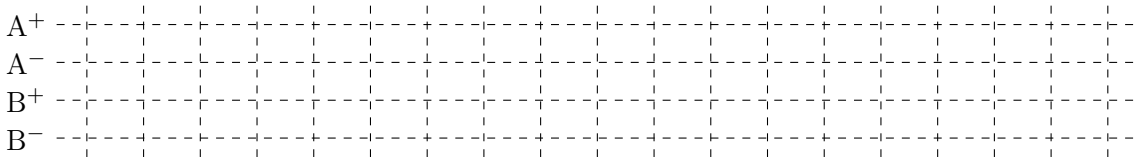
```
type moteurDir = Ouvre | Ferme | Bloque

node control_moteur(dir : moteurDir) returns (moteurA, moteurB : bool)
let
  switch dir
  | Ouvre do
    moteurA = true fby (not moteurB);
    moteurB = true fby moteurA;
  | Ferme do
    moteurA = true fby moteurB;
    moteurB = true fby (not moteurA);
  | Bloque do
    moteurA = true fby moteurA;
    moteurB = true fby moteurB;
  end;
tel
```



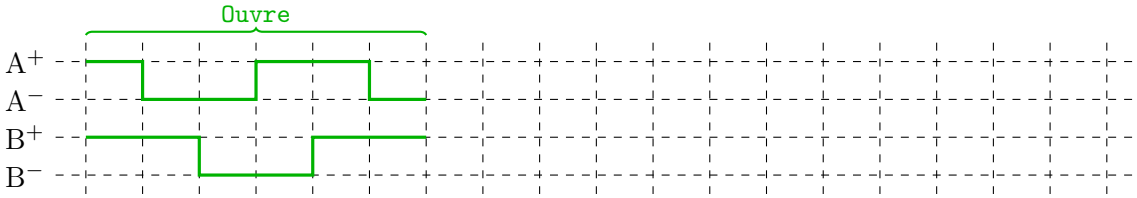
Implémentation du contrôle moteur — un défaut

```
node control_moteur(dir : moteurDir) returns (moteurA, moteurB : bool)
let
  switch dir
  | Ouvre do
    moteurA = true fby (not moteurB);
    moteurB = true fby moteurA;
  | Ferme do
    moteurA = true fby moteurB;
    moteurB = true fby (not moteurA);
  | Bloque do
    moteurA = true fby moteurA;
    moteurB = true fby moteurB;
  end;
tel
```



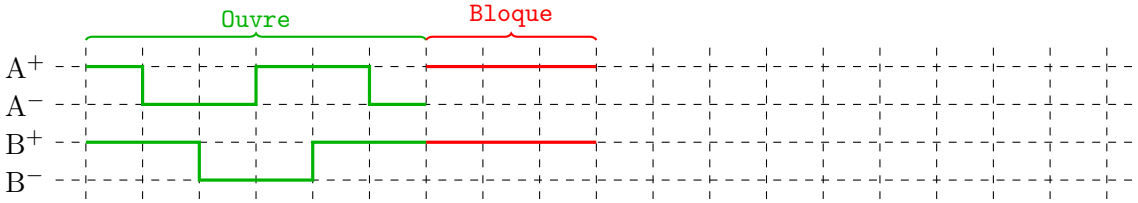
Implémentation du contrôle moteur — un défaut

```
node control_moteur(dir : moteurDir) returns (moteurA, moteurB : bool)
let
  switch dir
  | Ouvre do
    moteurA = true fby (not moteurB);
    moteurB = true fby moteurA;
  | Ferme do
    moteurA = true fby moteurB;
    moteurB = true fby (not moteurA);
  | Bloque do
    moteurA = true fby moteurA;
    moteurB = true fby moteurB;
  end;
tel
```



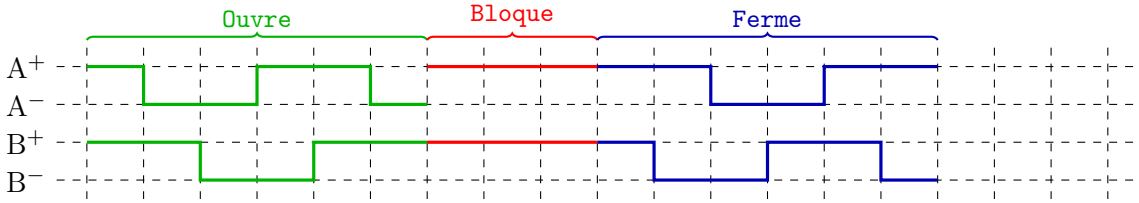
Implémentation du contrôle moteur — un défaut

```
node control_moteur(dir : moteurDir) returns (moteurA, moteurB : bool)
let
  switch dir
  | Ouvre do
    moteurA = true fby (not moteurB);
    moteurB = true fby moteurA;
  | Ferme do
    moteurA = true fby moteurB;
    moteurB = true fby (not moteurA);
  | Bloque do
    moteurA = true fby moteurA;
    moteurB = true fby moteurB;
  end;
tel
```



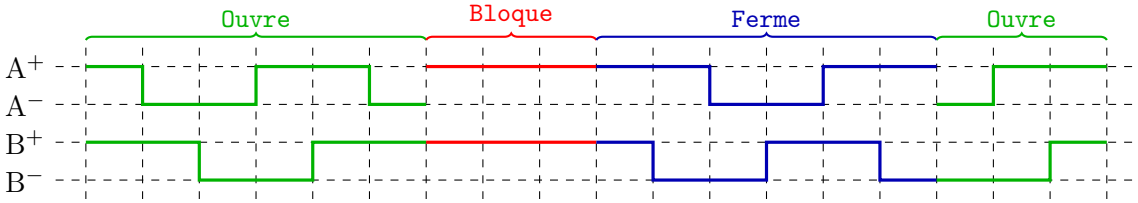
Implémentation du contrôle moteur — un défaut

```
node control_moteur(dir : moteurDir) returns (moteurA, moteurB : bool)
let
  switch dir
  | Ouvre do
    moteurA = true fby (not moteurB);
    moteurB = true fby moteurA;
  | Ferme do
    moteurA = true fby moteurB;
    moteurB = true fby (not moteurA);
  | Bloque do
    moteurA = true fby moteurA;
    moteurB = true fby moteurB;
  end;
tel
```



Implémentation du contrôle moteur — un défaut

```
node control_moteur(dir : moteurDir) returns (moteurA, moteurB : bool)
let
  switch dir
  | Ouvre do
    moteurA = true fby (not moteurB);
    moteurB = true fby moteurA;
  | Ferme do
    moteurA = true fby moteurB;
    moteurB = true fby (not moteurA);
  | Bloque do
    moteurA = true fby moteurA;
    moteurB = true fby moteurB;
  end;
tel
```

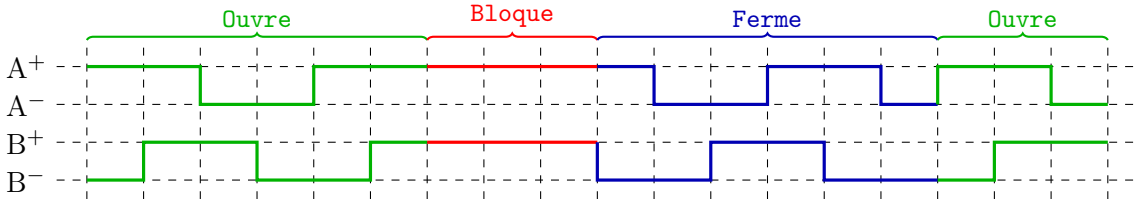


Implémentation du contrôle moteur — version 2

```
node control_moteur(dir : moteurDir) returns (last moteurA : bool = true; last moteurB : bool = true)
let
  switch dir
  | Ouvre do
    moteurA = not last moteurB;
    moteurB = last moteurA;
  | Ferme do
    moteurA = last moteurB;
    moteurB = not last moteurA;
  | Bloque do
    moteurA = last moteurA;
    moteurB = last moteurB;
  end;
tel
```

Implémentation du contrôle moteur — version 2

```
node control_moteur(dir : moteurDir) returns (last moteurA : bool = true; last moteurB : bool = true)
let
  switch dir
  | Ouvre do
    moteurA = not last moteurB;
    moteurB = last moteurA;
  | Ferme do
    moteurA = last moteurB;
    moteurB = not last moteurA;
  | Bloque do
    moteurA = last moteurA;
    moteurB = last moteurB;
  end;
tel
```



Sémantique du noyau flots de données de Vélus

dir	O	O	O	O	O	O	B	B	B	F	F	F	F	F	F	O	O	O	...
moteurA	1	1	0	0	1	1	1	1	1	1	0	0	1	1	0	1	1	0	...
moteurB	0	1	1	0	0	1	1	1	1	0	0	1	1	0	0	0	1	1	...

Sémantique du noyau flots de données de Vélus

dir	O	O	O	O	O	O	B	B	B	F	F	F	F	F	F	O	O	O	...
moteurA	1	1	0	0	1	1	1	1	1	1	0	0	1	1	0	1	1	0	...
moteurB	0	1	1	0	0	1	1	1	1	0	0	1	1	0	0	0	1	1	...

$$\frac{H(x) \equiv vs}{G, H \vdash x \Downarrow [vs]}$$

Sémantique du noyau flots de données de Vélus

dir	O	O	O	O	O	O	B	B	B	F	F	F	F	F	F	O	O	O	...
moteurA	1	1	0	0	1	1	1	1	1	1	0	0	1	1	0	1	1	0	...
moteurB	0	1	1	0	0	1	1	1	1	0	0	1	1	0	0	0	1	1	...

$$\frac{H(x) \equiv vs}{G, H \vdash x \Downarrow [vs]} \qquad \frac{\forall i \in 1...n,\ H(x_i) \equiv vs_i \qquad G, H \vdash es \Downarrow (vs_1, \ldots, vs_n)}{G, H \vdash (x_1, \ldots, x_n) = es}$$

Sémantique du noyau flots de données de Vélus

dir	O	O	O	O	O	O	B	B	B	F	F	F	F	F	F	O	O	O	...
moteurA	1	1	0	0	1	1	1	1	1	1	0	0	1	1	0	1	1	0	...
moteurB	0	1	1	0	0	1	1	1	1	0	0	1	1	0	0	0	1	1	...

$$\frac{H(x) \equiv vs}{G, H \vdash x \Downarrow [vs]} \qquad \frac{\forall i \in 1...n,\ H(x_i) \equiv vs_i \qquad G, H \vdash es \Downarrow (vs_1, \ldots, vs_n)}{G, H \vdash (x_1, \ldots, x_n) = es}$$

$$\frac{\begin{array}{c} G(f) = \texttt{node}\ f(x_1, \ldots, x_n)\ \texttt{returns}\ (y_1, \ldots, y_m)\ blk \\ \forall i \in 1...n,\ H(x_i) \equiv xs_i \qquad \forall i \in 1...m,\ H(y_i) \equiv ys_i \qquad G, H \vdash blk \end{array}}{G \vdash f(xs_1, \ldots, xs_n) \Downarrow (ys_1, \ldots, ys_m)}$$

Sémantique de switch

```
node control_moteur(dir : moteurDir) returns (last moteurA : bool = true; last moteurB : bool = true)
let
  switch dir
  | Ouvre do
    moteurA = not last moteurB;
    moteurB = last moteurA;
  | Ferme do
    moteurA = last moteurB;
    moteurB = not last moteurA;
  | Bloque do
    moteurA = last moteurA;
    moteurB = last moteurB;
  end;
tel
```

Sémantique de `switch`

```
node control_moteur(dir : moteurDir) returns (last moteurA : bool = true; last moteurB : bool = true)
let
  switch dir
  | Ouvre do
    moteurA = not last moteurB;
    moteurB = last moteurA;
  | Ferme do
    moteurA = last moteurB;
    moteurB = not last moteurA;
  | Bloque do
    moteurA = last moteurA;
    moteurB = last moteurB;
  end;
tel
```

$$\begin{aligned}
\text{when}^{C}\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^{C}\ xs\ ys \\
\text{when}^{C}\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) &\equiv \langle v\rangle \cdot \text{when}^{C}\ xs\ ys \\
\text{when}^{C}\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^{C}\ xs\ ys
\end{aligned}$$

dir	...
when^{Ouvre} *motA dir*	...
when^{Ferme} *motA dir*	...
when^{Bloque} *motA dir*	...
motA	...

Sémantique de `switch`

```
node control_moteur(dir : moteurDir) returns (last moteurA : bool = true; last moteurB : bool = true)
let
  switch dir
  | Ouvre do
    moteurA = not last moteurB;
    moteurB = last moteurA;
  | Ferme do
    moteurA = last moteurB;
    moteurB = not last moteurA;
  | Bloque do
    moteurA = last moteurA;
    moteurB = last moteurB;
  end;
tel
```

$$
\begin{aligned}
\text{when}^{C}\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^{C}\ xs\ ys \\
\text{when}^{C}\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) &\equiv \langle v\rangle \cdot \text{when}^{C}\ xs\ ys \\
\text{when}^{C}\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^{C}\ xs\ ys
\end{aligned}
$$

dir	O	O	O	O	O	O	1	...
when^{Ouvre} *motA dir*	1	1	0	0	1	1	0	...
when^{Ferme} *motA dir*								...
when^{Bloque} *motA dir*								...
motA	1	1	0	0	1	1	1	...

Sémantique de `switch`

```
node control_moteur(dir : moteurDir) returns (last moteurA : bool = true; last moteurB : bool = true)
let
  switch dir
  | Ouvre do
    moteurA = not last moteurB;
    moteurB = last moteurA;
  | Ferme do
    moteurA = last moteurB;
    moteurB = not last moteurA;
  | Bloque do
    moteurA = last moteurA;
    moteurB = last moteurB;
  end;
tel
```

$$\begin{aligned}
\text{when}^{C}\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^{C}\ xs\ ys \\
\text{when}^{C}\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) &\equiv \langle v\rangle \cdot \text{when}^{C}\ xs\ ys \\
\text{when}^{C}\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^{C}\ xs\ ys
\end{aligned}$$

dir	B	B	B	...
when^{Ouvre} *motA dir*				...
when^{Ferme} *motA dir*				...
when^{Bloque} *motA dir*	1	1	1	...
motA	1	1	1	...

Sémantique de switch

```
node control_moteur(dir : moteurDir) returns (last moteurA : bool = true; last moteurB : bool = true)
let
  switch dir
  | Ouvre do
    moteurA = not last moteurB;
    moteurB = last moteurA;
  | Ferme do
    moteurA = last moteurB;
    moteurB = not last moteurA;
  | Bloque do
    moteurA = last moteurA;
    moteurB = last moteurB;
  end;
tel
```

$$
\begin{aligned}
\text{when}^{C}\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^{C}\ xs\ ys \\
\text{when}^{C}\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) &\equiv \langle v\rangle \cdot \text{when}^{C}\ xs\ ys \\
\text{when}^{C}\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^{C}\ xs\ ys
\end{aligned}
$$

dir	F	F	F	F	F	F	...
when^{Ouvre} *motA dir*							...
when^{Ferme} *motA dir*	1	0	0	1	1	0	...
when^{Bloque} *motA dir*							...
motA	1	0	0	1	1	0	...

Sémantique de `switch`

```
node control_moteur(dir : moteurDir) returns (last moteurA : bool = true; last moteurB : bool = true)
let
  switch dir
  | Ouvre do
    moteurA = not last moteurB;
    moteurB = last moteurA;
  | Ferme do
    moteurA = last moteurB;
    moteurB = not last moteurA;
  | Bloque do
    moteurA = last moteurA;
    moteurB = last moteurB;
  end;
tel
```

$$\begin{aligned}
\text{when}^C\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) &\equiv \langle v\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^C\ xs\ ys
\end{aligned}$$

dir	O	O	O	O	O	O	B	B	B	F	F	F	F	F	F	1	...
when^{Ouvre} *motA dir*	1	1	0	0	1	1	‹›	‹›	‹›	‹›	‹›	‹›	‹›	‹›	‹›	0	...
when^{Ferme} *motA dir*	‹›	‹›	‹›	‹›	‹›	‹›	‹›	‹›	‹›	1	0	0	1	1	0	‹›	...
when^{Bloque} *motA dir*	‹›	‹›	‹›	‹›	‹›	‹›	1	1	1	‹›	‹›	‹›	‹›	‹›	‹›	‹›	...
motA	1	1	0	0	1	1	1	1	1	1	0	0	1	1	0	1	...

Sémantique de `switch`

```
node control_moteur(dir : moteurDir) returns (last moteurA : bool = true; last moteurB : bool = true)
let
  switch dir
  | Ouvre do
    moteurA = not last moteurB;
    moteurB = last moteurA;
  | Ferme do
    moteurA = last moteurB;
    moteurB = not last moteurA;
  | Bloque do
    moteurA = last moteurA;
    moteurB = last moteurB;
  end;
tel
```

$$\begin{aligned}
\text{when}^C\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) &\equiv \langle v\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^C\ xs\ ys
\end{aligned}$$

$$\frac{G, H \vdash e \Downarrow [cs] \qquad \forall i,\ G, \text{when}^{C_i}\ H\ cs \vdash blks_i}{G, H \vdash \texttt{switch}\ e\ \overline{(C_i\ \texttt{do}\ blks_i)}^{\,i}\ \texttt{end}}$$

dir	O	O	O	O	O	O	B	B	B	F	F	F	F	F	F	1	...
when^{Ouvre} *motA dir*	1	1	0	0	1	1	‹›	‹›	‹›	‹›	‹›	‹›	‹›	‹›	‹›	0	...
when^{Ferme} *motA dir*	‹›	‹›	‹›	‹›	‹›	‹›	‹›	‹›	‹›	1	0	0	1	1	0	‹›	...
when^{Bloque} *motA dir*	‹›	‹›	‹›	‹›	‹›	‹›	1	1	1	‹›	‹›	‹›	‹›	‹›	‹›	‹›	...
motA	1	1	0	0	1	1	1	1	1	1	0	0	1	1	0	1	...

Sémantique de `last`

$$\frac{\forall x,\ x \in \mathrm{dom}(H') \iff x \in \mathit{locs} \qquad G, H + H' \vdash \mathit{blks}}{G, H \vdash \texttt{var}\ \mathit{locs}\ \texttt{let}\ \mathit{blks}\ \texttt{tel}}$$

$$(H_1 + H_2)(x) = \begin{cases} H_2(x) & \text{if } x \in H_2 \\ H_1(x) & \text{otherwise.} \end{cases}$$

Sémantique de `last`

$$\frac{\forall x,\ x \in \mathrm{dom}(H') \iff x \in \mathit{locs} \qquad \forall x\ e,\ (\texttt{last}\ x = e) \in \mathit{locs} \implies G, H + H' \vdash \texttt{last}\ x = e \qquad G, H + H' \vdash \mathit{blks}}{G, H \vdash \texttt{var}\ \mathit{locs}\ \texttt{let}\ \mathit{blks}\ \texttt{tel}}$$

$$(H_1 + H_2)(x) = \begin{cases} H_2(x) & \text{if } x \in H_2 \\ H_1(x) & \text{otherwise.} \end{cases}$$

$$\frac{H(x) \equiv vs}{G, H \vdash x \Downarrow [vs]} \qquad \frac{H(\mathrm{last}\ x) \equiv vs}{G, H \vdash \texttt{last}\ x \Downarrow [vs]}$$

$$\frac{G, H + H' \vdash e \Downarrow [vs_0] \qquad H'(x) \equiv vs_1 \qquad H'(\mathrm{last}\ x) \equiv \mathrm{fby}\ vs_0\ vs_1}{G, H + H' \vdash \texttt{last}\ x = e}$$

Analyse de dépendance

Considérons les définitions suivantes :

Analyse de dépendance

Considérons les définitions suivantes :

- `x = x;` accepte n'importe quelle valeur

Analyse de dépendance

Considérons les définitions suivantes :

- `x = x;` accepte n'importe quelle valeur
- `x = x + 1;` n'accepte aucune valeur

Analyse de dépendance

Considérons les définitions suivantes :

- `x = x;` accepte n'importe quelle valeur
- `x = x + 1;` n'accepte aucune valeur
- `x = f(x)` nécessite un schéma de compilation fin, et une analyse modulaire

  [Cuoq et Pouzet (2001): Modular Causality in a Synchronous Stream Language]

Ces programmes sont rejetés par analyse statique. Dans Vélus, on analyse le graphe de dépendance de chaque nœud [Caspi, Pilaud, Halbwachs et Plaice (1987): LUSTRE: A declarative language for programming synchronous systems]

Analyse de dépendance

Considérons les définitions suivantes :

- `x = x;` accepte n'importe quelle valeur
- `x = x + 1;` n'accepte aucune valeur
- `x = f(x)` nécessite un schéma de compilation fin, et une analyse modulaire
  [Cuoq et Pouzet (2001): Modular Causality in a Synchronous Stream Language]

Ces programmes sont rejetés par analyse statique. Dans Vélus, on analyse le graphe de dépendance de chaque nœud [Caspi, Pilaud, Halbwachs et Plaice (1987): LUSTRE: A declarative language for programming synchronous systems]

On étend l'analyse de dépendance de Vélus :

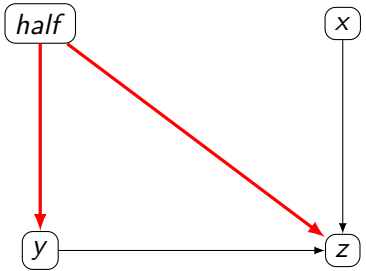
- Traitement des structures d'activations avec des étiquettes
- Algorithme d'analyse de graphe (similaire à [Pottier (2015): Depth-First Search and Strong Connectivity in Coq])
- On l'utilise pour prouver des propriétés de la sémantique

Analyse du noyau flots de données

```
node f(x : int) returns (y, z : int)
var half : bool;
let
    half = true fby (not half);
    (y, z) = if half then (0, x) else (1, y);
tel
```

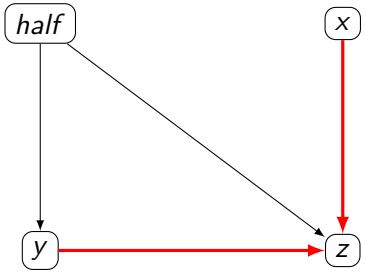
Analyse du noyau flots de données

```
node f(x : int) returns (y, z : int)
var half : bool;
let
    half = true fby (not half);
    (y, z) = if half then (0, x) else (1, y);
tel
```



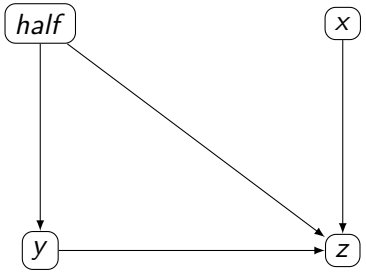
Analyse du noyau flots de données

```
node f(x : int) returns (y, z : int)
var half : bool;
let
    half = true fby (not half);
    (y, z) = if half then (0, x) else (1, y);
tel
```



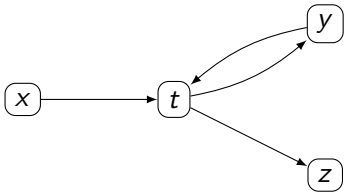
Analyse du noyau flots de données

```
node f(x : int) returns (y, z : int)
var half : bool;
let
    half = true fby (not half);
    (y, z) = if half then (0, x) else (1, y);
tel
```



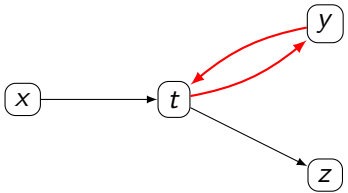
Dépendances induites par les déclarations locales

```
node f(x : int) returns (z : bool)
var y : int;
let
  var t : int;
  let t = x fby (t + 1);
      y = t;
  tel;
  var t : int;
  let t = y + 1;
      z = t > 0;
  tel
tel
```



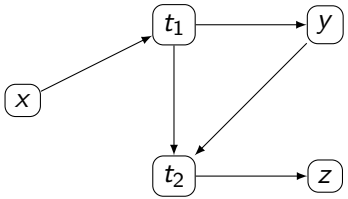
Dépendances induites par les déclarations locales

```
node f(x : int) returns (z : bool)
var y : int;
let
  var t : int;
  let t = x fby (t + 1);
      y = t;
  tel;
  var t : int;
  let t = y + 1;
      z = t > 0;
  tel
tel
```



Dépendances induites par les déclarations locales

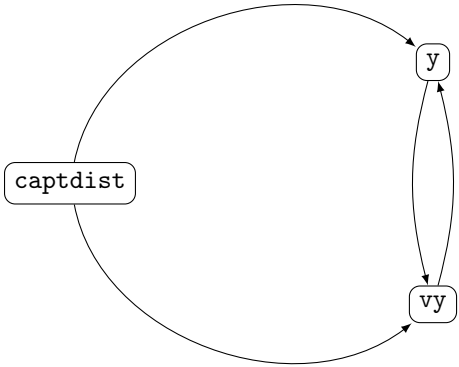
```
node f(x_x1 : int) returns (z_z1 : bool)
var y_y1 : int;
let
  var t_t1 : int;
  let t_t1 = x_x1 fby (t_t1 + 1);
      y_y1 = t_t1;
  tel;
  var t_t2 : int;
  let t_t2 = y_y1 + 1;
      z_z1 = t_t2 > 0;
  tel
tel
```



On associe une étiquette unique à chaque déclaration

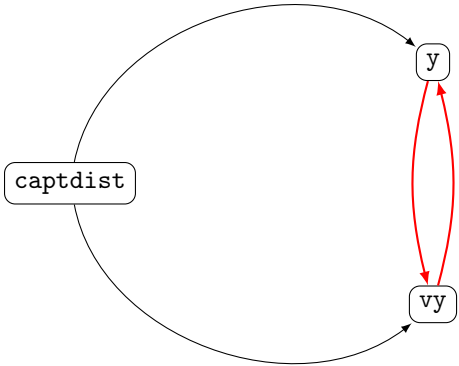
Dépendances induites par les switch

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y = captdist;
    vy = (y - last y) / dt;
  | false do
    vy = last vy + accely * dt;
    y = last y + vy * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



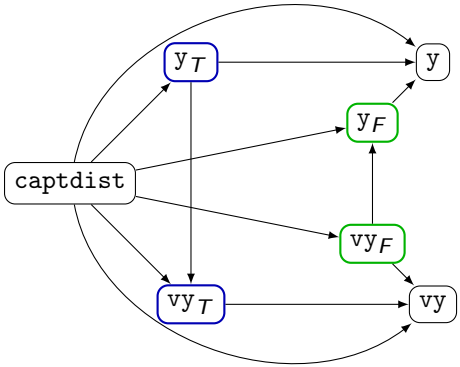
Dépendances induites par les switch

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y = captdist;
    vy = (y - last y) / dt;
  | false do
    vy = last vy + accely * dt;
    y = last y + vy * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



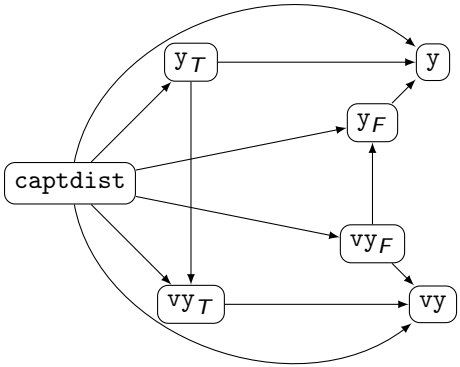
Dépendances induites par les switch

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y) / dt;
  | false do
    vy_F = last vy + accely * dt;
    y_F = last y + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



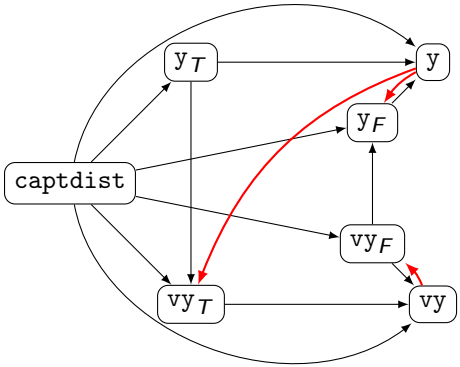
Dépendances induites par les `last`

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y) / dt;
  | false do
    vy_F = last vy + accely * dt;
    y_F = last y + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



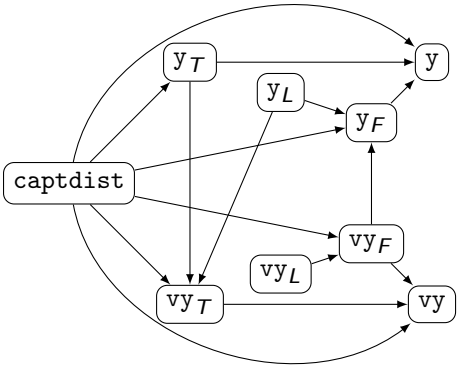
Dépendances induites par les `last`

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y) / dt;
  | false do
    vy_F = last vy + accely * dt;
    y_F = last y + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



Dépendances induites par les `last`

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



Analyse du graphe de dépendance

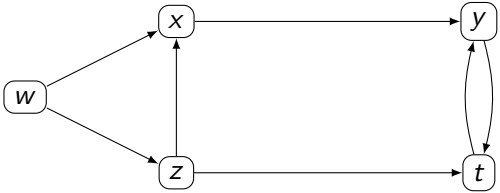
$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset}$$

$$\frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E}$$

$$\frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$

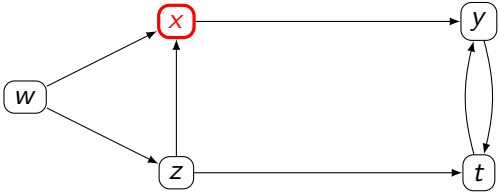
Analyse du graphe de dépendance

$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset} \qquad \frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E} \qquad \frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$



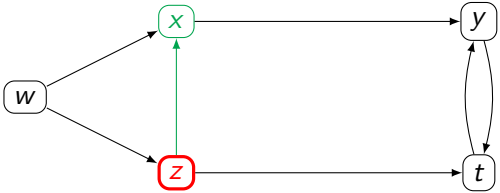
Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



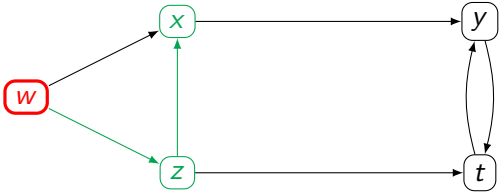
Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



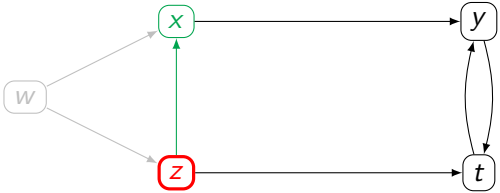
Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



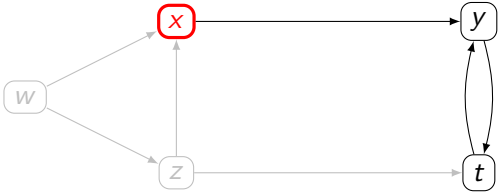
Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



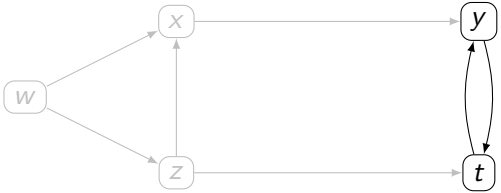
Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



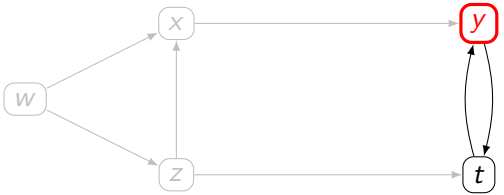
Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



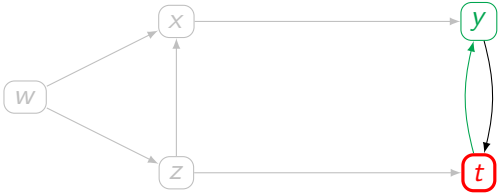
Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



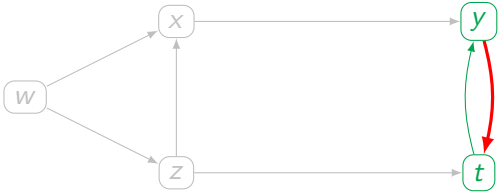
Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



Analyse du graphe de dépendance

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$

```
Definition visited (p : set) (v : set) : Prop :=
    (∀ x, x ∈ p → ¬(x ∈ v))
  ∧ ∃ a, AcyGraph v a
        ∧ (∀ x, x ∈ v → ∃ zs, graph(x) = Some zs
                            ∧ (∀ y, y ∈ zs → has_arc a y x)).

Program Fixpoint dfs'
  (s : { p | ∀ x, x ∈ p → x ∈ graph }) (x : ident)
  (v : { v | visited s v }) {measure (|graph| - |s|)}
  : option { v' | visited s v' & x ∈ v' ∧ v ⊆ v' } := ...
```

Analyse du graphe de dépendance

$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset} \qquad \frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E} \qquad \frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$

```
Definition visited (p : set) (v : set) : Prop :=
    (∀ x, x ∈ p → ¬(x ∈ v))
  ∧ ∃ a, AcyGraph v a
        ∧ (∀ x, x ∈ v → ∃ zs, graph(x) = Some zs
                              ∧ (∀ y, y ∈ zs → has_arc a y x)).

Program Fixpoint dfs'
  (s : { p | ∀ x, x ∈ p → x ∈ graph }) (x : ident)
  (v : { v | visited s v }) {measure (|graph| - |s|)}
  : option { v' | visited s v' & x ∈ v' ∧ v ⊆ v' } := ...
```

Analyse du graphe de dépendance

$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset} \qquad \frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E} \qquad \frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$

```
Definition visited (p : set) (v : set) : Prop :=
    (∀ x, x ∈ p → ¬(x ∈ v))
  ∧ ∃ a, AcyGraph v a
        ∧ (∀ x, x ∈ v → ∃ zs, graph(x) = Some zs
                              ∧ (∀ y, y ∈ zs → has_arc a y x)).

Program Fixpoint dfs'
  (s : { p | ∀ x, x ∈ p → x ∈ graph }) (x : ident)
  (v : { v | visited s v }) {measure (|graph| - |s|)}
  : option { v' | visited s v' & x ∈ v' ∧ v ⊆ v' } := ...
```

Analyse du graphe de dépendance

$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset} \qquad \frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E} \qquad \frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$

```
Definition visited (p : set) (v : set) : Prop :=
    (∀ x, x ∈ p → ¬(x ∈ v))
  ∧ ∃ a, AcyGraph v a
        ∧ (∀ x, x ∈ v → ∃ zs, graph(x) = Some zs
                              ∧ (∀ y, y ∈ zs → has_arc a y x)).

Program Fixpoint dfs'
  (s : { p | ∀ x, x ∈ p → x ∈ graph }) (x : ident)
  (v : { v | visited s v }) {measure (|graph| - |s|)}
  : option { v' | visited s v' & x ∈ v' ∧ v ⊆ v' } := ...
```

Preuves sur un programme causal

$$\frac{}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ []}$$

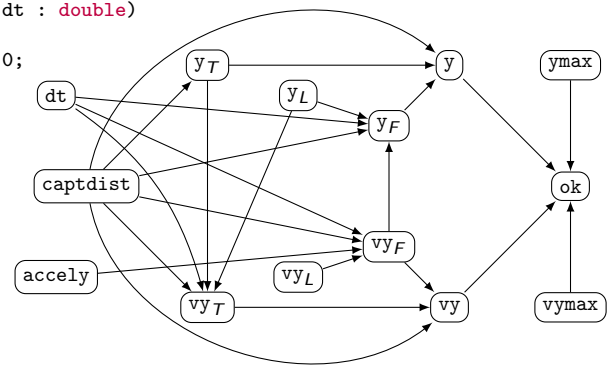
$$\frac{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \mathsf{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \mathsf{In}\ y\ l)}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ (x :: l)}$$

Preuves sur un programme causal

$$\frac{}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ []}$$

$$\frac{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \mathsf{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \mathsf{In}\ y\ l)}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```

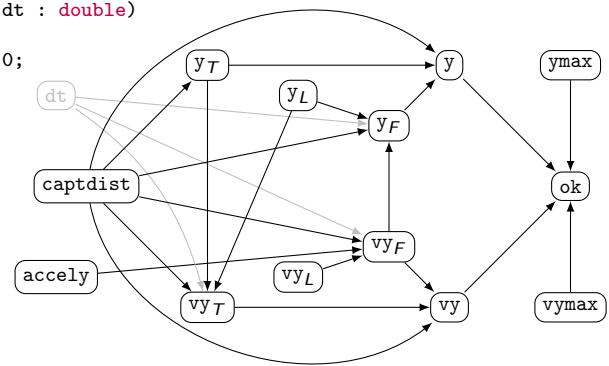


Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```

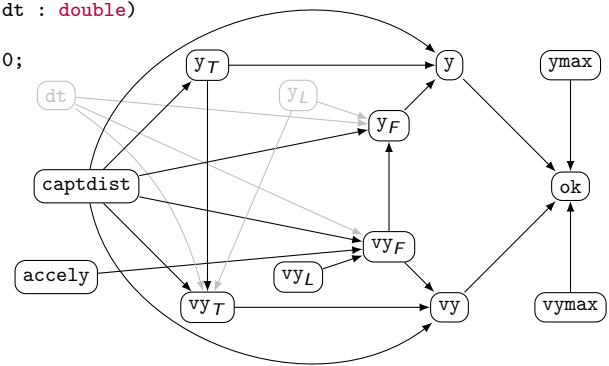


Preuves sur un programme causal

$$\frac{}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ []}$$

$$\frac{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \mathsf{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \mathsf{In}\ y\ l)}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```

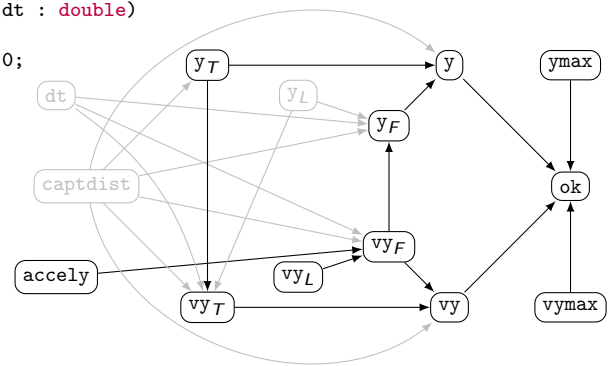


Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



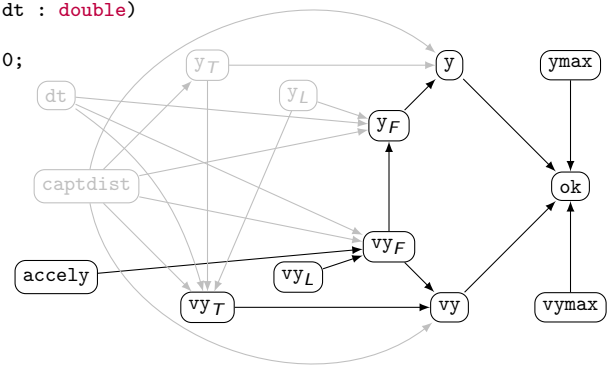
(dt) (y_L) (captdist)

Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



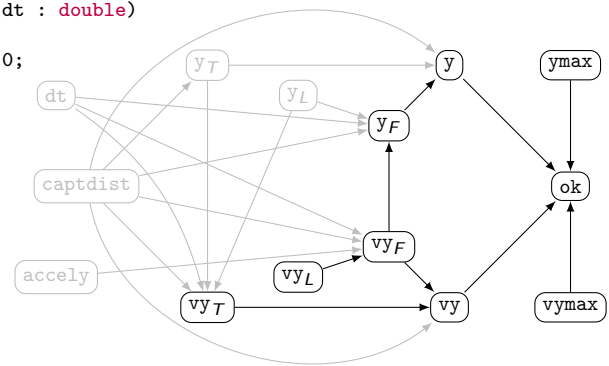
(dt) (y_L) (captdist) (y_T)

Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



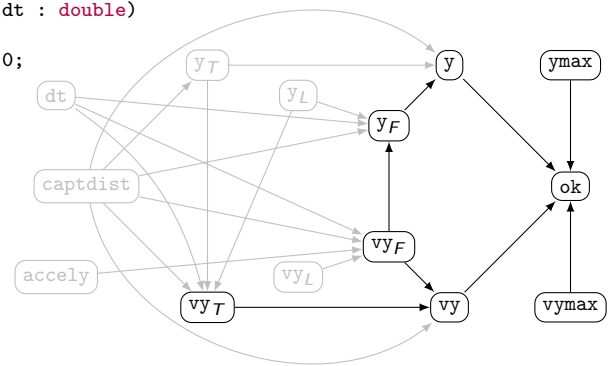
dt y_L captdist y_T accely

Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



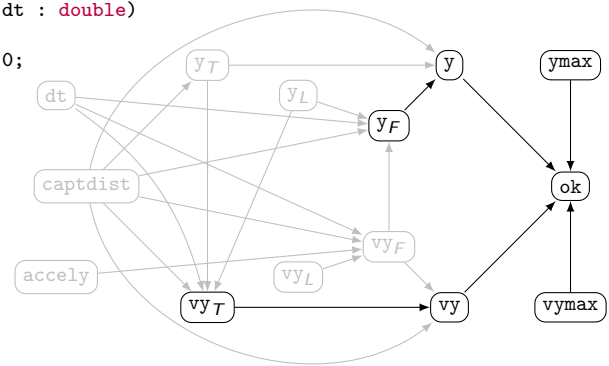
dt y_L captdist y_T accely vy_L

Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



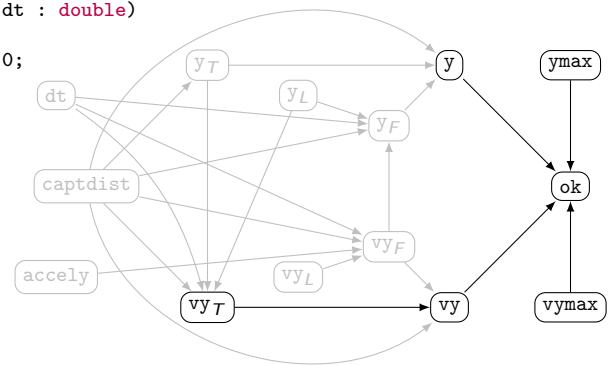
dt y_L captdist y_T accely vy_L vy_F

Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```

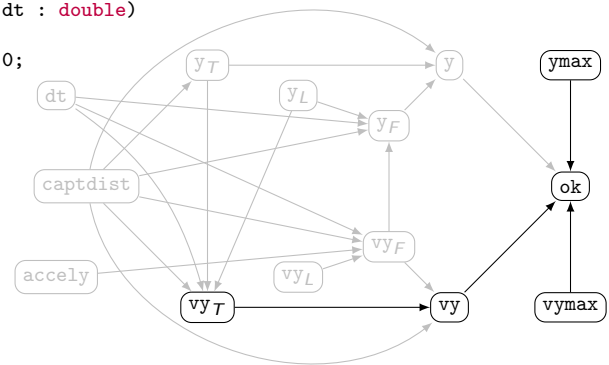


Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



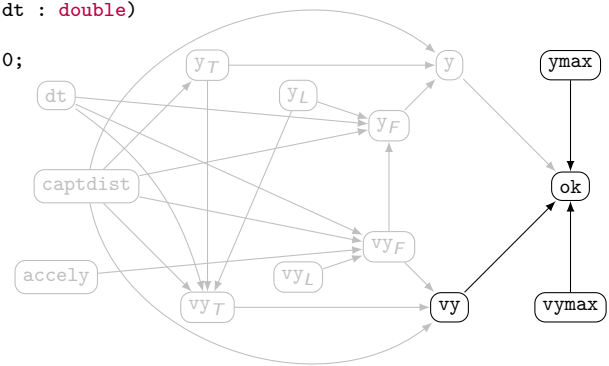
dt y_L captdist y_T accely vy_L vy_F y_F y

Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



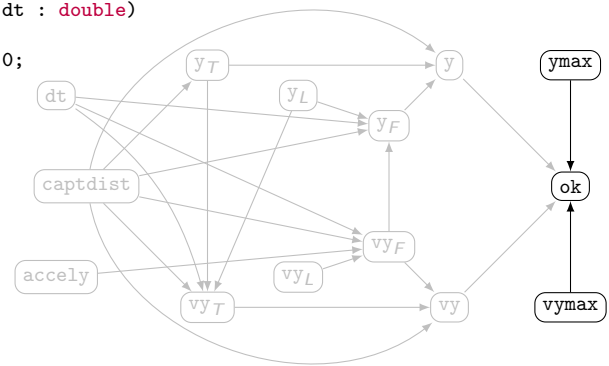
dt y_L captdist y_T accely vy_L vy_F y_F y vy_T

Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```

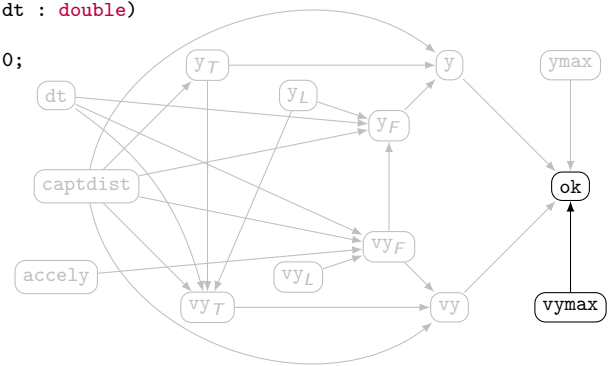


Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```

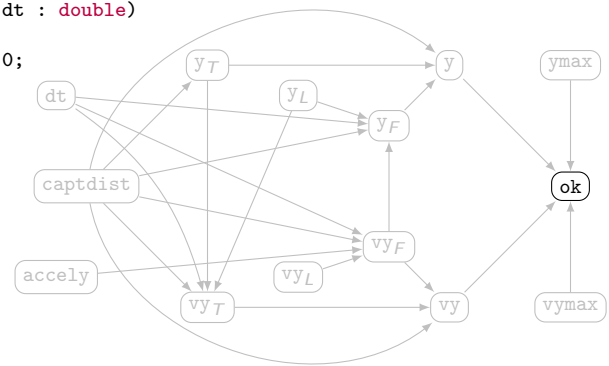


Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```

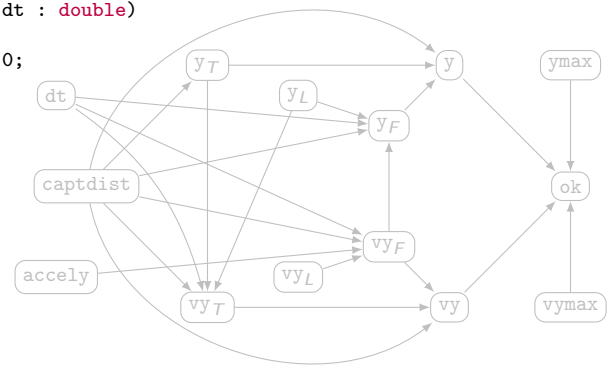


Preuves sur un programme causal

$$\frac{}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ []}$$

$$\frac{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \mathsf{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \mathsf{In}\ y\ l)}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node ouvrir_ok(y_max, vy_max, captdist, accely, dt : double)
returns (ok : bool)
var last vy : double = 0.0; last y : double = 0.0;
let
  switch (captdist > 0.0)
  | true do
    y_T = captdist;
    vy_T = (y_T - last y_L) / dt;
  | false do
    vy_F = last vy_L + accely * dt;
    y_F = last y_L + vy_F * dt;
  end;
  ok = y < y_max and vy < vy_max;
tel
```



Preuves sur un programme causal

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []} \qquad \frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

Utilisée pour prouver :

- Le déterminisme du modèle sémantique :
  if $G \vdash f(xs) \Downarrow ys_1$ **and** $G \vdash f(xs) \Downarrow ys_2$ **then** $ys_1 \equiv ys_2$
- La correction du système d'horloges :
  if $\Gamma \vdash e : ck$ **and** $G, H, bs \vdash e \Downarrow vs$ **then** $H, bs \vdash ck \Downarrow (\text{abstract-clock}\ vs)$

Conclusion

Dans cet article, nous présentons :

- l'ajout de deux constructions de Scade à Vélus (`switch` et `last`)
- la formalisation d'un algorithme d'analyse de graphe classique et de sa correction
- la construction de schémas de preuves pour les programmes sans cycles

Conclusion

Dans cet article, nous présentons :

- l'ajout de deux constructions de Scade à Vélus (`switch` et `last`)
- la formalisation d'un algorithme d'analyse de graphe classique et de sa correction
- la construction de schémas de preuves pour les programmes sans cycles

Ces idées :

- s'adaptent bien aux machines à états de Scade
- pourraient être généralisées à d'autres langages à flots de données

Un prototype de Vélus est disponible : `https://velus.inria.fr/jfla2023/`

Bibliographie I

Bourke, T., P. Jeanmaire, B. Pesin et M. Pouzet (oct. 2021). "Verified Lustre Normalization with Node Subsampling". In : ACM Trans. Embedded Computing Systems 20.5s, Article 98.

Caspi, P., D. Pilaud, N. Halbwachs et J. A. Plaice (jan. 1987). "LUSTRE: A declarative language for programming synchronous systems". In : Proc. 14th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL 1987). Munich, Germany : ACM Press, p. 178-188.

Colaço, J.-L., G. Hamon et M. Pouzet (oct. 2006). "Mixing Signals and Modes in Synchronous Data-flow Systems". In : Proc. 6th ACM Int. Conf. on Embedded Software (EMSOFT 2006). Sous la dir. de S. L. Min et Y. Wang. Seoul, South Korea : ACM Press, p. 73-82.

Colaço, J.-L., B. Pagano et M. Pouzet (sept. 2005). "A Conservative Extension of Synchronous Data-flow with State Machines". In : Proc. 5th ACM Int. Conf. on Embedded Software (EMSOFT 2005). Sous la dir. de W. Wolf. Jersey City, USA : ACM Press, p. 173-182.

— (sept. 2017). "Scade 6: A Formal Language for Embedded Critical Software Development". In : Proc. 11th Int. Symp. Theoretical Aspects of Software Engineering (TASE 2017). Nice, France : IEEE Computer Society, p. 4-15.

Bibliographie II

Cuoq, P. et M. Pouzet (avr. 2001). "Modular Causality in a Synchronous Stream Language". In : 10th European Symposium on Programming (ESOP 2001), part of European Joint Conferences on Theory a Sous la dir. de D. Sands. T. 2028. LNCS. Genova, Italy : Springer, p. 237-251.

Pottier, F. (jan. 2015). "Depth-First Search and Strong Connectivity in Coq". In : Journées Francophones des Langages Applicatifs (JFLA).