Vérification formelle d'une méthodologie pour la conception et la production de systèmes numériques critiques

Soutenance de thèse

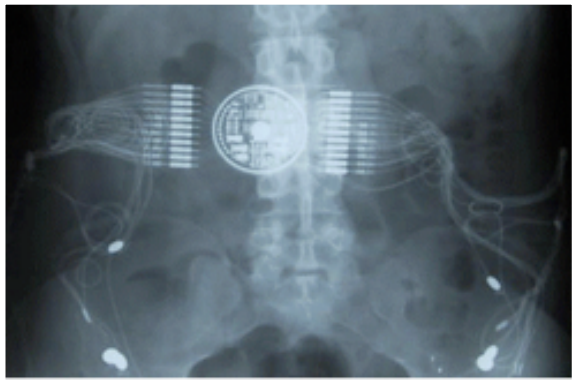
Présentée par :
Vincent Iampietro

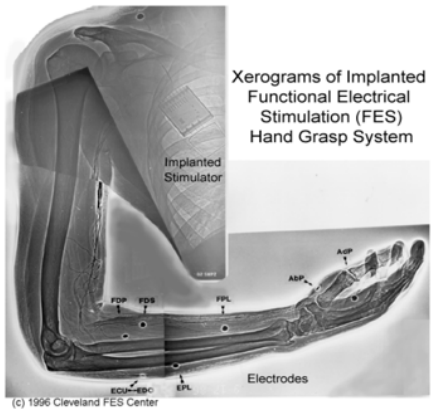
Sous la direction de :
David Andreu, David Delahaye

16 décembre 2021

Plan

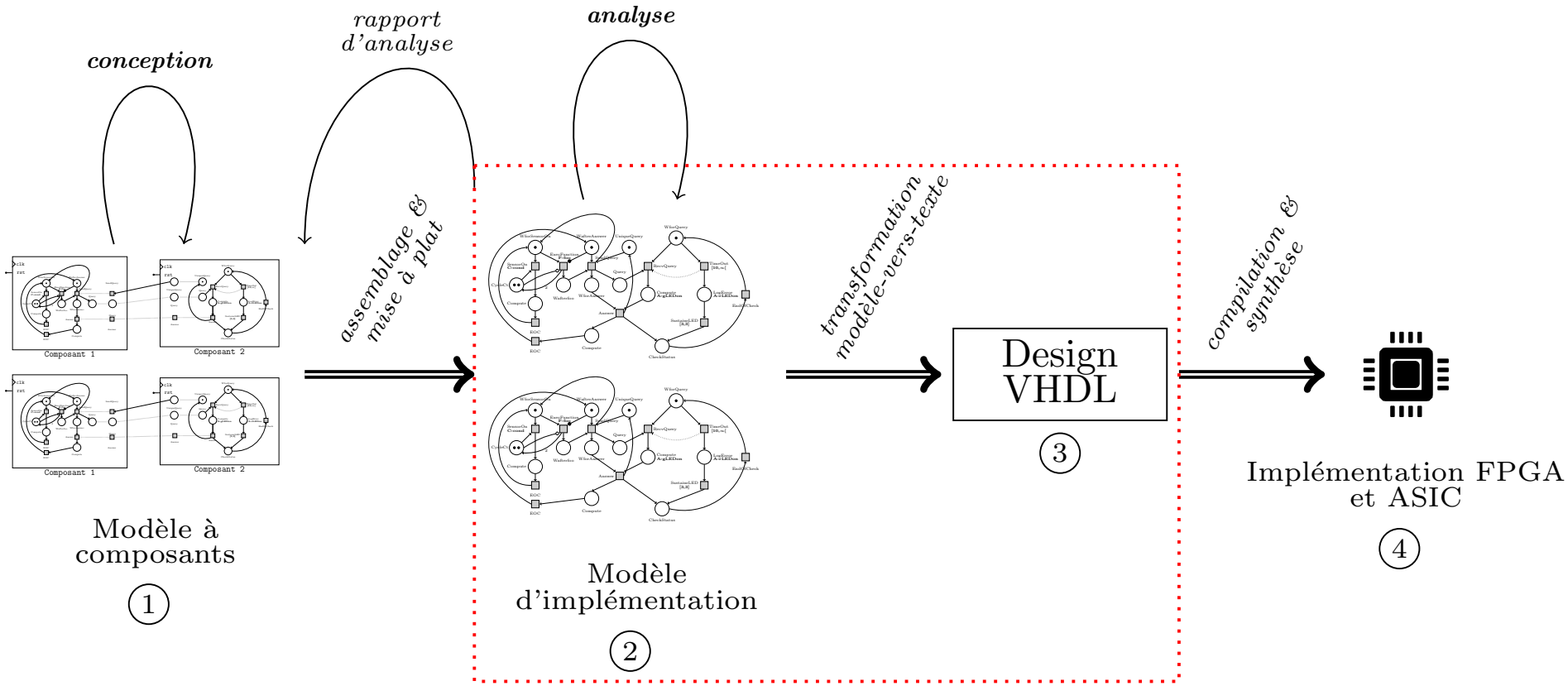
Systèmes numériques dans un contexte critique





- Dispositifs médicaux implantables.
- Vérification de la conception.
- Automatisation de la production.
- **Création de HILECOP (D. Andreu, 2008).**

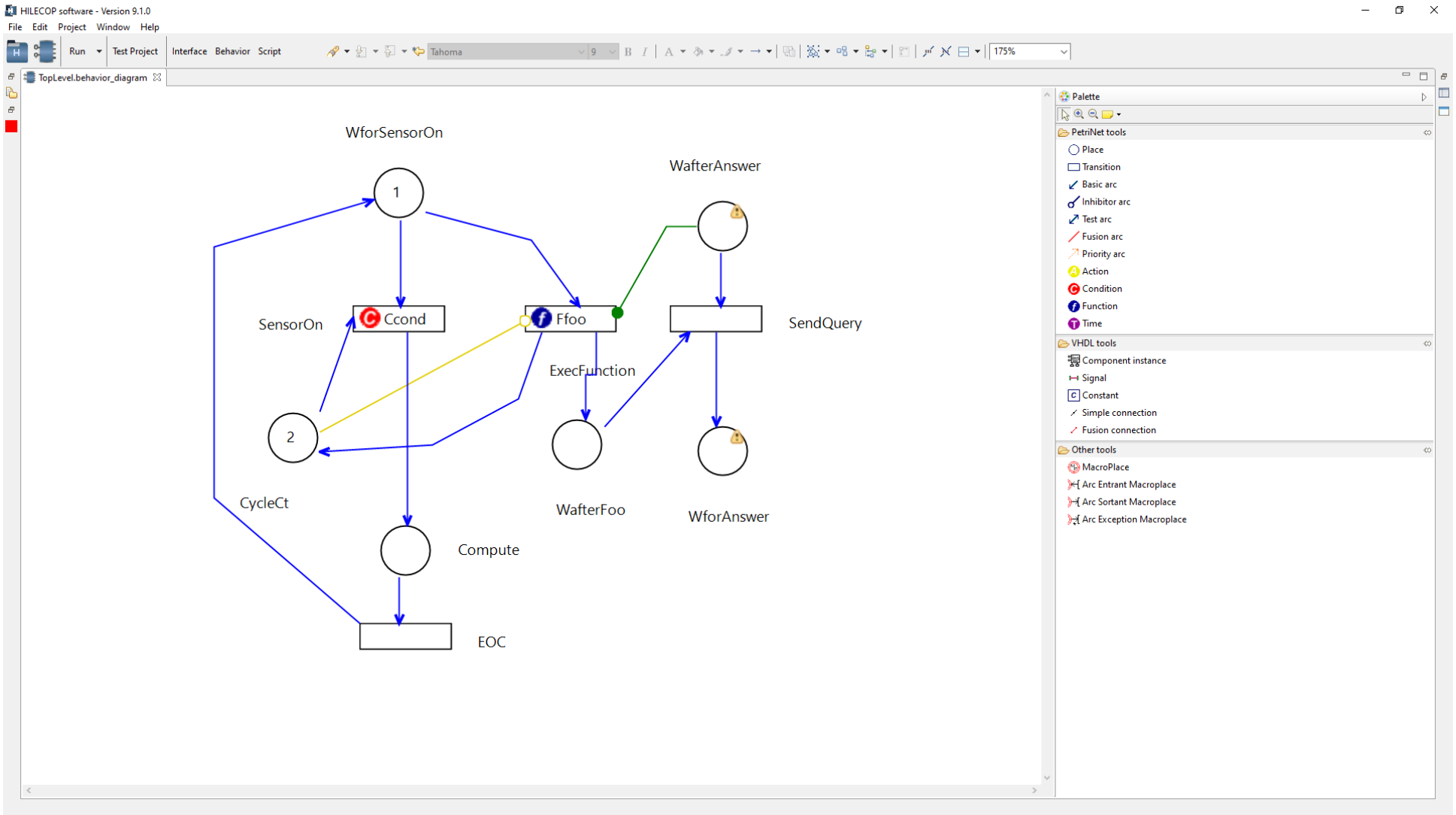
La méthodologie HILECOP



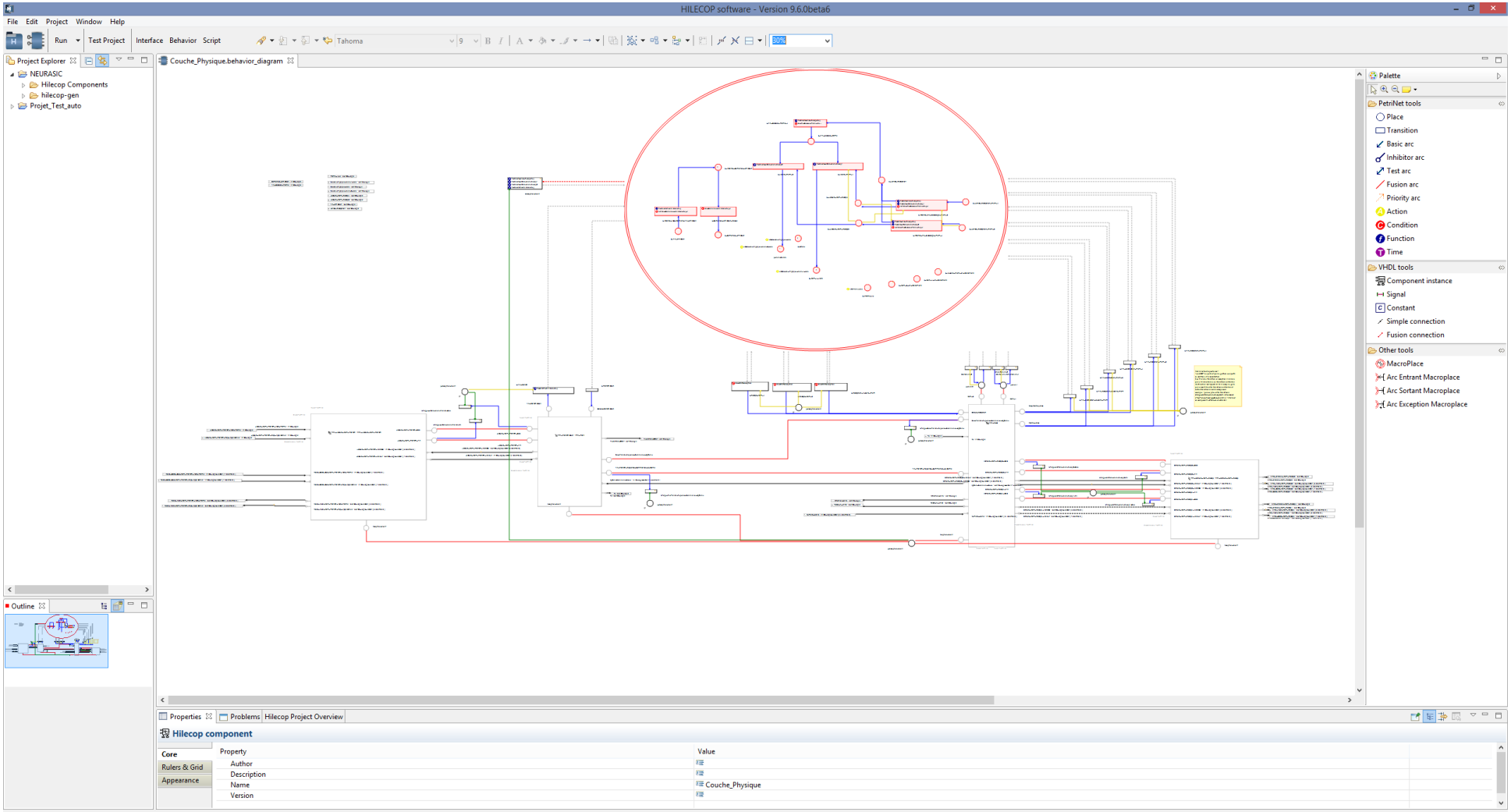
Un travail d'équipe : contributions à HILECOP

- Thèse de H. Leroux (2014).
- Thèse de I. Merzoug (2018).

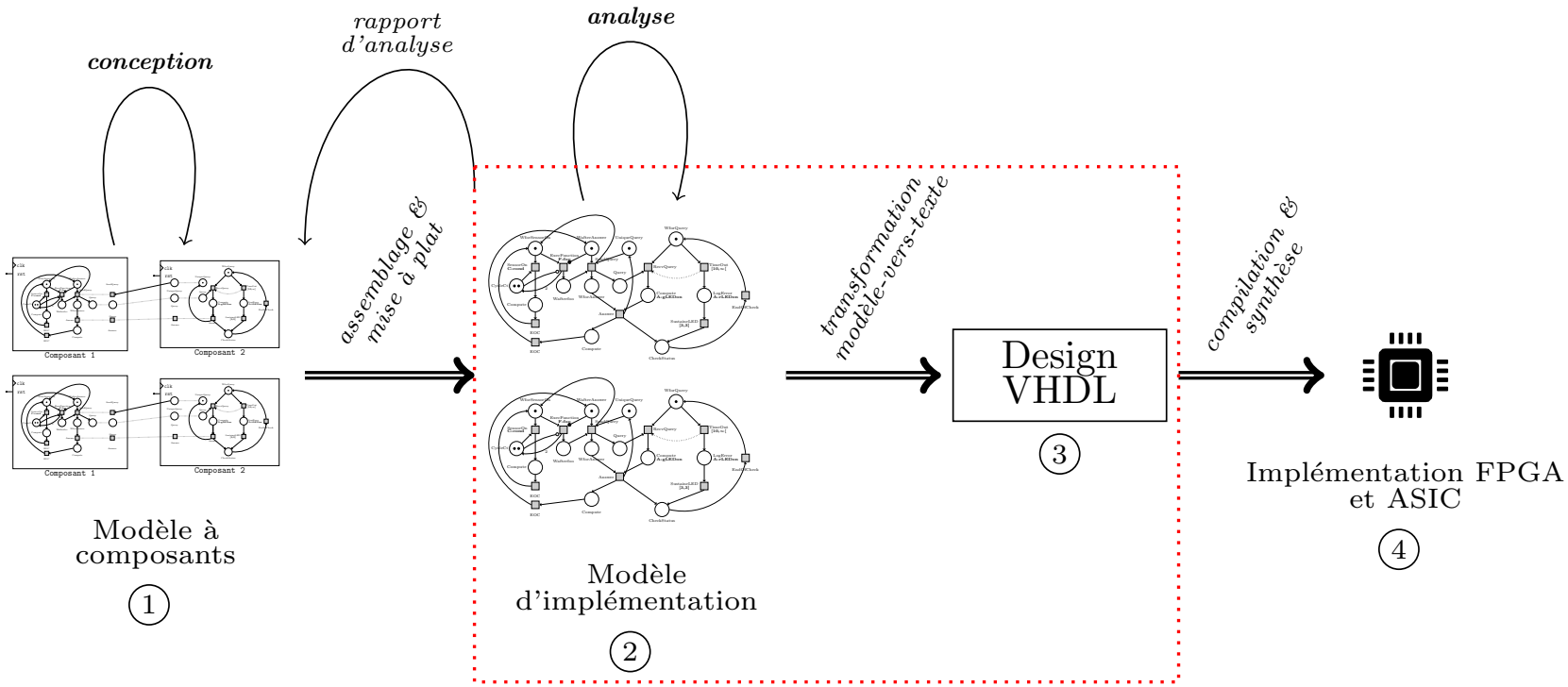
La méthodologie HILECOP



La méthodologie HILECOP



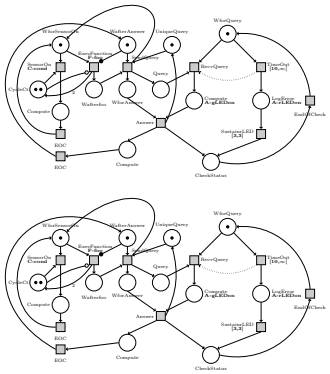
Preuve de préservation de comportement



Sûreté de la conception

- Utilisation de modèles formels.
- Preuves sur les modèles (analyse).

Preuve de préservation de comportement



Modèle d'implémentation

transformation modèle-vers-texte

```
entity Model01 is
  port(
    clock : in std_logic;
    reset_n : in std_logic;
    In1 : in std_logic;
    Out1 : out std_logic
    );
end entity Condition01;

architecture Model01_architecture of Model01 is
  ...
begin
  t1 : entity work.transition generic map(...) port map(...);
  p1 : entity work.place generic map(...) port map(...);
  p0 : entity work.place generic map(...) port map(...);
  t0 : entity work.transition generic map(...) port map(...);
  ...
end architecture Model01_architecture;
```

Design VHDL

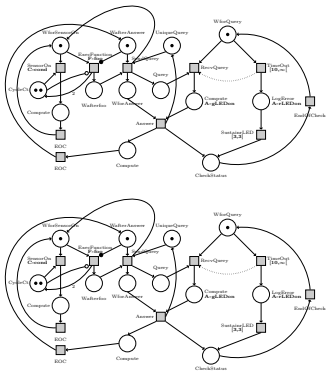
Objectif

Prouver le théorème de préservation de comportement.

Théorème de préservation de comportement

Pour tout modèle en entrée de la transformation et design résultant, le comportement du modèle et du design sont similaires.

Preuve de préservation de comportement



Modèle d'implémentation

transformation modèle-vers-texte

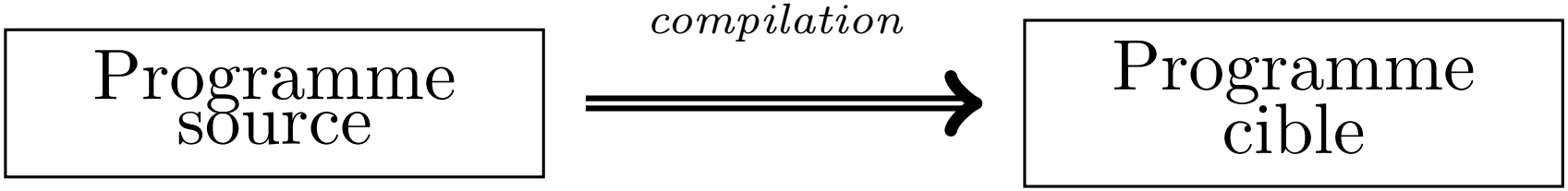
```
entity Model01 is
  port(
     clock : in std_logic;
     reset_n : in std_logic;
     In1 : in std_logic;
     Out1 : out std_logic
     );
end entity Condition01;

architecture Model01_architecture of Model01 is
  ...
begin
  t1 : entity work.transition generic map(...) port map(...);
  p1 : entity work.place generic map(...) port map(...);
  p0 : entity work.place generic map(...) port map(...);
  t0 : entity work.transition generic map(...) port map(...);
  ...
end architecture Model01_architecture;
```

Design VHDL

Intérêts de la preuve

- Fiabiliser la méthodologie.
- Utilisation par l'entreprise **Neurinnov** (D.Andreu, D.Guiraud, 2018) dans la production de dispositifs médicaux implantables (neuroprothèses).
- Obtention de la certification CE (classe III).

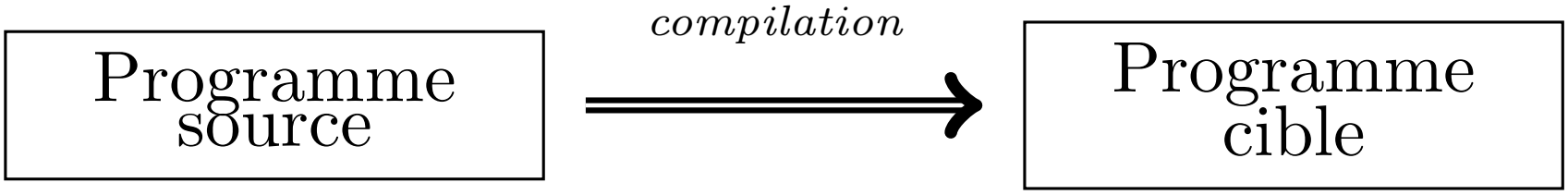
Préservation de comportement et compilateurs



Compilateurs pour langages de programmation génériques

- CompCert, langage C (S. Blazy, X. Leroy, 2005).
- Langage Java (M. Strecker, 2002).
- Langage fonctionnel (A. Chlipala, 2010), CakeML (Y. Tan, 2016).

Préservation de comportement et compilateurs



Question de recherche

Peut-on appliquer les techniques de vérification de compilateurs au cas HILECOP ?

Procédure pour prouver la préservation de comportement

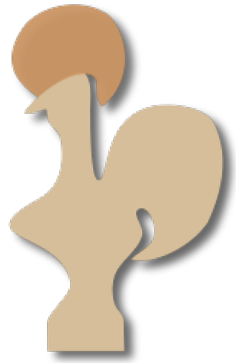
1. Formaliser la sémantique du langage source/cible.
2. Formaliser/Programmer la transformation.
3. Prouver le théorème de préservation de comportement.

« [. . .] grâce aux assistants de preuve, ils[les ordinateurs] garantissent les démonstrations découvertes par les mathématiciens. »

« Du rêve à la réalité des preuves », Jean-Paul Delahaye.

Outil d'aide à la preuve Coq

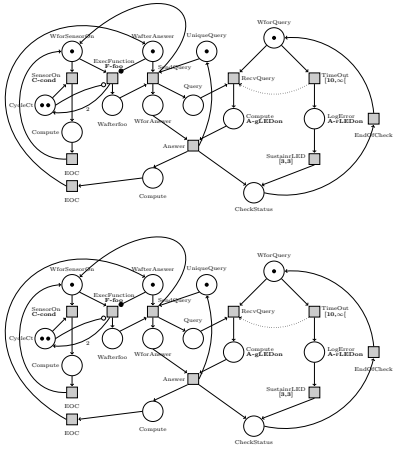
- Premières versions, début des années 80.
- G.Huet, T. Coquand, puis C. Paulin.
- Spécifications, programmes et preuves.



Faits marquants

- Théorème des quatre couleurs (Gonthier, Werner, 2005).
- Compilateur certifié CompCert (Blazy, Leroy, 2005).
- Certification de la Java Card (Gemalto, 2007).
- Théorème de Feit-Thompson (Gonthier, 2013).
- Prix ACM SIGPLAN pour les langages de programmation en 2013.

Plan



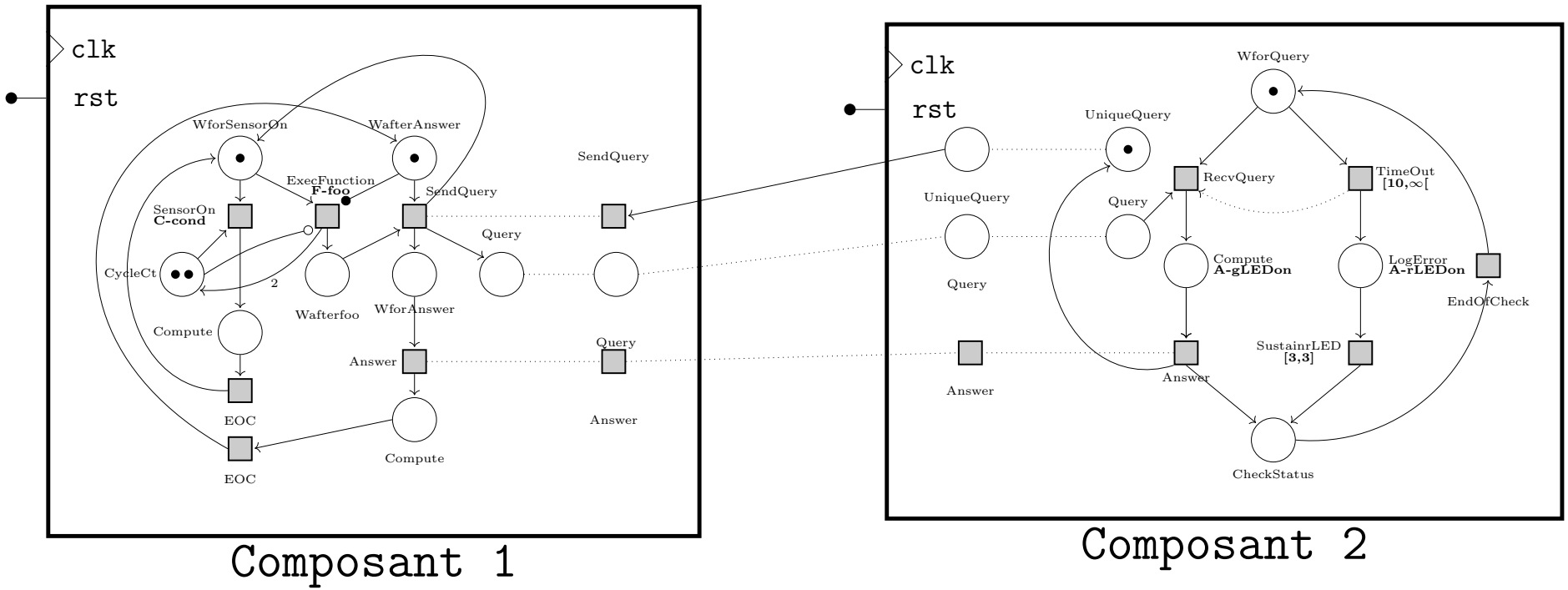
Modèle d'implémentation

transformation modèle-vers-texte

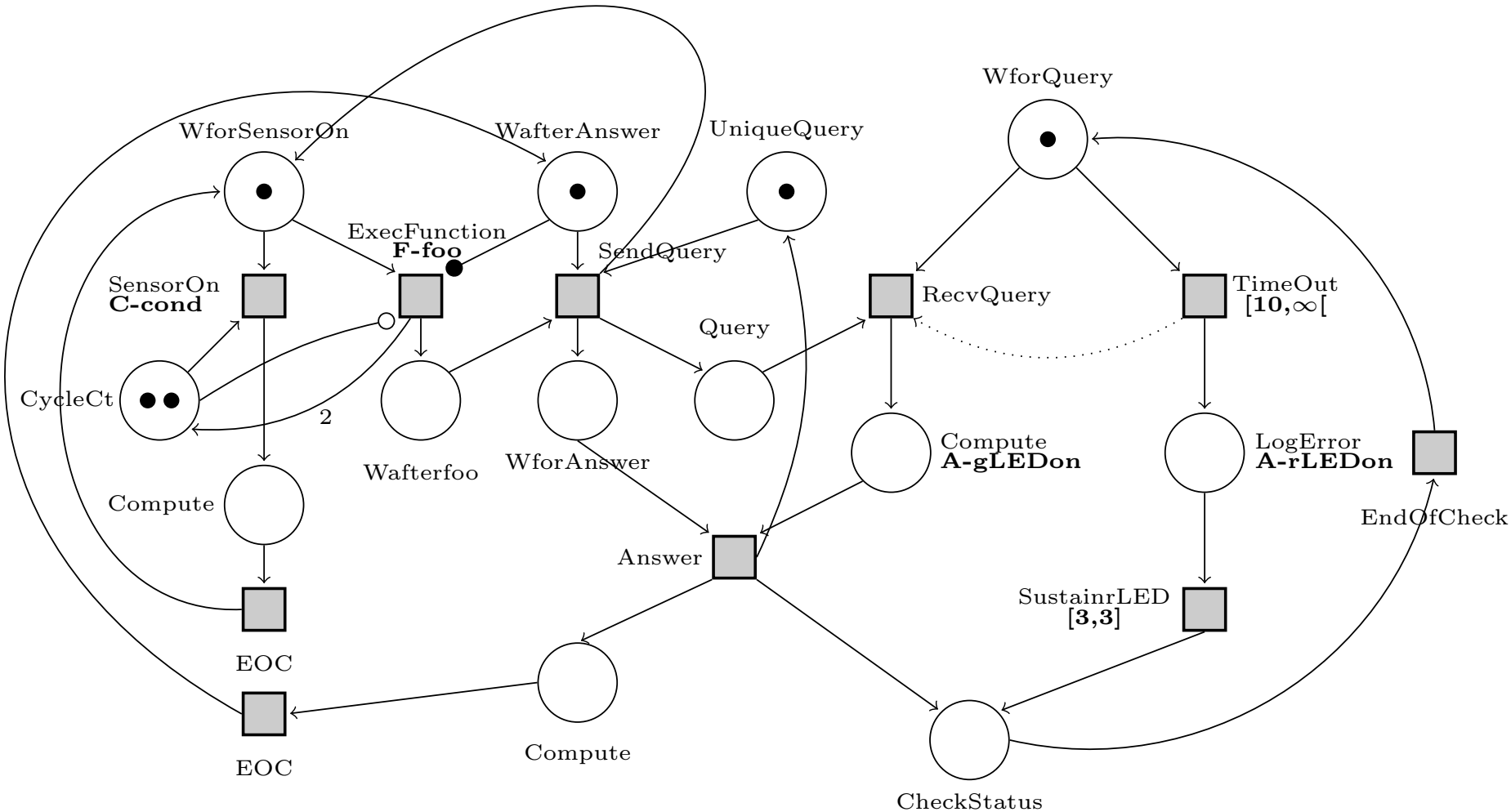
```
entity Model01 is
  port(
     clock : in std_logic;
     reset_n : in std_logic;
     In1 : in std_logic;
     Out1 : out std_logic
     );
end entity Condition01;

architecture Model01_architecture of Model01 is
  ...
begin
  t1 : entity work.transition generic map(...) port map(...);
  p1 : entity work.place generic map(...) port map(...);
  p0 : entity work.place generic map(...) port map(...);
  t0 : entity work.transition generic map(...) port map(...);
  ...
end architecture Model01_architecture;
```

Design VHDL

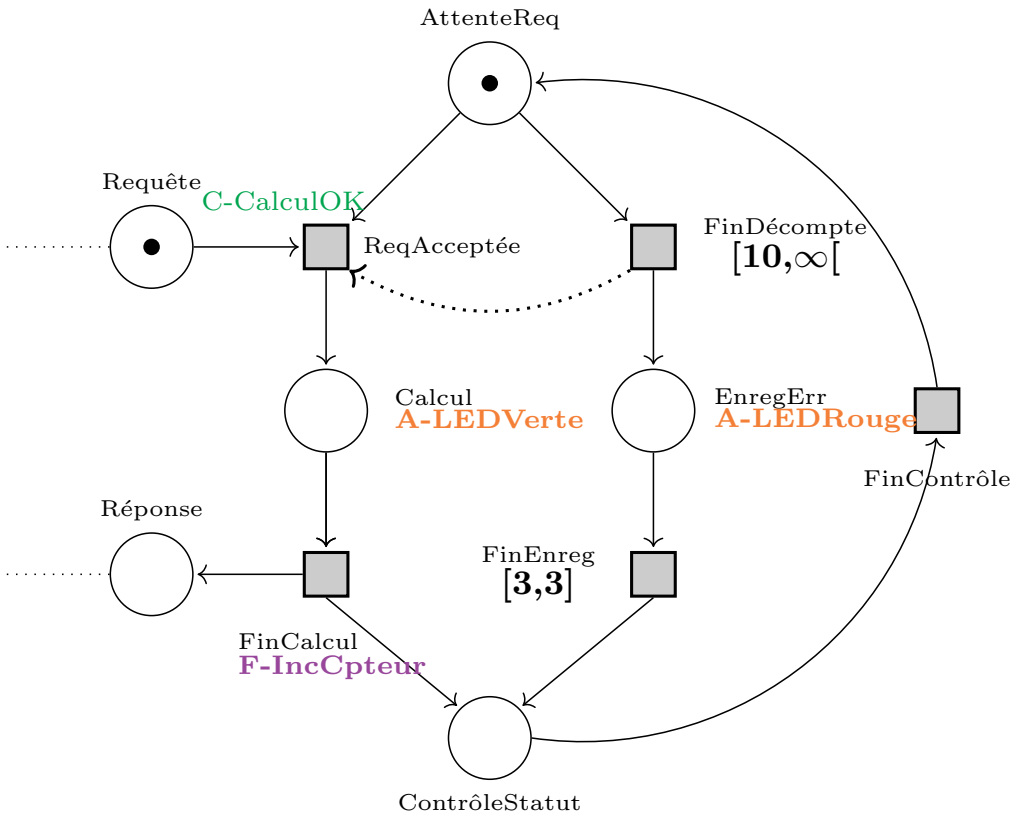
Architecture d'un système numérique HILECOP



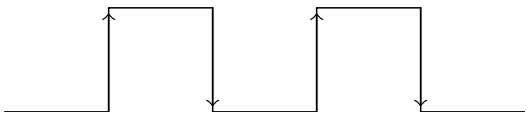
Mise à plat du modèle



Les réseaux de Petri HILECOP (SITPNs)

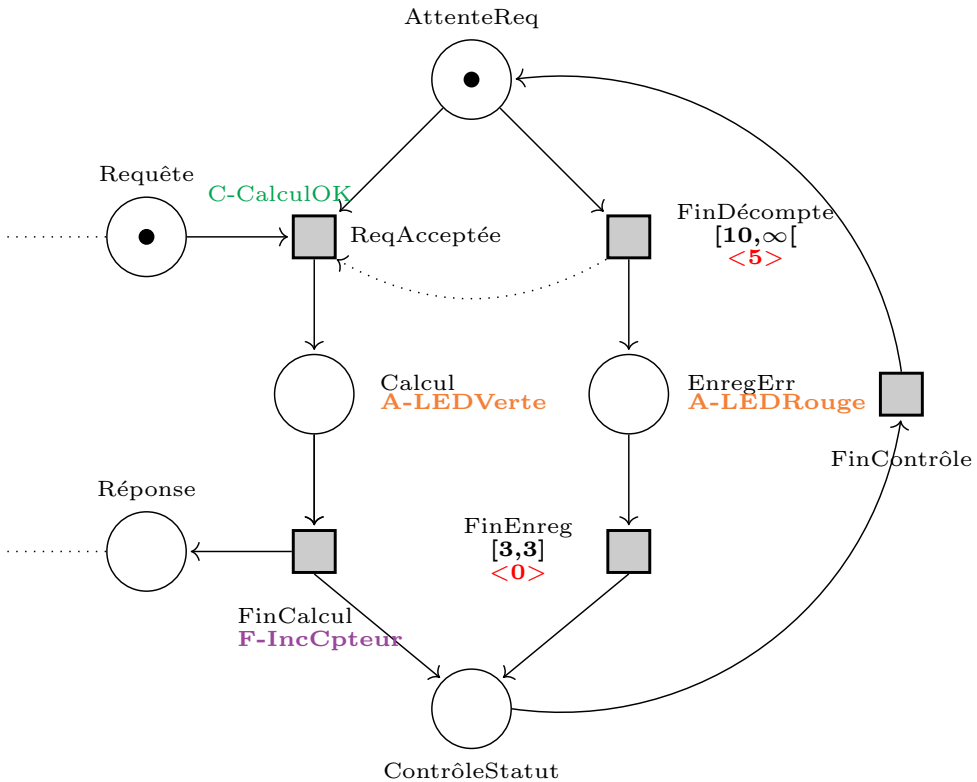


- Synchrones et déterministes.



- Temporels : intervalles de temps (I_s).
- Interprétés : conditions ($\mathcal{C}$), actions ($\mathcal{A}$), fonctions ($\mathcal{F}$).

Etat d'un SITPN



Etat d'un SITPN

Etat= $<M, I, reset_t, ex, cond>$

- Marquage (M, jetons).
- Compteurs de temps (I).
- Conditions ($cond$, valeurs Booléennes).
- Activation des actions (ex), exécution des fonctions (ex).
- Ordres de ràz des compteurs de temps ($reset_t$).

Implémentation en Coq de la sémantique des SITPNs

$$\frac{(1)\ (2)\ (3)\ (4)\ (5)\ (6)}{E, \tau \vdash s \overset{\downarrow}{\rightarrow} s'} \qquad \frac{(7)\ (8)\ (9)}{E, \tau \vdash s \overset{\uparrow}{\rightarrow} s'}$$

Coq

```
Inductive SitpnStateTransition sitpn (E : nat → C sitpn →
bool) (τ : nat)
          (s s' : SitpnState sitpn) : Clk → Prop :=
| SitpnStateTransition_falling :
    (* Premises (1) (2) (3) (4) (5) (6) *)
    ... → SitpnStateTransition E τ s s' ↓

| SitpnStateTransition_rising:
    (* Premises (7) (8) (9) *)
    ... → SitpnStateTransition E τ s s' ↑.
```

Implémentation en Coq de la sémantique des SITPNs

$$\frac{(1)\ (2)\ (3)\ (4)\ (5)\ (6)}{E, \tau \vdash s \xrightarrow{\downarrow} s'} \qquad \frac{(7)\ (8)\ (9)}{E, \tau \vdash s \xrightarrow{\uparrow} s'}$$

$$(7)\ \forall p \in P,\ s'.M(p) = s.M(p) - \sum_{t \in Fired(s)} pre(p,t) + \sum_{t \in Fired(s)} post(t,p)$$

Coq

```
(* Premise 7 *)
forall fired p sum_pre sum_post,
       IsFiredList s fired →
       PreSum p (fun t ⇒ In t fired) sum_pre →
       PostSum p (fun t ⇒ In t fired) sum_post →
       M s' p = M s p − sum_pre + sum_post
```

Implémentation en Coq de la sémantique des SITPNs

$$E, \tau \vdash sitpn \xrightarrow{full} \theta \text{ où}$$
$$\theta = s_0 \xrightarrow{\uparrow} s'_0 \xrightarrow{\downarrow} s_1 \xrightarrow{\uparrow} s'_1 \xrightarrow{\downarrow} \ldots \xrightarrow{\downarrow} s_{\tau-1} \xrightarrow{\uparrow} s'_{\tau-1} \xrightarrow{\downarrow} s_\tau$$

Implémentation en Coq de la sémantique des SITPNs

ExecutionEnd

$$\frac{}{E, 0 \vdash sitpn, s \rightarrow [\,]}$$

ExecutionLoop

$$\frac{E, \tau \vdash s \xrightarrow{\uparrow} s' \qquad E, \tau \vdash s' \xrightarrow{\downarrow} s'' \qquad E, \tau - 1 \vdash sitpn, s'' \rightarrow \theta}{E, \tau \vdash sitpn, s \rightarrow (s' :: s'' :: \theta)} \; \tau > 0$$

Coq

```
Inductive SitpnExecute sitpn
  (E : nat → C sitpn → bool) (s : SitpnState sitpn) :
  nat → list (SitpnState sitpn) → Prop :=
| SitpnExecute_end : SitpnExecute E s 0 [ ]
| SitpnExecute_loop: forall τ θ s' s'',
    SitpnStateTransition E (S τ) s s' ↑ →
    SitpnStateTransition E (S τ) s' s'' ↓ →
    SitpnExecute E s'' τ θ →
    SitpnExecute E s (S τ) (s' :: s'' :: θ).
```

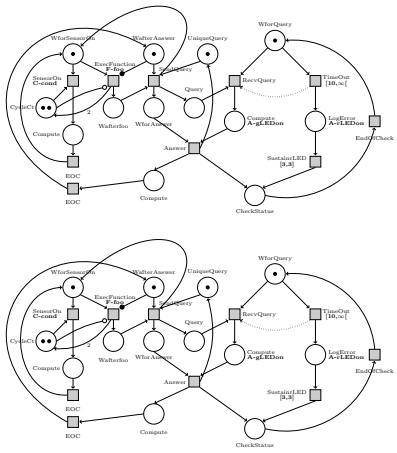
Contributions sur la partie SITPNs

- Travail de précision de la sémantique/complément de la sémantique.
- Implémentation en Coq.
  - Structure, sémantique, notion de modèles bien définis.
- Implémentation d'un interprète vérifié de SITPNs.
  - Interprète et spécification (5000 lignes de code).
  - Preuve de correction/complétude (7000 lignes de code).

Publication

Extended Abstract in the proceedings of the PNSE (Petri Nets and Software Engineering) 2020 international workshop.

Plan



Modèle
d'implémentation

transformation
modèle-vers-texte

```
entity Model01 is
  port(
    clock : in std_logic;
    reset_n : in std_logic;
    In1 : in std_logic;
    Out1 : out std_logic
    );
end entity Condition01;

architecture Model01_architecture of Model01 is
  ...
begin
  t1 : entity work.transition generic map(...) port map(...);
  p1 : entity work.place generic map(...) port map(...);
  p0 : entity work.place generic map(...) port map(...);
  t0 : entity work.transition generic map(...) port map(...);
  ...
end architecture Model01_architecture;
```

Design
VHDL

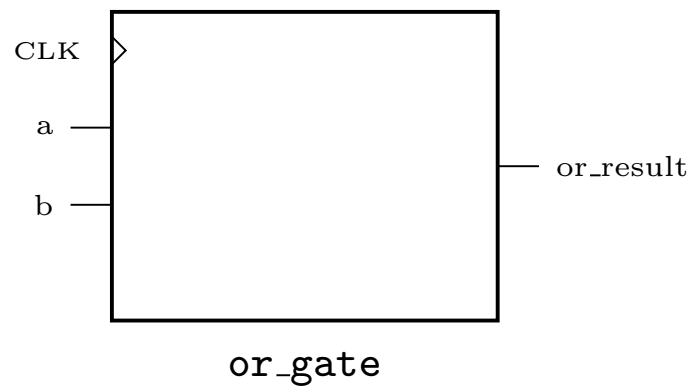
Le langage VHDL

Langage de description de circuits électroniques.

- Signaux véhiculent les valeurs.
- Design = entité + architecture.
- Architecture = processus et/ou composants.
- Exécution concurrente.

VHDL

```
entity or_gate is
  port(
    clk : in std_logic;
    a : in std_logic;
    b : in std_logic;
    or_result : out std_logic
  );
end or_gate;

architecture or_gate_beh of or_gate is
   {signaux internes}
begin
   {instructions concurrentes}
end or_gate_beh;
```



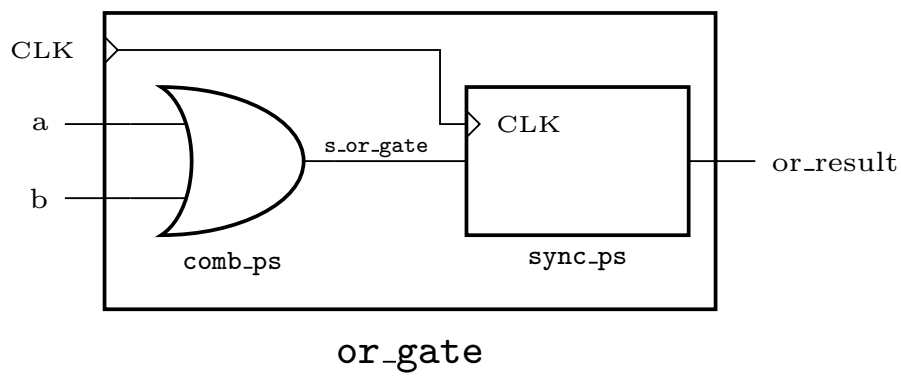
Le langage VHDL

Langage de description de circuits électroniques.

- Signaux véhiculent les valeurs.
- Design = entité + architecture.
- Architecture = processus et/ou composants.
- Exécution concurrente.

VHDL

```
architecture or_gate_beh of or_gate is
   signal s_or_gate : std_logic;
begin

   comb_ps : process(a,b)
   begin
      s_or_gate ⇐ a or b;
   end process;

   sync_ps : process(s_or_gate, clk)
   begin
      if falling_edge(clk) then
         or_result ⇐ s_or_gate;
      end if;
   end process;

end or_gate_beh;
```



or_gate

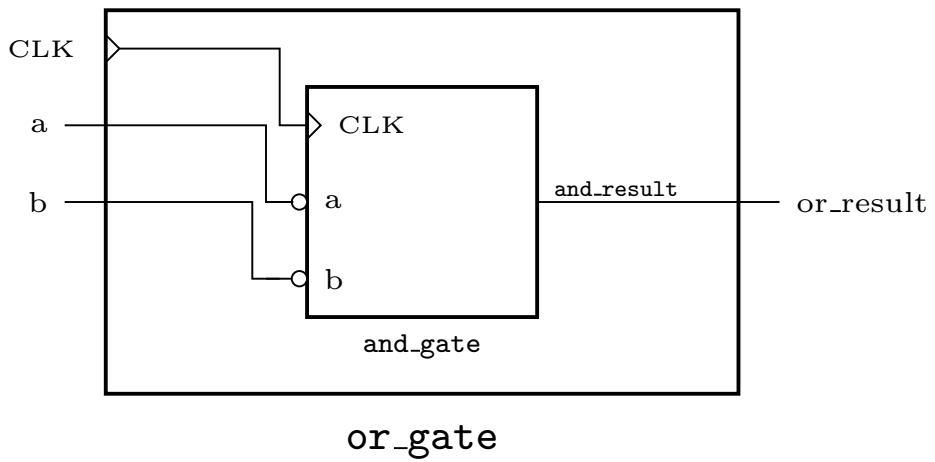
Le langage VHDL

Langage de description de circuits électroniques.

- Signaux véhiculent les valeurs.
- Design = entité + architecture.
- Architecture = processus et/ou composants.
- Exécution concurrente.

VHDL

```
architecture or_gate_beh of or_gate is
begin

   my_and_gate : and_gate
   port map (
      clk ⇒ clk
      a ⇒ not a,
      b ⇒ not b,
      and_result ⇒ or_result
   );

end or_gate_beh;
```



Formalisation d'une sémantique pour VHDL

Sémantique du manuel de référence

Un algorithme de simulation.

Formalisation dans la littérature

- Proche de l'algorithme ou abstraite.
- Opérationnelles, dénotationnelles, axiomatiques...

Nos besoins : VHDL dans HILECOP

- Sous-ensemble *synthétisable* de VHDL.
- Instanciation de composants.
- Exécution synchrone.

Définition de $\mathcal{H}$-VHDL

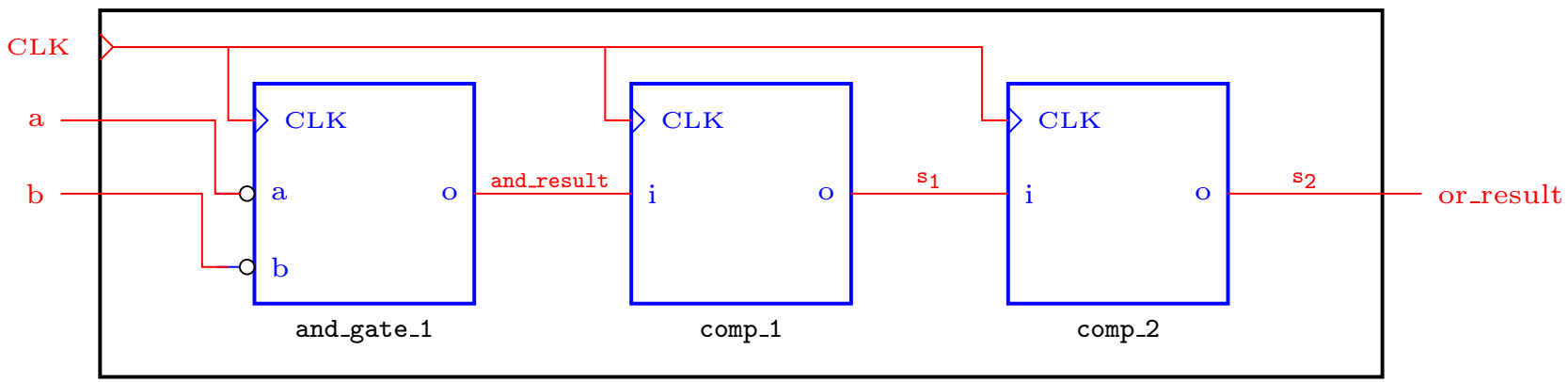
Un sous-ensemble synthétisable et synchrone de VHDL.

design	: :=	`design` id_e id_a gens ports sigs cs
cs	: :=	`process` (id_p, sl, vars, ss)
		| `comp` (id_c, id_e, g, i, o) | cs || cs | `null`
ss	: :=	name $\Leftarrow$ e | name := e
		| `if` (e) ss [ss] | `for` (id,e,e) ss
		| `falling` ss | `rising` ss | `rst` ss ss' | ss; ss | `null`

Etat d'un design $\mathcal{H}$-VHDL

Etat $\sigma \in \Sigma$ avec $\sigma = <\mathcal{S}, \mathcal{C}, \mathcal{E}>$ où :

- $\mathcal{S} \in id \rightarrow value$, le magasin des signaux.
- $\mathcal{C} \in id \rightarrow \Sigma$, le magasin des instances de composants.
- $\mathcal{E}$, signaux à évènements.



Une sémantique opérationnelle synchrone pour $\mathcal{H}$-VHDL

$$\frac{\vdash \mathrm{d} \xrightarrow{\mathit{elab}} \Delta, \sigma_e \qquad \Delta, \sigma_e \vdash \mathrm{d.cs} \xrightarrow{\mathit{init}} \sigma_0 \qquad E_p, \Delta, \tau, \sigma_0 \vdash \mathrm{d.cs} \to \theta}{E_p, \tau \vdash \mathrm{d} \xrightarrow{\mathit{full}} (\sigma_0 :: \theta)}$$

- Elaboration du design (Δ).
- Génération de l'état initial (σ_0).
- Simulation pendant τ cycles d'horloge.

Une sémantique opérationnelle synchrone pour $\mathcal{H}$-VHDL

Boucle de simulation simplifiée, inspirée de (D. Borrione & A. Salem, 1995) et (J. Van Tassel, 1995).

Algorithm 2: SimulationLoop(E_p, τ, Δ, σ_0, d)

$\theta \leftarrow [\sigma_0]$; $\sigma \leftarrow \sigma_0$

while $\tau > 0$ **do**

- $\sigma_i \leftarrow \texttt{Inject}(\Delta, \sigma, E_p, \tau)$
- $\sigma_\uparrow \leftarrow \texttt{RisingEdge}(\Delta, \sigma_i, d.cs)$
- $\sigma' \leftarrow \texttt{Stabilize}(\Delta, \sigma_\uparrow, d.cs)$
- $\sigma_\downarrow \leftarrow \texttt{FallingEdge}(\Delta, \sigma', d.cs)$
- $\sigma \leftarrow \texttt{Stabilize}(\Delta, \sigma_\downarrow, d.cs)$
- $\theta \leftarrow \theta +\!\!+ [\sigma', \sigma]$
- $\tau \leftarrow \tau - 1$

return θ

Exécution aux évènements d'horloge

PsRENoClk

$$\frac{}{\mathcal{D}, \Delta, \sigma \vdash \texttt{process}\ (\mathsf{id}_p, \mathsf{sl}, \mathsf{vars}, \mathsf{ss}) \xrightarrow{\uparrow} \sigma} \quad \texttt{clk} \notin \texttt{sl}$$

PsRECIk

$$\frac{\Delta, \sigma, \sigma, \Lambda \vdash \mathrm{ss} \xrightarrow{ss\uparrow} \sigma', \Lambda'}{\mathcal{D}, \Delta, \sigma \vdash \texttt{process}\ (\mathsf{id}_p, \mathsf{sl}, \mathsf{vars}, \mathsf{ss}) \xrightarrow{\uparrow} \sigma'} \quad \begin{array}{l} \texttt{clk} \in \texttt{sl} \\ \Delta(\mathrm{id}_p) = \Lambda \end{array}$$

Exécution aux évènements d'horloge

ParFE

$$\frac{\mathcal{D}, \Delta, \sigma \vdash \mathrm{cs} \overset{\downarrow}{\rightarrow} \sigma' \qquad \mathcal{D}, \Delta, \sigma \vdash \mathrm{cs}' \overset{\downarrow}{\rightarrow} \sigma''}{\mathcal{D}, \Delta, \sigma \vdash \mathrm{cs} \mid\mid \mathrm{cs}' \overset{\downarrow}{\rightarrow} \mathtt{merge}(\sigma, \sigma', \sigma'')} \quad \mathcal{E}' \cap \mathcal{E}'' = \emptyset$$

Coq

$$\begin{aligned}
&\mathtt{Definition}\ \mathtt{merge}(\sigma,\ \sigma',\ \sigma'') :=\\
&\quad \mathtt{let}\ \sigma = <\mathcal{S},\mathcal{C},\mathcal{E}>\ \mathtt{in}\\
&\quad \mathtt{let}\ \sigma' = <\mathcal{S}',\mathcal{C}',\mathcal{E}'>\ \mathtt{in}\\
&\quad \mathtt{let}\ \sigma'' = <\mathcal{S}'',\mathcal{C}'',\mathcal{E}''>\ \mathtt{in}\\
&\quad \mathtt{let}\ \mathcal{S}_m(\mathrm{id}) = \begin{cases} \mathcal{S}'(\mathrm{id}) & \mathtt{if}\ \mathrm{id} \in \mathcal{E}' \\ \mathcal{S}''(\mathrm{id}) & \mathtt{if}\ \mathrm{id} \in \mathcal{E}'' \\ \mathcal{S}(\mathrm{id}) & \textit{otherwise} \end{cases} \mathtt{in}\\
&\quad \mathtt{let}\ \mathcal{C}_m(\mathrm{id}) = \begin{cases} \mathcal{C}'(\mathrm{id}) & \mathtt{if}\ \mathrm{id} \in \mathcal{E}' \\ \mathcal{C}''(\mathrm{id}) & \mathtt{if}\ \mathrm{id} \in \mathcal{E}'' \\ \mathcal{C}(\mathrm{id}) & \textit{otherwise} \end{cases} \mathtt{in}\\
&\quad \mathtt{let}\ \mathcal{E}_m = \mathcal{E}' \cup \mathcal{E}''\ \mathtt{in}\ <\mathcal{S}_m,\mathcal{C}_m,\mathcal{E}_m>.
\end{aligned}$$

Phase de stabilisation

StabilizeEnd

$$\frac{\mathcal{D}, \Delta, \sigma \vdash \mathrm{cs} \xrightarrow{\mathit{comb}} \sigma}{\mathcal{D}, \Delta, \sigma \vdash \mathrm{cs} \xrightarrow{\rightsquigarrow} \sigma} \; \mathcal{E} = \emptyset$$

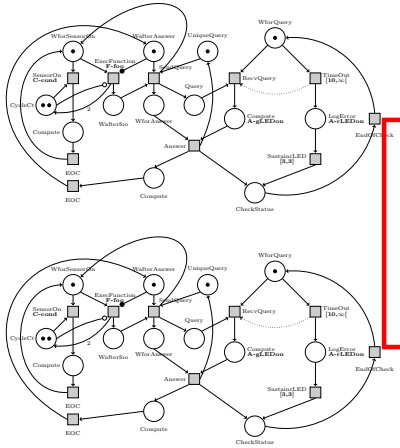
StabilizeLoop

$$\frac{\mathcal{D}, \Delta, \sigma \vdash \mathrm{cs} \xrightarrow{\mathit{comb}} \sigma' \qquad \mathcal{D}, \Delta, \sigma' \vdash \mathrm{cs} \xrightarrow{\rightsquigarrow} \sigma''}{\mathcal{D}, \Delta, \sigma \vdash \mathrm{cs} \xrightarrow{\rightsquigarrow} \sigma''} \; \begin{array}{l} \mathcal{E}' \neq \emptyset \\ \mathcal{E}'' = \emptyset \end{array}$$

Contributions sur la partie VHDL

- Formalisation de la syntaxe et de la sémantique de $\mathcal{H}$-VHDL.
  - Une vingtaine de pages.
  - Une trentaine de relations.
- Implémentation de la sémantique avec Coq.
  - 31 fichiers.
  - 2100 lignes de code.

Plan



Modèle
d'implémentation

transformation
modèle-vers-texte

```
entity Model01 is
  port(
     clock : in std_logic;
     reset_n : in std_logic;
     In1 : in std_logic;
     Out1 : out std_logic
     );
end entity Condition01;

architecture Model01_architecture of Model01 is
  ...
begin
  t1 : entity work.transition generic map(...) port map(...);
  p1 : entity work.place generic map(...) port map(...);
  p0 : entity work.place generic map(...) port map(...);
  t0 : entity work.transition generic map(...) port map(...);
  ...
end architecture Model01_architecture;
```

Design
VHDL

La transformation HILECOP

Algorithm 3: `sitpn_to_hvhdl`(*sitpn*)

$d \leftarrow$ `design` $\emptyset$ $\emptyset$ $\emptyset$ `null`

$\gamma \leftarrow \emptyset$

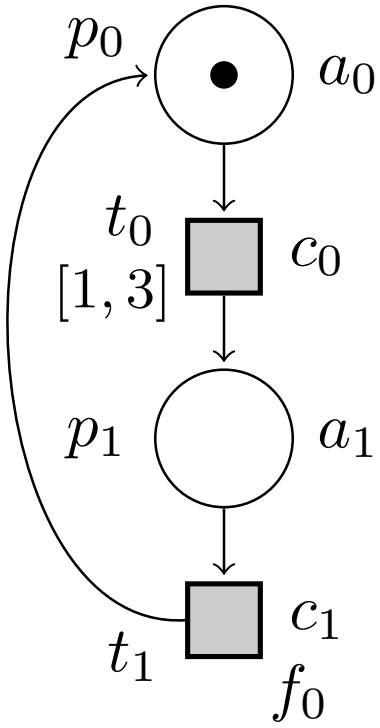
`generate_architecture`(*sitpn*, *d*, γ)

`generate_interconnections`(*sitpn*, *d*, γ)

`generate_ports`(*sitpn*, *d*, γ)

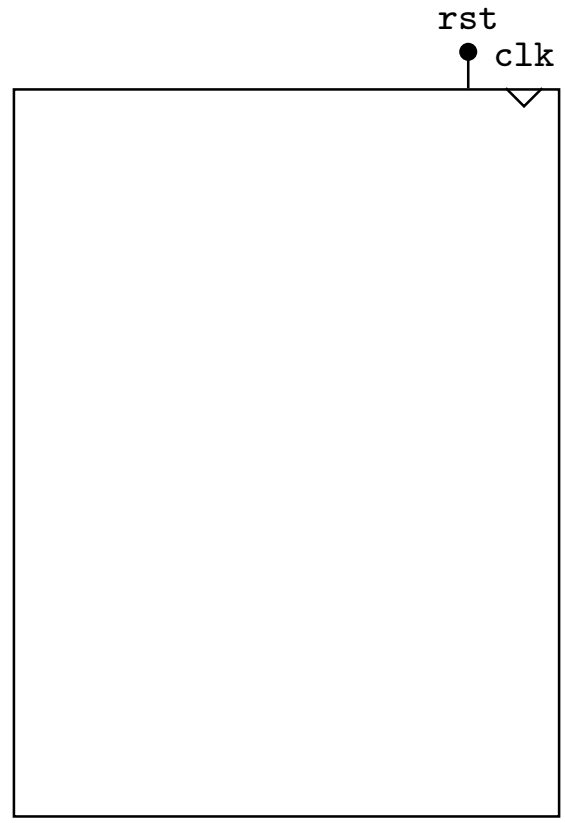
return (d,γ)

La transformation HILECOP



SITPN

Transformation



Design $\mathcal{H}$-VHDL de top-niveau
+ lien γ

Générer les composants `place` et `transition`

Algorithm 5: `sitpn_to_hvhdl`(*sitpn*)

$d \leftarrow$ `design` $\emptyset$ $\emptyset$ $\emptyset$ `null`

$\gamma \leftarrow \emptyset$

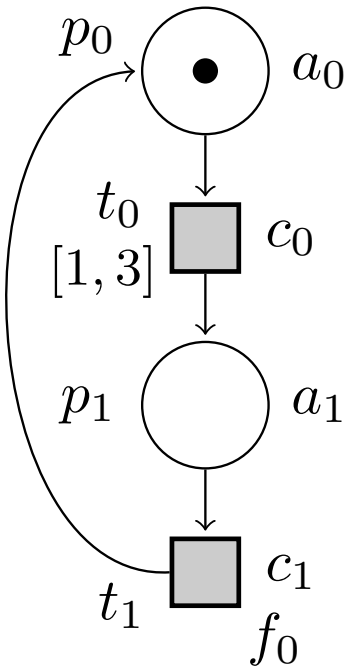
`generate_architecture`(*sitpn*, *d*, γ)

`generate_interconnections`(*sitpn*, *d*, γ)

`generate_ports`(*sitpn*, *d*, γ)

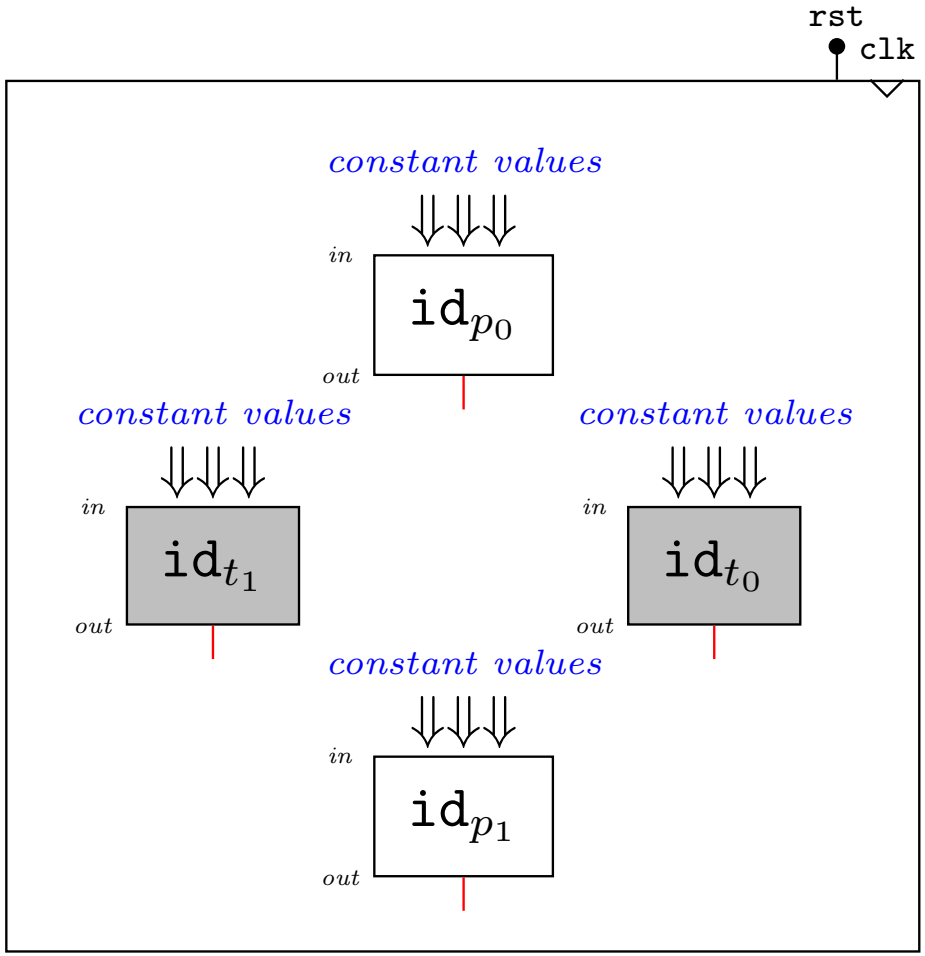
return *(d,*γ*)*

Générer les composants `place` et `transition`



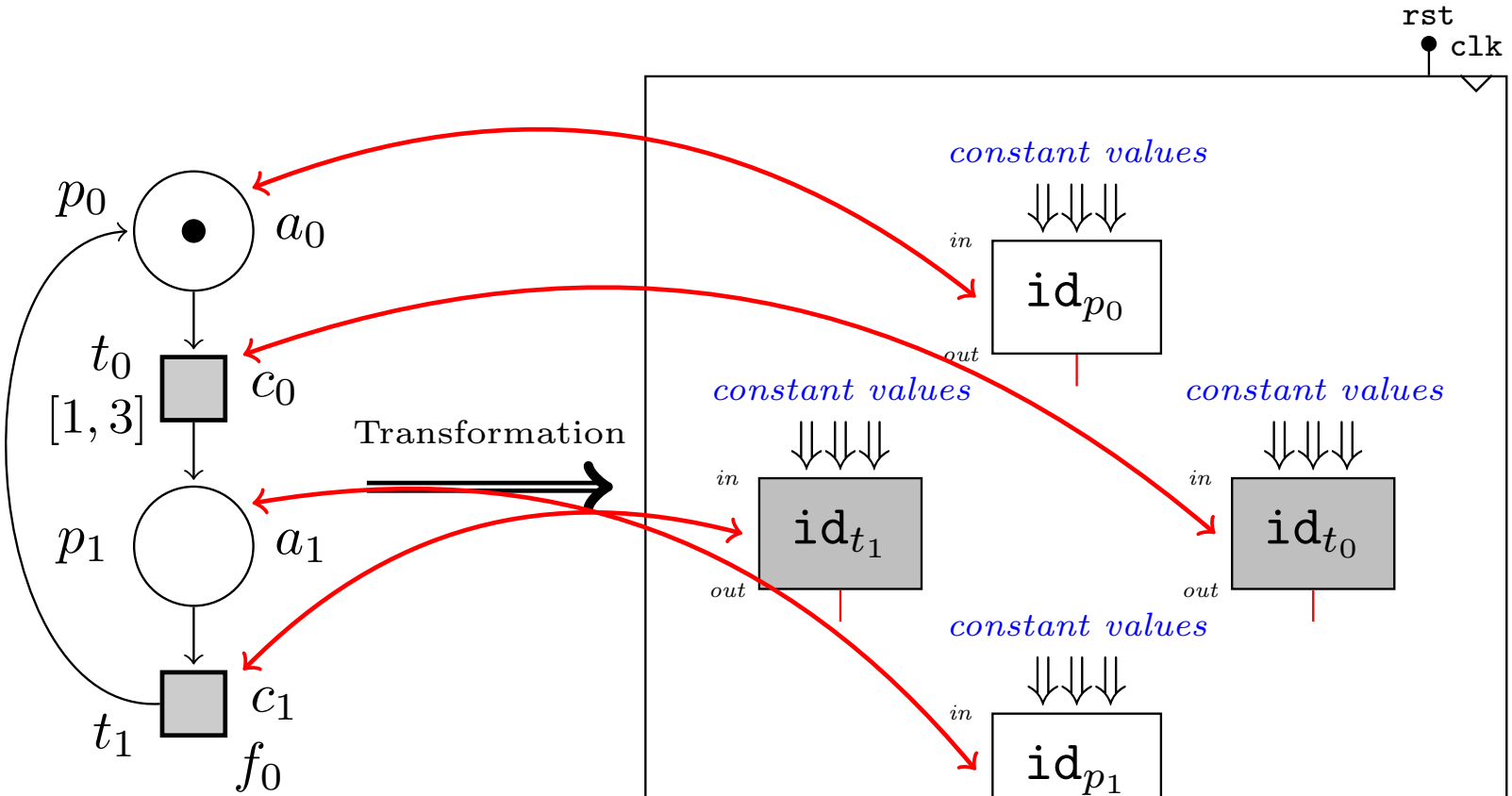
SITPN

Transformation



Design $\mathcal{H}$-VHDL de top-niveau

Générer les composants `place` et `transition`



Interconnecter les composants

Algorithm 8: `sitpn_to_hvhdl`(*sitpn*)

$d \leftarrow$ `design` $\emptyset$ $\emptyset$ $\emptyset$ `null`

$\gamma \leftarrow \emptyset$

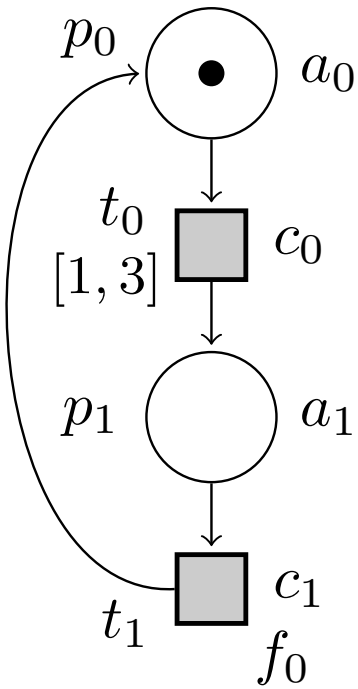
`generate_architecture`(*sitpn*, *d*, γ)

`generate_interconnections`(*sitpn*, *d*, γ)

`generate_ports`(*sitpn*, *d*, γ)

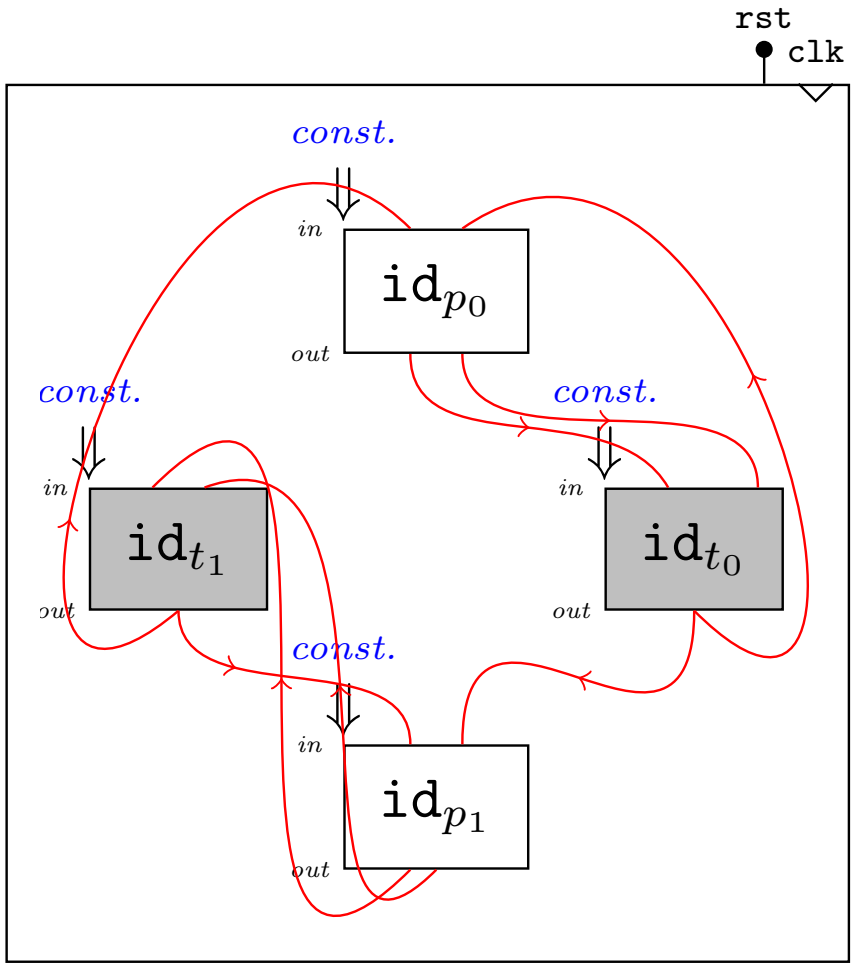
return (d,γ)

Interconnecter les composants



SITPN

Transformation



Design $\mathcal{H}$-VHDL de top-niveau

Générer les ports d'entrée et de sortie (interface)

Algorithm 10: `sitpn_to_hvhdl`(*sitpn*)

$d \leftarrow$ `design` $\emptyset$ $\emptyset$ $\emptyset$ `null`

$\gamma \leftarrow \emptyset$

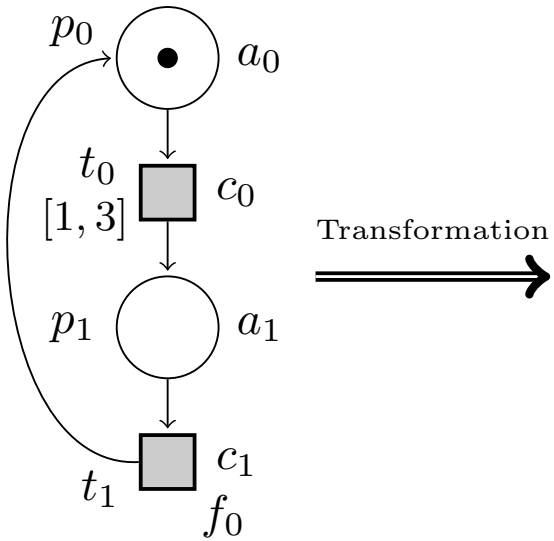
`generate_architecture`(*sitpn*, *d*, γ)

`generate_interconnections`(*sitpn*, *d*, γ)

`generate_ports`(*sitpn*, *d*, γ)

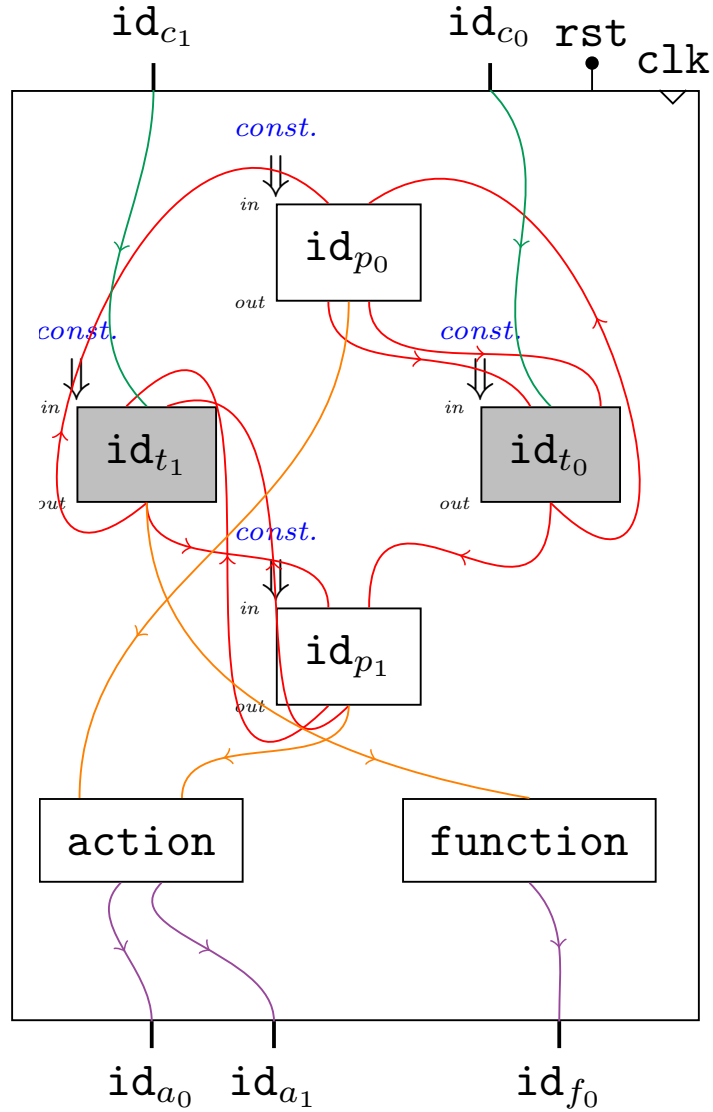
return (d,γ)

Générer les ports d'entrée et de sortie (interface)



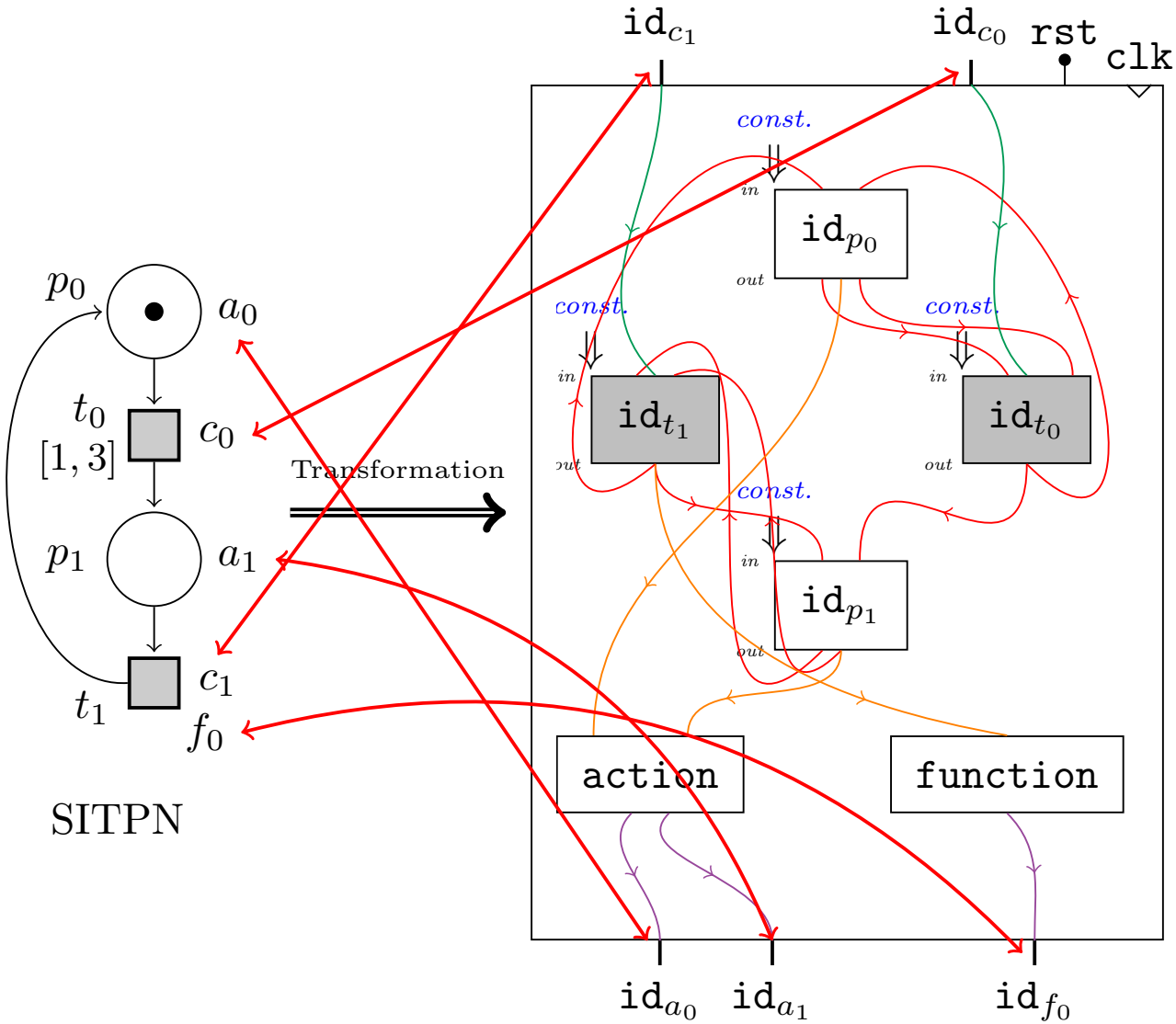
SITPN

Transformation



Design $\mathcal{H}$-VHDL de top-niveau

Générer les ports d'entrée et de sortie (interface)



Design $\mathcal{H}$-VHDL de top-niveau

Implémentation de la transformation HILECOP

$i \leftarrow 0$
foreach $t \in \texttt{input}(p)$ **do**
  $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \leftarrow \texttt{get\_comp}(\gamma(t), d.cs)$
  $i_p \leftarrow i_p \cup \{(\texttt{itf}(i), \texttt{actual}(\texttt{fired}, o_t))\}$
  $i \leftarrow i + 1$

Coq

```
Definition connect_to_input_tci (i_p : inputmap) (i : nat) (t : T sitpn) :
  CompileTimeState (inputmap * nat) :=
  do id_t ← get_tci_id_from_binder t;
  do '( id_e, g_t, i_t, o_t) ← get_comp id_t;
  do a ← actual Transition.fired o_t;
  Ret (i_p ++ [associp_ (Place.itf [[i]]) a], i + 1).

Definition connect_to_input_tcis (pinfo : PlaceInfo sitpn) (i_p : inputmap) :
  CompileTimeState inputmap :=
  let conn_to_in_tci := (fun '(i_p, i) ⇒ connect_to_input_tci pinfo i_p i) in
  do iidx ← ListMonad.fold_left conn_to_in_tci (tinputs pinfo) (i_p, 0);
  Ret (fst iidx).
```

Contributions sur la partie transformation

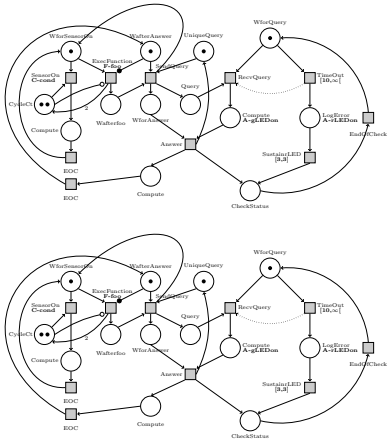
Algorithme

- Une trentaine de primitives et procédures.
- 130 lignes.

Implémentation

- Environ 1000 lignes de code, réparties en 7 modules.
- Inspiration : CompCert et les travaux d'Adam Chlipala.
- Adoption du monad *state-and-error*.
- Génération du lien γ.

Plan



Modèle
d'implémentation

transformation
modèle-vers-texte

```
entity Model01 is
  port(
     clock : in std_logic;
     reset_n : in std_logic;
     In1 : in std_logic;
     Out1 : out std_logic
     );
end entity Condition01;

architecture Model01_architecture of Model01 is
   ...
begin
   t1 : entity work.transition generic map(...) port map(...);
   p1 : entity work.place generic map(...) port map(...);
   p0 : entity work.place generic map(...) port map(...);
   t0 : entity work.transition generic map(...) port map(...);
   ...
end architecture Model01_architecture;
```

Design
VHDL

Préservation de comportement?

Le théorème de préservation de comportement

Pour tout modèle SITPN *sitpn* et design $\mathcal{H}$-VHDL *d* résultant de la transformation modèle-vers-texte HILECOP, l'exécution de *sitpn* et *d* depuis leur état initial durant τ cycles d'horloge produit des traces d'exécutions similaires.

Coq

```
Theorem full_bisimulation :
  forall sitpn d γ E_c τ θ_s θ_σ,

    sitpn_to_hvhdl sitpn = (d, γ) →
    SitpnFullExec sitpn E_c τ θ_s →
    hfullsim E_p τ d θ_σ →

    FullSimTrace γ θ_s θ_σ.
```

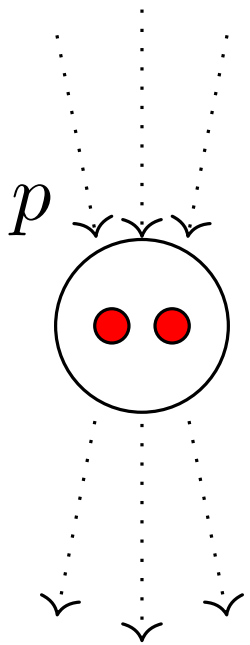
Le théorème de préservation de comportement

Similarité comportementale

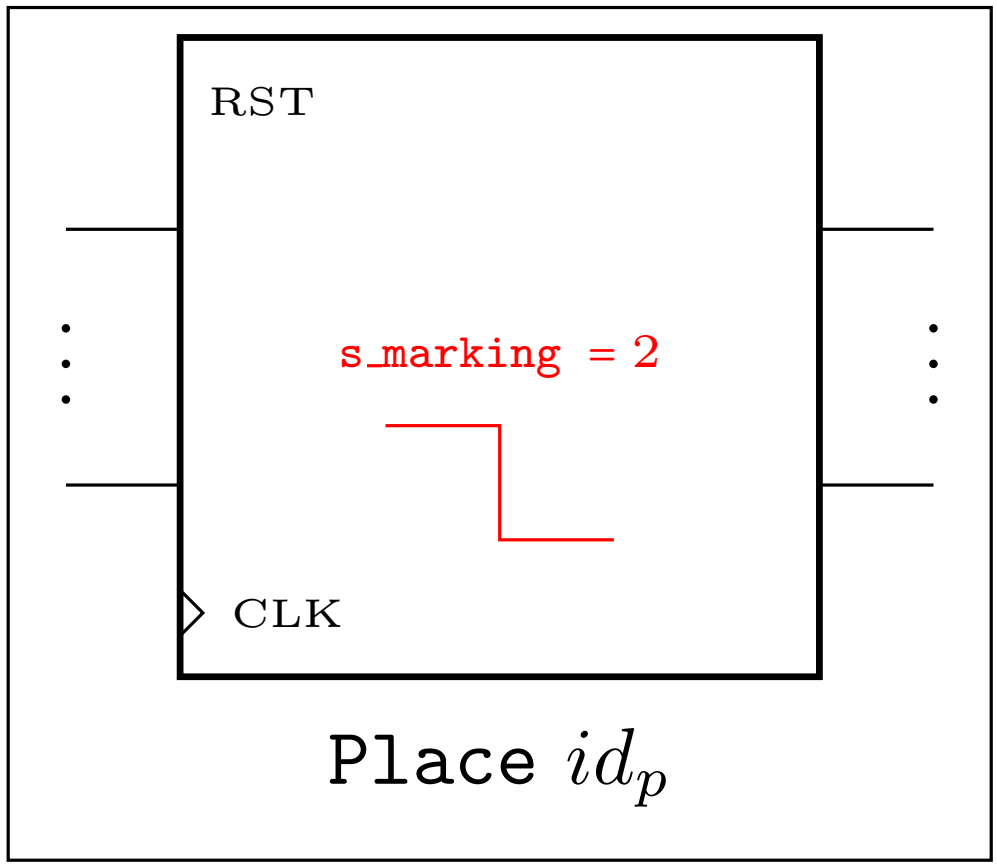
Pour tout modèle SITPN *sitpn* et design $\mathcal{H}$-VHDL *d* résultant de la transformation modèle-vers-texte HILECOP, l'exécution de *sitpn* et *d* depuis leur état initial durant τ cycles d'horloge produit des traces d'exécutions similaires.

- Quoi ?
- Quand ?

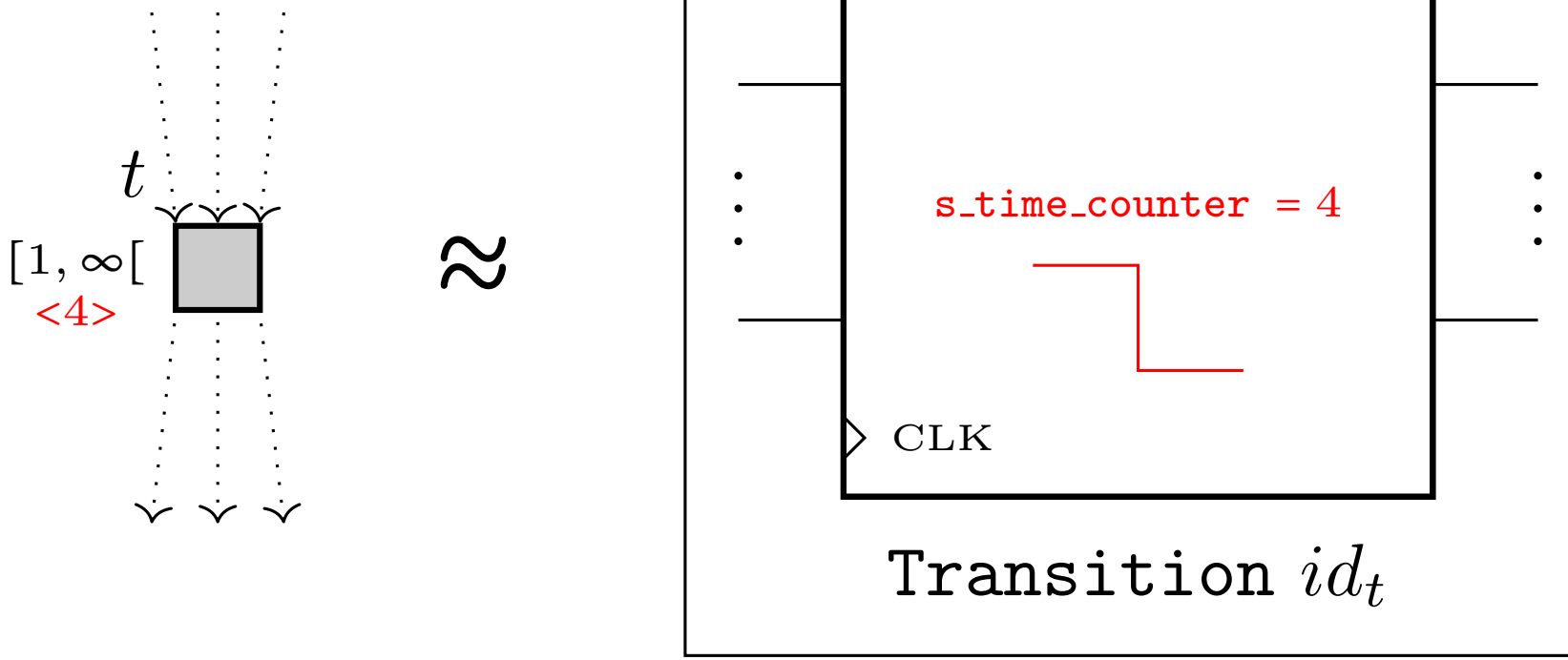
La relation de similarité entre états



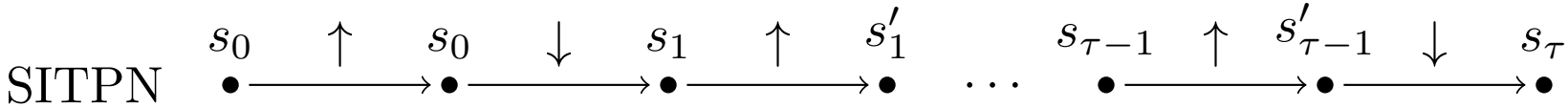
$\approx$



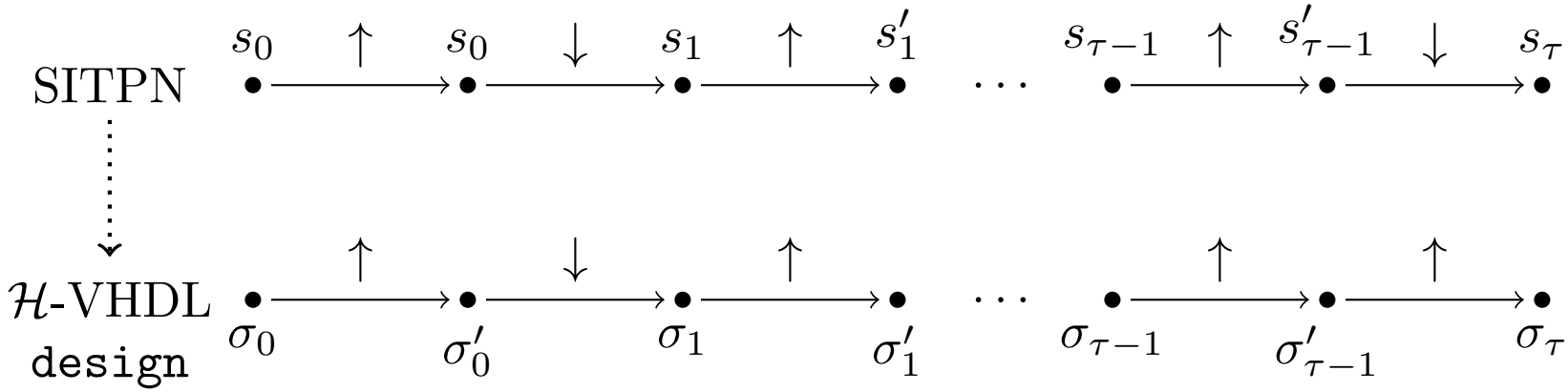
La relation de similarité entre états



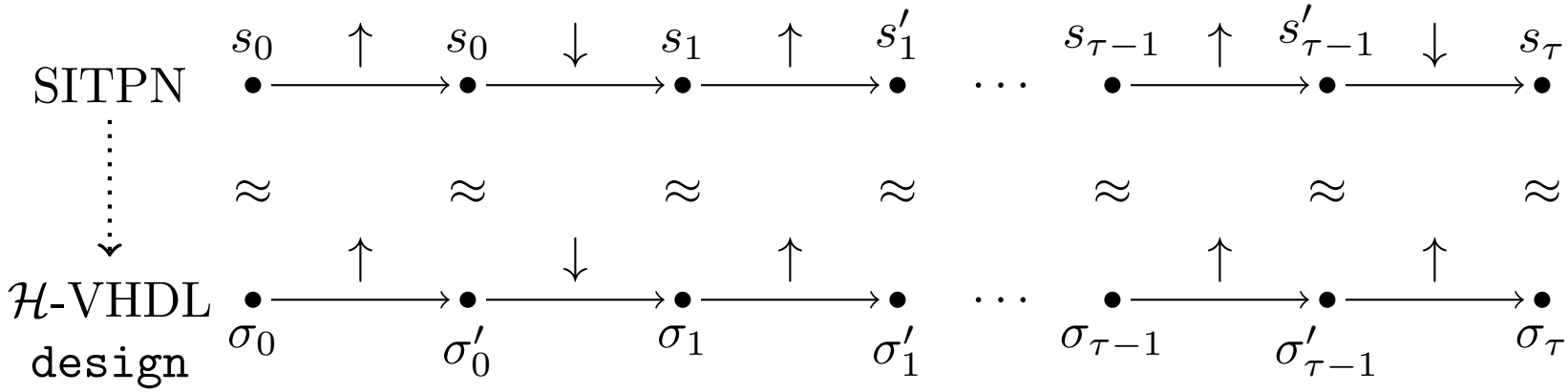
Traces d'exécution et preuve



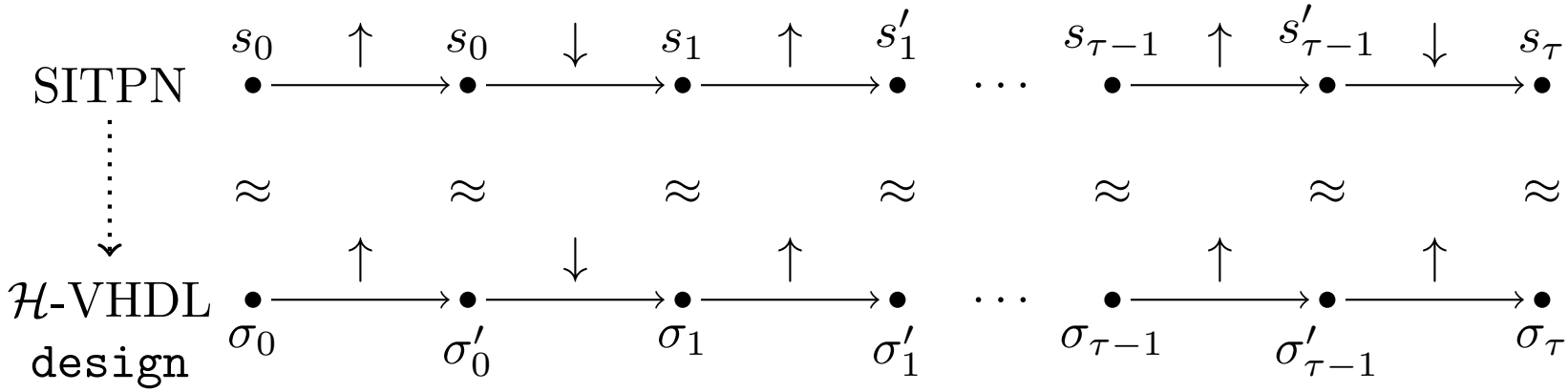
Traces d'exécution et preuve



Traces d'exécution et preuve



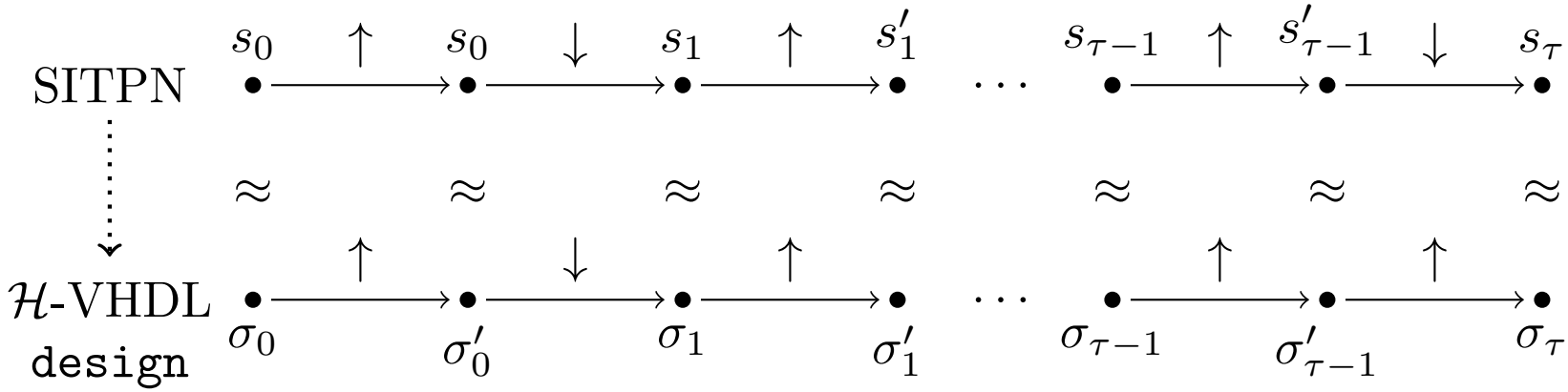
Traces d'exécution et preuve



Preuve de bisimulation

Par induction sur la structure des traces d'exécution.

Traces d'exécution et preuve

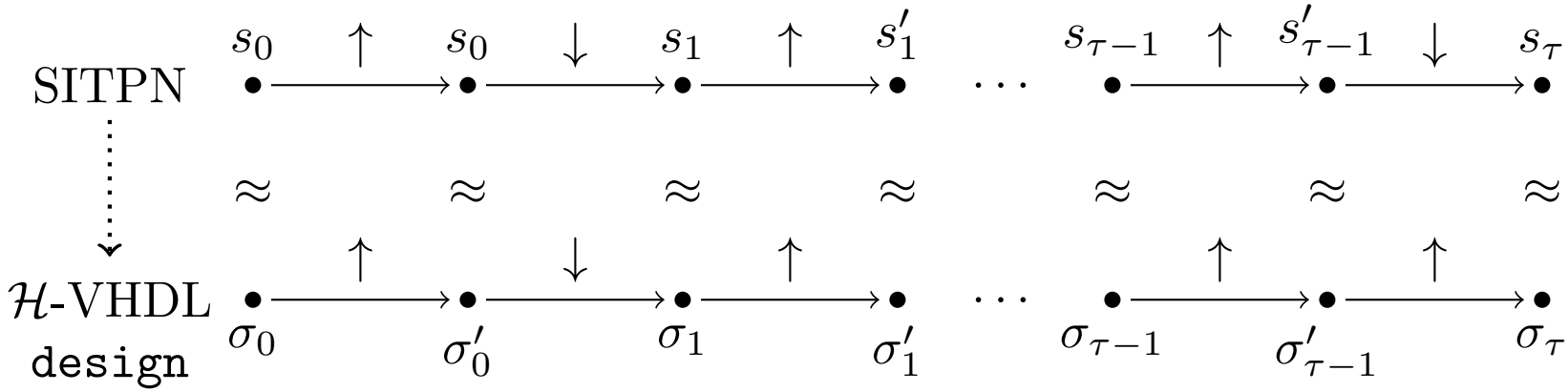


Preuve de bisimulation

Par induction sur la structure des traces d'exécution.

1. Prouver que les états initiaux sont similaires.

Traces d'exécution et preuve

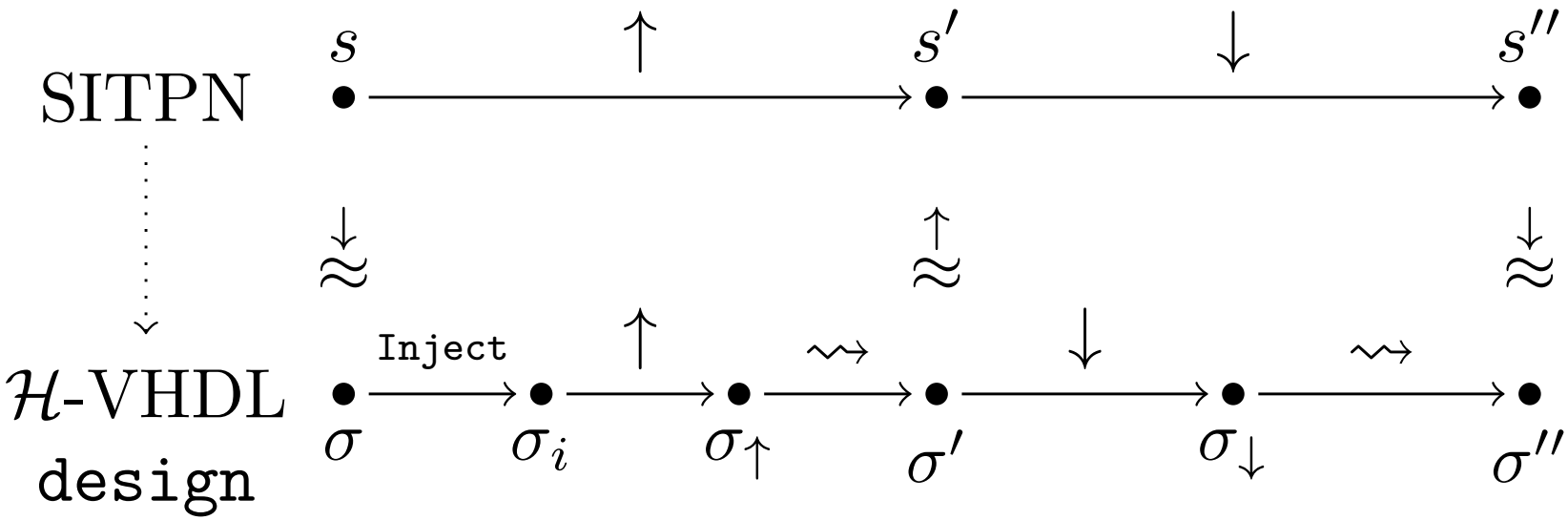


Preuve de bisimulation

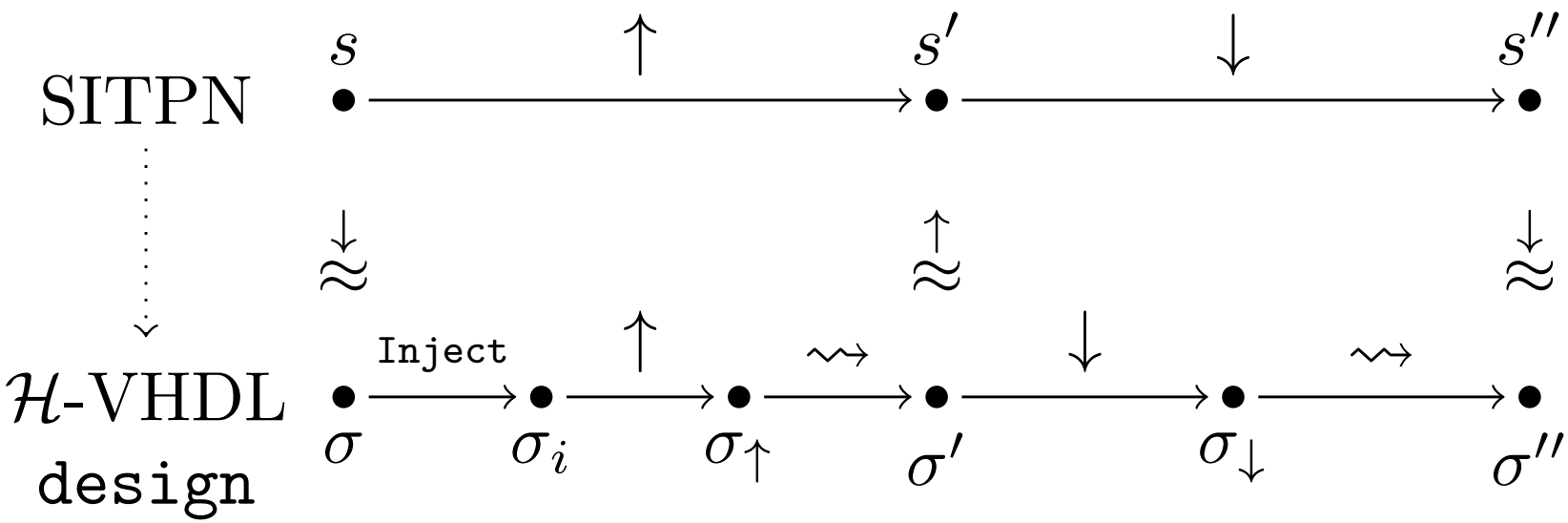
Par induction sur la structure des traces d'exécution.

1. Prouver que les états initiaux sont similaires.
2. Prouver que si on part d'états similaires en début de cycle d'horloge, on arrive à des états similaires au prochain cycle d'horloge.

Preuve sur un cycle d'horloge

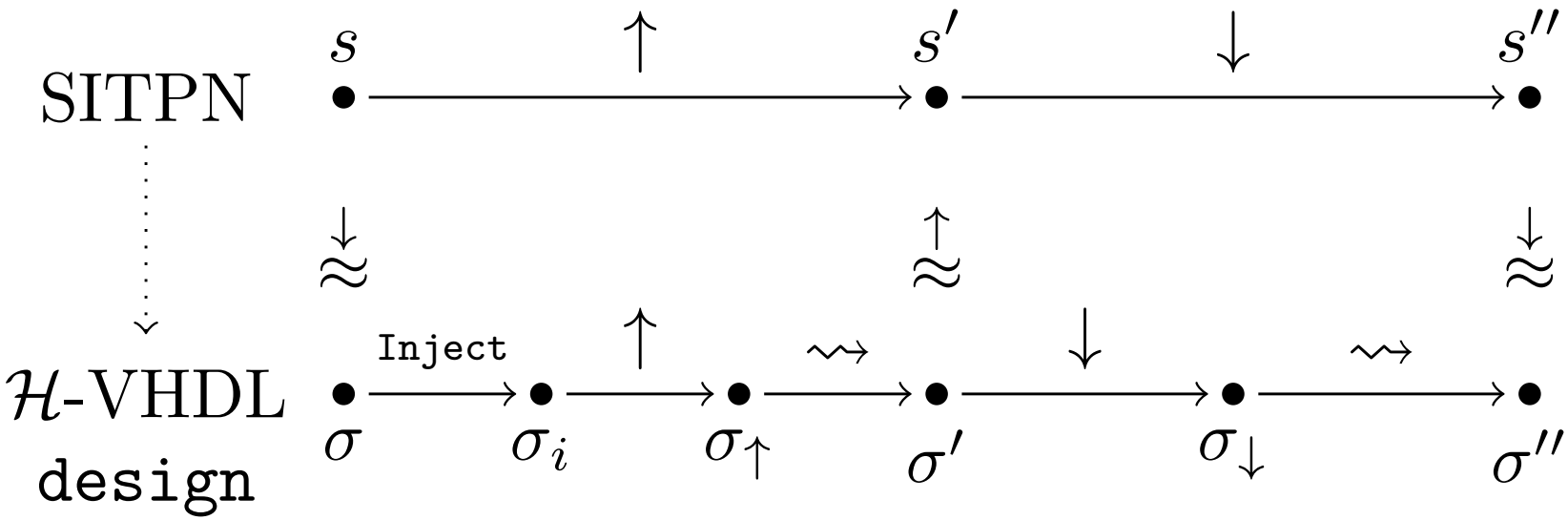


Preuve sur un cycle d'horloge



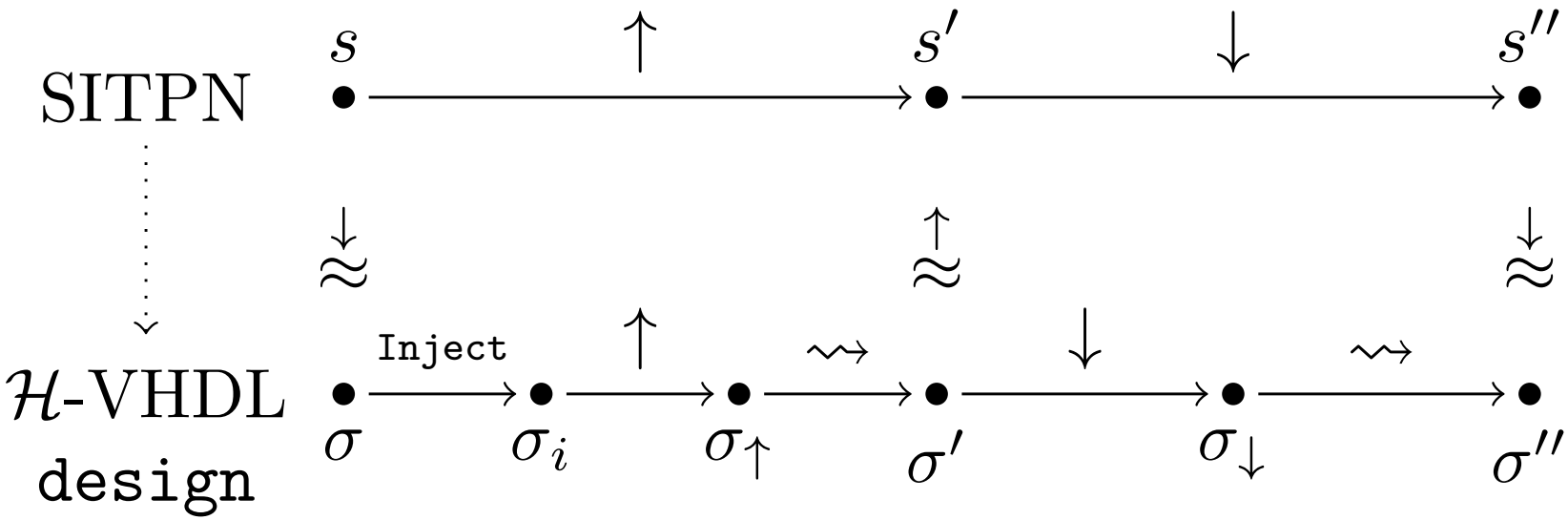
1. Côté SITPN : raisonner sur la relation de transition d'états.

Preuve sur un cycle d'horloge



1. Côté SITPN : raisonner sur la relation de transition d'états.
2. Raisonner sur la fonction de transformation.

Preuve sur un cycle d'horloge

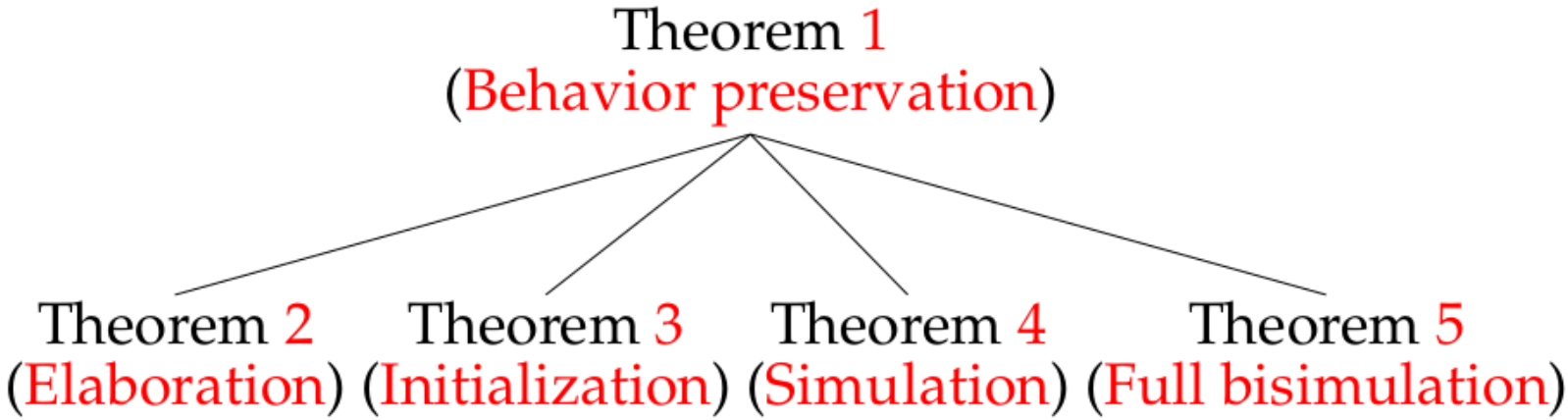


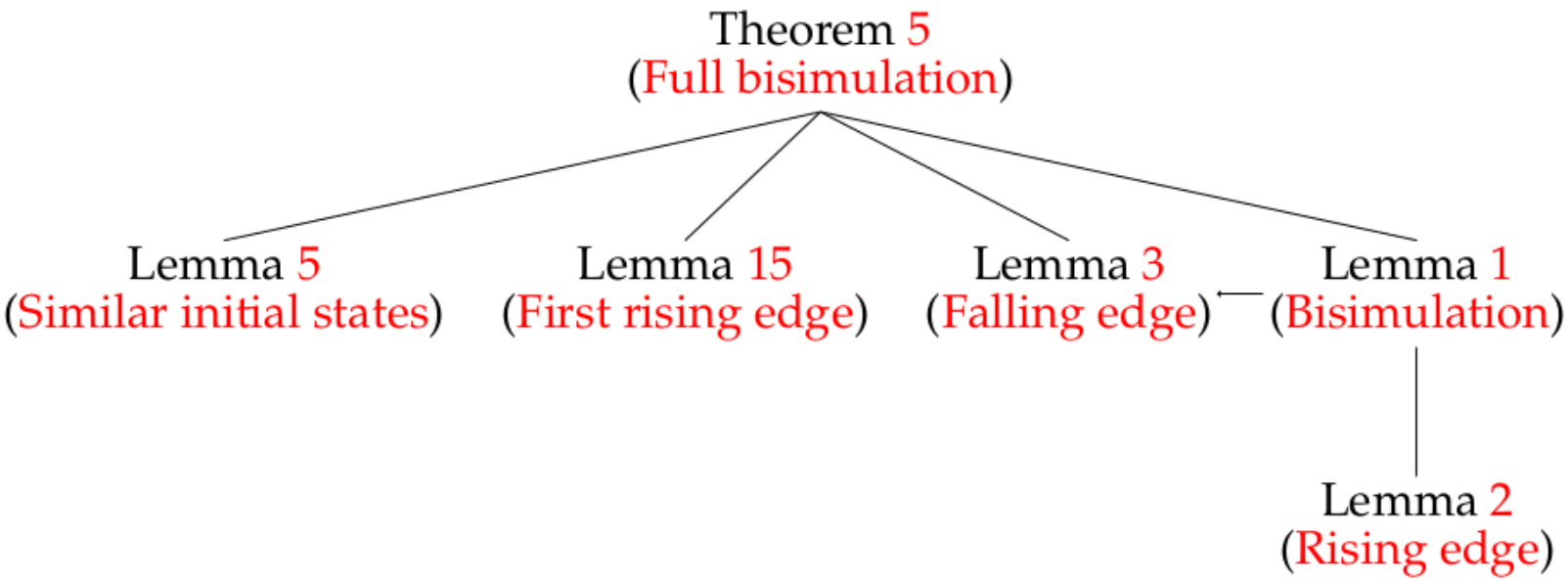
1. Côté SITPN : raisonner sur la relation de transition d'états.
2. Raisonner sur la fonction de transformation.
3. Côté $\mathcal{H}$-VHDL : raisonner sur les relations du cycle de simulation.

Preuve papier

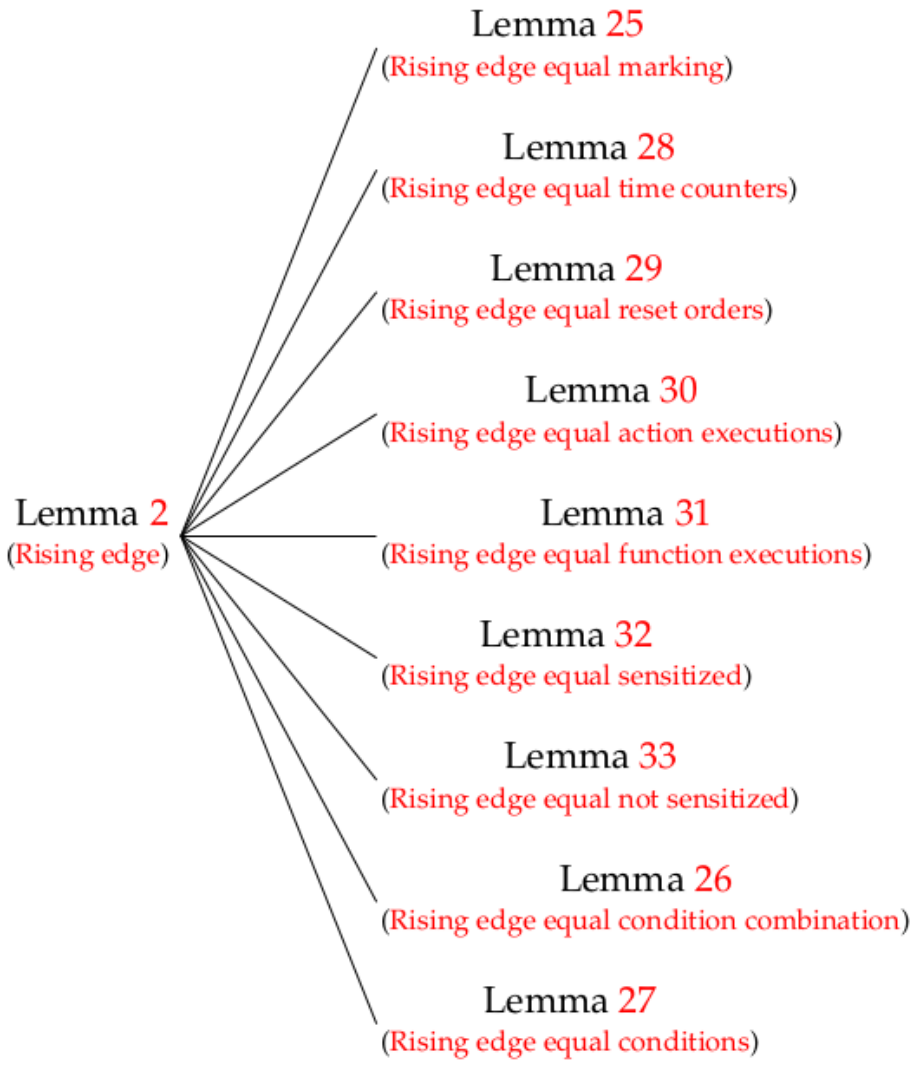
- Preuve papier 100% complétée.
- Fait une centaine de pages.
- Une cinquantaine de théorèmes et lemmes.
- Détection de bug et correction (design `place`).

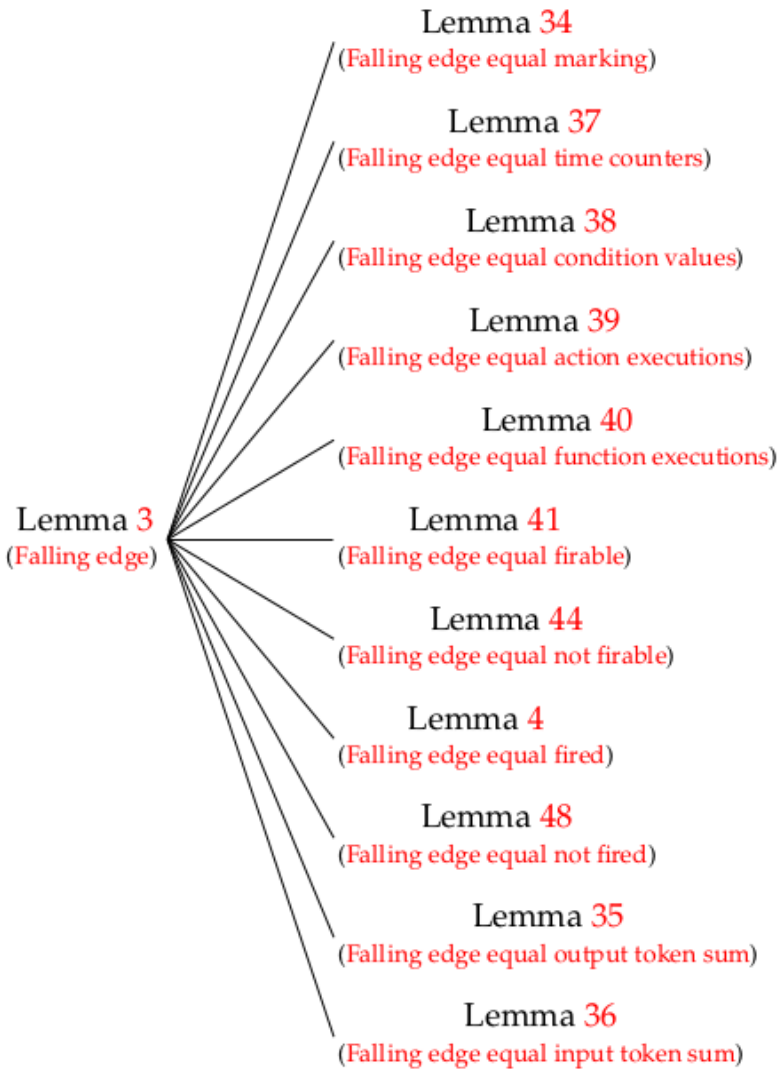
Preuve papier





Preuve papier





Preuve mécanisée

- Mécanisation la preuve papier, 3 mois de travail.

Preuve mécanisée

- Mécanisation la preuve papier, 3 mois de travail.
- Grande distance entre papier et code :

  « Par propriété de la transformation, pour toute place p, il existe un composant place id_p t.q. $\gamma(p) = id_p$. »

  $\Rightarrow$ 1500 lignes de preuve en Coq.

Preuve mécanisée

- Mécanisation la preuve papier, 3 mois de travail.
- Grande distance entre papier et code :
- Points les plus complexes de la mécanisation :
  - Prouver les propriétés de la fonction de transformation.
  - Raisonner sur la valeur des signaux VHDL.

Plan

Objectifs & Contributions

Objectif

Preuve du théorème de préservation de comportement
$\Rightarrow$ OK sur papier ✓

Autres contributions

- Implémentation de la sémantique des SITPNs.
- Formalisation et implémentation d'une sémantique synchrone pour VHDL.
- Implémentation de la transformation modèle-vers-texte HILECOP.

Objectifs & Contributions

Question de recherche

Peut-on appliquer les techniques de vérification de compilateurs au cas HILECOP ?

- Sémantiques spécifiques pour source/cible.
- Relation de similarité entre états.
- Comparaison de traces d'exécutions.

Objectifs & Contributions

Métriques sur l'implémentation

- Une centaine de fichiers.
- 9000 lignes de programmes et spécifications.
- 7000 lignes de preuves.

Perspectives

Challenges scientifiques

- Gestion des exceptions (macroplaces).

Perspectives

Challenges scientifiques

- Gestion des exceptions (macroplaces).
- Multiples domaines d'horloge.

Perspectives

Challenges scientifiques

- Gestion des exceptions (macroplaces).
- Multiples domaines d'horloge.

Challenges d'ingénierie

- Finaliser la mécanisation de la preuve.
- Planification : 2 personnes pendant 1 an.
- Post-doc + ingénieur de recherche.

Merci de votre attention !

Etat de l'art : vérification formelle de transformations

Compilateurs pour langages de programmation génériques

- CompCert, langage C (S. Blazy, X. Leroy, 2005).
- Langage Java (Martin Strecker, 2002).
- Langage fonctionnel (A. Chlipala, 2010), CakeML (Y. Tan, 2016).

Compilateurs pour langages de description de circuits

- Verilog (Lööw, 2021), Lustre (Bourke, 2017), SystemC (Habibi, Tahar, 2006).

Transformations de modèles

- Réseaux de Petri vers LLVM (Fronc, Pommereau, 2011), AADL (Architecture Analysis and Design Language) vers ASM (Abstract State Machines) (Yang et al., 2014).

Exécution d'une instance de composant sur front montant

CompRE

$$\frac{\begin{array}{c}\Delta, \Delta_c, \sigma, \sigma_c \vdash \mathrm{i} \xrightarrow{\mathit{mapip}} \sigma'_c \\ \mathcal{D}, \Delta_c, \sigma'_c \vdash \mathcal{D}(\mathit{id}_e).\mathrm{cs} \xrightarrow{\uparrow} \sigma''_c \\ \Delta, \Delta_c, \sigma, \sigma''_c \vdash \mathrm{o} \xrightarrow{\mathit{mapop}} \sigma'\end{array}}{\mathcal{D}, \Delta, \sigma \vdash \texttt{comp}\ (\mathrm{id}_c,\ \mathrm{id}_e,\ \mathrm{g},\ \mathrm{i},\ \mathrm{o}) \xrightarrow{\uparrow} \sigma''}$$

$\mathit{id}_e \in \mathcal{D}$

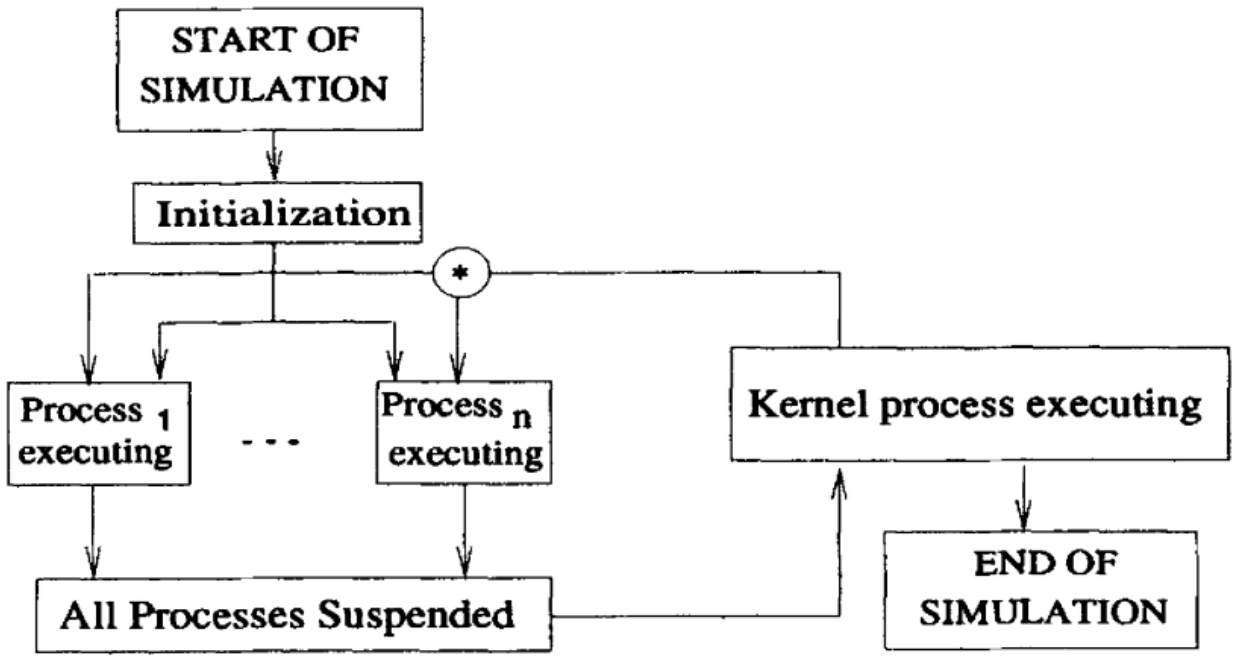
$\Delta(\mathrm{id}_c) = \Delta_c,\ \sigma(\mathrm{id}_c) = \sigma_c$

$\sigma'' = <\mathcal{S}', \mathcal{C}'', \mathcal{E}' \cup (\mathcal{C} \overset{\neq}{\cap} \mathcal{C}'')>$

$\mathcal{C}'' = \mathcal{C}'(\mathrm{id}_c) \leftarrow \sigma''_c$

La sémantique de VHDL

- Algorithme de simulation (Manuel de Référence).
- Deux paradigmes temporels :
  - Instruction avec retard : `s1` $\Leftarrow$ `a and b after 3ms;`
  - Instruction sans retard : `s1` $\Leftarrow$ `a and b`
- Illustration extraite de (Börger, 1995).



Evaluation de l'instruction d'affectation de signal

SigAssign

$$\frac{\Delta, \sigma, \Lambda \vdash \mathrm{e} \xrightarrow{e} v \qquad v \in_c T}{\Delta, \sigma, \sigma_w, \Lambda \vdash \mathrm{id}_s \Leftarrow \mathrm{e} \xrightarrow{ss} <\mathcal{S}'_w, \mathcal{C}_w, \mathcal{E}'_w>, \Lambda}$$

$$\mathrm{id}_s \in \mathit{Sigs}(\Delta) \cup \mathit{Outs}(\Delta)$$

$$\Delta(\mathrm{id}_s) = T$$

$$\mathcal{S}'_w = \mathcal{S}_w(\mathit{id}_s) \leftarrow v$$

$$\mathcal{E}'_w = \mathcal{E}_w \cup (\mathcal{S} \overset{\neq}{\cap} \mathcal{S}'_w)$$

```
| VSeqSigAssignEvent :
    forall flag id e newv t currv σ'_w,
      (* * Premises * *)
      vexpr Δ σ Λ false e newv →
      is_of_type newv t →
      is_of_type currv t →
      (* * Side conditions * *)
      (MapsTo id (Declared t) Δ ∨ MapsTo id (Output t) Δ) →
      MapsTo id currv (sigstore σ) →
      OVEq newv currv (Some false) →
      σ'_w = (events_add id (sstore_add id newv σ_w)) →

      vseq Δ σ σ_w Λ flag (ss_sig id e) σ'_w Λ
```

Fragment de generate_interconnections

Algorithm 13: `generate_interconnections`$(sitpn,\ d,\ \gamma)$

foreach $p \in P$ **do**
- $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \leftarrow \texttt{get\_comp}(\gamma(p), d.cs)$
- $i \leftarrow 0$
- **foreach** $t \in \texttt{input}(p)$ **do**
  - $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \leftarrow \texttt{get\_comp}(\gamma(t), d.cs)$
  - $i_p \leftarrow i_p \cup \{(\texttt{itf}(i), \texttt{actual}(\texttt{fired}, o_t))\}$
  - $i \leftarrow i + 1$
- $i \leftarrow 0$
- ...
- **foreach** $t \in \texttt{output}_{nc}(p)$ **do**
  - $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \leftarrow \texttt{get\_comp}(\gamma(t), d.cs)$
  - $i_p \leftarrow i_p \cup \{(\texttt{otf}(i), \texttt{actual}(\texttt{fired}, o_t))\}$
  - $\texttt{connect}(o_p, i_t, \texttt{oav}(i), \texttt{iav}, d)$
  - $\texttt{connect}(o_p, i_t, \texttt{rtt}(i), \texttt{rt}, d)$
  - $id_s \leftarrow \texttt{genid}()$
  - $d.sigs \leftarrow d.sigs \cup (id_s, \texttt{boolean})$
  - $o_p \leftarrow o_p \cup \{(\texttt{pauths}(i), id_s)\}$
  - $\texttt{cassoc}(i_t, \texttt{pauths}, \texttt{true})$
  - $\texttt{put\_comp}(id_t, \texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t), d.cs)$
  - $i \leftarrow i + 1$
- $\texttt{put\_comp}(id_p, \texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p), d.cs)$

Théorème de préservation de comportement

Pour tout modèle SITPN *sitpn* et design $\mathcal{H}$-VHDL *d* résultant de la transformation modèle-vers-texte HILECOP, si l'exécution de *sitpn* pendant τ cycles d'horloge donne une trace θ_s, alors il existe une trace θ_σ produite par l'exécution de *d* qui est similaire à θ_s.

```
Theorem sitpn2vhdl_semantic_preservation :
forall sitpn d γ E_c τ θ_s,

IsWellDefined sitpn →
sitpn_to_hvhdl sitpn = (d, γ) →
SitpnFullExec sitpn E_c τ θ_s →

exists θ_σ, hfullsim E_p τ d θ_σ ∧ FullSimTrace γ θ_s θ_σ.
```

Gestion des exceptions (macroplaces)