THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM

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

Présenté par Vincent IAMPIETRO

Le Date de la soutenance

Sous la direction de David Delahaye
et David Andreu

Devant le jury composé de

[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

List of Abbreviations

SITPN	**S**ynchronously executed **I**nterpreted **T**ime **P**etri **N**et with priorities
VHDL	**V**ery high speed integrated circuit **H**ardware **D**escription **L**anguage
CIS	**C**omponent **I**nstantiation **S**tatement
PCI	**P**lace **C**omponent **I**nstance
TCI	**T**ransition **C**omponent **I**nstance
GPL	**G**eneric **P**rogramming **L**anguage
HDL	**H**ardware **D**escription **L**anguage
LRM	**L**anguage **R**eference **M**anual
DSL	**D**omain **S**pecific **L**anguage
MDE	**M**odel-**D**riven **E**ngineering

For/Dedicated to/To my...

Chapter 1

Preliminary Notions

In this chapter, we introduce the mathematical formalisms and notations used throughout this thesis to express and formalize our ideas. In Section 1.3, we provide the basics to understand the Coq proof assistant, which is the system we use to write our programs and to mechanize our proofs. This chapter is inspired by the book *The Formal Semantics of Programming Languages: an Introduction* [24], the courses of the University of Cambridge on the semantics of programming languages[1], the documentation of the Coq proof assistant[2], and the book *Certified Programming with Dependent Types* [9].

1.1 Mathematical notations

1.1.1 Intuitionistic first order logic

The intuitionistic first-order logic [18] constitutes our framework for the expression and the interpretation of logical formulas. The language to express logical formulas is the same between classical and intuitionistic first-order logic. A logical formula is either:

- $\bot$ (bottom), the always *false* formula, or $\top$ (top), the always *true* formula
- a predicate (i.e. an atomic formula). A predicate P possibly takes n parameters as inputs. We write $P(x_0, \ldots, x_n)$ to denote an n-ary predicate P applied to the inputs $x_0, \ldots, x_n$.
- the composition of two subformulas with one of the following binary operators: the conjunction $\wedge$, the disjunction $\vee$, the implication $\Rightarrow$, the double implication $\Leftrightarrow$
- the composition of one subformula with the negation operator $\neg$
- a subformula prefixed by the universal quantifier $\forall$ or the existantial quantifier $\exists$. For instance, the formula $\forall x.P(x)$ denotes the atomic formula $P(x)$ where the parameter x is a universally quantified variable of the formula. As a shorthand notation, we write $\forall x, y, z. \ldots$ to denote $\forall x, \forall y, \forall z. \ldots$. The same stands for the existantial quantifier $\exists$. Variables that are introduced in a logical formula by one of the previous quantifiers are called the *bound* variables of the formula. Variables that appear in a logical formula without being introduced by

[1] https://www.cl.cam.ac.uk/teaching/2021/Semantics/
[2] https://coq.inria.fr/distrib/current/refman/index.html

a quantifier are called *free* variables. For instance, in the logical formula $\forall x.\ P(x) \wedge Q(y)$, x is a bound variable and y is a free variable of the formula.

The difference between the classical first-order logic and the intuitionistic one relies in the absence of the *law of the excluded middle* in the latter logic. In classical logic, the *law of the excluded middle* considers that a formula can always be either interpreted as true or false, i.e. a formula is always *decidable* regardless the valuation of its inputs. In intuitionistic logic, such an assumption is discarded. Thus, a formula of the intuitionistic logic can be *undecidable*, i.e. given a valuation of its input, one can not always state that the formula is true or false. The consequence is that one as to exhibit a *explicitly-built* proof to show that a given formula is true. One can not rely on the law of excluded middle to perform a proof by contradiction. As so, the intuitionistic logic is said to be *constructive* in the sense that one as to *construct* a concrete proof object to show that a formula is true. The intuitionistic first-order logic is built in the Coq proof assistant. Thus, a logic formula expressed with the Coq proof assistant is an intuitionistic formula by default. For this reason, we choose the intuitionistic first-order logic as our mathematical framework for the expression of logic formulas.

In what follows, we often give the definition of a predicate by relating its semantics to the semantics of an underlying formula. For instance, we define the predicate $IsEven(n)$, for all $n \in \mathbb{N}$, as follows: $IsEven(n) \equiv \exists k \in \mathbb{N}\ s.t.\ 2k = n$.

1.1.2 Set theory

In this thesis, we use set theory as the base formalism for all our mathematical definitions and proofs. In set theory, a set represents a group of elements called the members of the set. For every set X, we write $x \in X$ to denote that the element x is a member of set X. We note $X \subseteq Y$ the fact that a set X is a subset of set Y such that for all x if $x \in X$ then $x \in Y$. From here, there are multiple ways to define a set.

Extensional definition

A set is defined by *extension* with the enumeration of all its members. For instance, $\{1, 0, -1\}$, $\{a, b, c\}$ or $\{p_0, \ldots, p_n\}$ are all sets defined by extension.

Intensional definition

The intensional definition of a set specifies a given property verified by all the members of the set. For instance, here is the intensional definition of the set of even numbers: $\{n \in \mathbb{N} \mid \exists k \in \mathbb{N}\ s.t.\ n = 2k\}$ or $\{n \in \mathbb{N} \mid IsEven(n)\}$.

Set operators

Set theory also defines some operators to compose sets together. Given two sets A and B, the following sets are formed:

- $A \cup B$ denotes the set formed by the union of the members of A and the members of B, i.e. $A \cup B = \{x \mid x \in A \vee x \in B\}$.

- $A \cap B$ denotes the set formed by the intersection of set A and B, i.e. $A \cap B = \{x \mid x \in A \wedge x \in B\}$.
- $A \setminus B$ denotes the set formed by the elements of set A that are not elements of set B (the difference between set A and B), i.e. $A \setminus B = \{x \mid x \in A \wedge x \notin B\}$.
- $A \times B$ denotes the cartesian product between the elements of set A and set B, i.e. the set of all ordered pairs defined by $\{(x, y) \mid x \in A \wedge y \in B\}$. We generalize the definition to build the set of n-tuples $A_0 \times A_1 \times \cdots \times A_n$ defined by $\{(x_0, (x_1, \ldots (\ldots, x_n))) \mid x_0 \in A_0, x_1 \in A_1, \ldots, x_n \in A_n\}$. It is sometimes useful to give a name to the elements of a tuple without refering to their index. In such a case, a tuple is called a record where each element, called a field, has been given an explicit name. This formalism is useful to represent rather complex data structures. For instance, say that we want to represent the set of humans by a triplet composed of the size, weight, and eye color of a given human. We can define this set as the set of triplet $\mathbb{R} \times \mathbb{R} \times \{green, blue, brown\}$. If we want to give a concrete name to the elements of the triplet, we can equivalently define such a triplet as a record, written $<size, weight, eye>$, where $size \in \mathbb{R}$, $weight \in \mathbb{R}$ and $eye \in \{green, blue, brown\}$.
- $A \sqcup B$ denotes the set formed by the disjoint union of set A and set B. The disjoint union is obtained by adjoining an index i to the elements of A and an index j to the elements of B such that $i \neq j$. Then, the two sets of couples are joined together to build the disjoint union of A and B. For instance, consider that $A = \{a, b, c\}$ and $B = \{a, b\}$. To obtain the disjoint union $A \sqcup B$, we create the two sets $A_i = \{(i, a), (i, b), (i, c)\}$ and $B_i = \{(j, a), (j, b)\}$, and then join the sets together s.t. $A \sqcup B = \{(i, a), (i, b), (i, c), (j, a), (j, b)\}$. When the two sets A and B are disjoint, i.e. $A \cap B = \emptyset$, then $A \sqcup B$ is isomorphic to $A \cup B$. To stress the fact that we are building a set from the union of two disjoint sets, we prefer to use the disjoint union operator. For instance, we write $\mathbb{N} \sqcup \{\infty\}$, instead of $\mathbb{N} \cup \{\infty\}$, to denote the set of values ranging from the set of natural numbers with the addition of the infinite value ∞.
- $\mathcal{P}(A)$ denotes the powerset of A defined by all the possible subsets formed with the elements of set A, i.e. $\mathcal{P}(A) = \{X \mid X \subseteq Y\}$.

Relations and functions

A binary relation R between two sets X and Y is a subset of the set of pairs $X \times Y$, i.e. $R \subseteq X \times Y$, or an element of the powerset $\mathcal{P}(X \times Y)$, i.e. $R \in \mathcal{P}(X \times Y)$. We write $R(x, y)$ to denote $(x, y) \in R$. We generalize the definition to n-ary relations. A n-ary relation between sets $X_0, \ldots, X_n$ is a subset of the set of n-tuples $X_0 \times \cdots \times X_n$, i.e. $R \subseteq X_0 \times \cdots \times X_n$, or an element of the powerset $\mathcal{P}(X_0 \times \cdots \times X_n)$, i.e. $R \in \mathcal{P}(X_0 \times \cdots \times X_n)$. We write $R(x_0, \ldots, x_n)$ to denote $(x_0, \ldots, x_n) \in R$.

A partial function f from set X to set Y is a binary relation from X to Y verifying that $\forall x \in X, y, y' \in Y,\ (x, y) \in f \wedge (x, y') \in f \Rightarrow y = y'$, i.e. x appears at most once as the first element of a pair in f. We note $f \in X \nrightarrow Y$ to denote a partial function from X to Y. The set of the first elements of the pairs defined in f is called the *domain* of f. We write it $\texttt{dom}(f) = \{x \mid \exists y \ s.t. (x, y) \in f\}$. When there is no ambiguity, and given an $x \in X$ and

$f \in X \nrightarrow Y$, we write $x \in f$ as a shorthand to $x \in \texttt{dom}(f)$.

A total function a, or application, from X to Y is a partial function verifying that all the elements of X appear as the first element of a pair in a, i.e. for all $x \in X$, there exists $y \in Y$ such that $(x, y) \in a$. In other words, the domain of an application from X to Y is equal to the set X. We note $a \in X \to Y$ to denote an application from X to Y.

1.1.3 Rule-based definition of sets

While describing the operational semantics of a subset of the VHDL language in Chapter **??**, we define some of the sets (e.g, simulation relations) with rule instances, also called inference rules or judgments. A rule instance, defining the members of a set A, can take the following forms:

$$\frac{}{C} \text{ or } \frac{P_1, \ldots, P_n}{C}$$

The left form of rule instance is called an axiom; it states that $C \in A$. The rule instance $\frac{P_1, \ldots, P_n}{C}$ states that $C \in A$ if $P_1, \ldots, P_n$ hold. Also, we say that C is derivable, if $P_1, \ldots, P_n$ are derivable. $P_1, \ldots, P_n$ are the premises of the rule, and C is the conclusion of the rule. The premises $P_1, \ldots, P_n$ hold if there exists a *finite* derivation tree for each one of them. A *finite* derivation tree is obtained by applying the rule instances defining the set until all branches of the derivation reach axioms. For instance, the two following rules, named rules Ev0 and Ev2, define the *IsEven* unary relation that states that a given natural number is even:

$$\frac{}{IsEven(0)}\ \text{Ev0} \qquad \frac{IsEven(n-2)}{IsEven(n)}\ \text{Ev2}$$

The rule Ev0 states as an axiom that 0 is an even number; the rule Ev2 states that for all natural number n, n is an even number if one can derive that fact that $n - 2$ is an even number. Thus, we can derive from the previous rules that 4 is an even number by building the following derivation tree:

$$\dfrac{\dfrac{\dfrac{}{IsEven(0)}\ \text{Ev0}}{IsEven(2)}\ \text{Ev2}}{IsEven(4)}\ \text{Ev2}$$

Starting from $IsEven(4)$, we can apply the Ev2 rule to derive $IsEven(2)$. Then, another application of the Ev2 rule leads to $IsEven(0)$, and we can close the derivation branch by applying the Ev0 rule. Here, the only branch of the tree has reached an axiom, and thus the derivation tree is finite. To further illustrate the use of rule instances in the definition of a set, let us consider the following minimal language of arithmetic expressions expressed in the Backus-Naur form:

$$e \quad ::= \quad n \mid id \mid e_0 + e_1$$

Here, n ranges over the set of natural numbers $\mathbb{N}$; id ranges over the set `string` of non-empty strings (i.e. it is the set of identifiers). To evaluate the arithmetic expressions, we need a state $s \in \texttt{string} \rightharpoonup \mathbb{N}$ that maps each variable identifier to a natural number value. We assume that only a certain set of declared identifiers can appear in an arithmetic expression. Thus, the state is a partial function from the set of non-empty strings to the set of natural numbers. We define the evaluation relation for the arithmetic expressions with the three following rules:

$$\frac{}{s \vdash n \to n}\ \textsc{Nat} \qquad \frac{}{s \vdash id \to s(id)}\ \textsc{Var}\ id \in \texttt{dom}(s) \qquad \frac{s \vdash e_0 \to n \qquad s \vdash e_1 \to m}{s \vdash e_0 + e_1 \to n + m}\ \textsc{Add}$$

Here, the evaluation relation is a subset of the set of triplets $(\texttt{string} \rightharpoonup \mathbb{N}) \times e \times \mathbb{N}$. In the rule instances defining the evaluation relation, the $\vdash$ symbol (pronounced *thesis*) stresses that the left part implies the right part, or it is involved in the evaluation of the right part. For instance, the second rule can be read: in the context of state s, id evaluates to $s(id)$ if $id \in \texttt{dom}(s)$. When the *context* is not involved in the evaluation of the syntactic constructs at the right side of the $\vdash$ symbol, we permit ourselves to remove the context and the $\vdash$ from the rule instances. For example, we can define to the NAT rule by

$$\frac{}{n \to n}\ \textsc{Nat}$$

as the state s in not involved in the evaluation of expressions that are natural numbers.

Note that in the VAR rule, there appears an extra statement, at the right of the judgment line, called a *side condition*. This is an extra condition that must hold with all the premises of the rule instance, but which does not generate a derivation tree of its own.

Finally, here is an example of a derivation tree for the evaluation of the expression $x + (y + 1)$ in the context of state $\{(x, 1), (y, 2)\}$:

$$\dfrac{\dfrac{}{\{(x,1),(y,2)\} \vdash x \to 1}\ \textsc{Var}\ x \in \{x,y\} \qquad \dfrac{\dfrac{}{\{(x,1),(y,2)\} \vdash y \to 2}\ \textsc{Var}\ y \in \{x,y\} \qquad \dfrac{}{1 \to 1}\ \textsc{Nat}}{\{(x,1),(y,2)\} \vdash y + 1 \to 3}\ \textsc{Add}}{\{(x,1),(y,2)\} \vdash x + (y + 1) \to 4}\ \textsc{Add}$$

Here again, all the branches of the derivation tree have reached axioms, and thus $\{(x, 1), (y, 2)\} \vdash x + (y + 1) \to 4$ is a member of the evaluation relation of arithmetic expressions. It states that the expression $x + (y + 1)$ evaluates to 4 in the state $\{(x, 1), (y, 2)\}$.

1.2 Induction principles

In the proceedings of proofs laid out in this thesis, we often perform proofs by induction. Here are some reminders on some induction principles to help the reader understand the proofs of Chapter 2 and Appendix A.

1.2.1 Mathematical induction

The principle of mathematical induction is tied to the set of natural numbers. It states that, in order to prove that a property P holds for all natural numbers, it is sufficient to prove that:

- P holds for 0
- if P holds for a given n then it holds at $n+1$

Mathematical induction describes a way to deduce that a property holds for the set of natural numbers, first by stating that the property holds for zero, i.e. the minimal element of the set, then by stating that if the property holds for a given number then it holds for its successor. Thus, knowing that $P(0)$ holds, we can deduce that $P(1)$ holds, $P(2)$ holds, $P(3)$ holds, etc.

1.2.2 Structural induction

Sometimes, reasoning by induction necesitates to follow the structure of a given set, i.e. the formation rules of a given set. For instance, if we want to prove that a given property P holds for the set of arithmetic expressions given as an example in Section 1.1.3, we must prove that:

- P holds for all natural number n
- P holds for all identifiers id
- if P holds for all sub-expressions e_0 and e_1, then P holds for $e_0 + e_1$

A proof that leverages structural induction follows the structure of the elements we are reasoning upon. Many times in this thesis, we are using structural induction to prove that a sum expression verifies a certain property. Thus, the structural induction follows the recursive definition of the sum term, which is, for any set A, function $f \in A \to \mathbb{N}$ and $X \subseteq A$ and :

$$\sum_{x \in X} f(x) = \begin{cases} 0 \texttt{ if } X = \emptyset \\ f(x) + \sum\limits_{x' \in X'} f(x') \texttt{ if } X = \{x\} \cup X' \end{cases}$$

In the second computation branch, it is left implicit that set X' is strict subset of X such that $x \notin X'$ or $X' = X \setminus \{x\}$. Given a set A and a function $f \in A \to \mathbb{N}$, to prove that for all $X \subseteq A$, the property $P(X, \sum\limits_{x \in X} f(x))$ holds, we must show that:

- $\forall X \subseteq A,\ X = \emptyset \Rightarrow P(\emptyset, 0)$

- $\forall X \subseteq A, x \in X, X' \subset X$,
  $X = \{x\} \cup X' \Rightarrow P(X', \sum_{x' \in X'} f(x')) \Rightarrow P(\{x\} \cup X', f(x) + \sum_{x' \in X'} f(x'))$

The induction follows the structure of the function. In this specific case, structural induction is often refered to as *functional* induction. Let us prove Proposition 1 to illustrate the use of structural induction over a sum term:

Proposition 1. *For all $X \subset \mathbb{N}$ a finite set of natural numbers, $\sum_{x \in X} 2x$ is even, i.e.*

$$\exists k \in \mathbb{N} \; s.t. \; \sum_{x \in X} 2x = 2k$$

Proof. Let us define the property P as follows:

$$P(X, \sum_{x \in X} 2x) \equiv \exists k \in \mathbb{N} \text{ s.t. } \sum_{x \in X} 2x = 2k$$

Then, let us use structural induction to prove $P(X, \sum_{x \in X} 2x)$.

First, let us show $P(\emptyset, 0)$, i.e. $\exists k \in \mathbb{N}$ s.t. $0 = 2k$. Let us take $k = 0$ to build a tautology.

Then, given a $X' \subset X$ and a $x \in X$ s.t. $X = \{x\} \cup X'$, and assuming that $P(X', \sum_{x' \in X'} 2x')$ holds (i.e. the induction *hypothesis*), let us show $P(\{x\} \cup X', 2x + \sum_{x' \in X'} 2x')$. Appealing to the induction hypothesis, let us take a j such that $\sum_{x' \in X'} 2x' = 2j$. Rewriting $\sum_{x' \in X'} 2x'$ as $2j$:

$\Rightarrow \exists k \in \mathbb{N}$ s.t. $2x + 2j = 2k$

$\Rightarrow \exists k \in \mathbb{N}$ s.t. $2(x + j) = 2k$

$\Rightarrow$Then, let us take $k = x + j$ to obtain a tautology.

□

1.2.3 Rule induction

A specific kind of induction, called *rule* induction, is applied to prove properties over sets that are defined by rule instances. Let us take the evaluation relation for arithmetic expressions used in Section 1.1.3 to illustrate the principle of rule induction. To prove that a property P holds for the evaluation relation of arithmetic expressions, which is a subset of triplets $(\texttt{string} \to \mathbb{N}) \times e \times \mathbb{N}$, we must prove that:

- For all $s \in \texttt{string} \rightharpoonup \mathbb{N}, n \in \mathbb{N}$, $P(s, n, n)$
- For all $s \in \texttt{string} \rightharpoonup \mathbb{N}, id \in \texttt{string}$, if $id \in \texttt{dom}(s)$ then $P(s, id, s(id))$
- For all $s \in \texttt{string} \rightharpoonup \mathbb{N}, e_0, e_1 \in e, n, m \in \mathbb{N}$,
  if $s \vdash e_0 \to n$ and $P(s, e_0, n)$, and $s \vdash e_1 \to m$ and $P(s, e_1, m)$
  then $P(s, e_0 + e_1, n + m)$

Rule induction states that in order to prove a property over a set defined by rule instances, the property must hold in any construction case of the considered set. The idea is that if the property is preserved from the premises of rules to the conclusions then the property holds for all the elements of the set.

Let us give an application of rule induction to prove a property over the evaluation relation of arithmetic expressions. First, we define, through the three following rules, the relation $\in_r$ stating that a given identiifer id is referenced in an arithmetic expression e, written $id \in_r e$:

$$\frac{}{id \in_r id}\ \textsc{InRId} \qquad \frac{id \in_r e_0}{id \in_r e_0 + e_1}\ \textsc{InRAddL} \qquad \frac{id \in_r e_1}{id \in_r e_0 + e_1}\ \textsc{InRAddR}$$

Then, the property of Proposition 2 states that an arithmetic expression that contains references to identifiers that are not part of the current state's domain can not be evaluated.

Proposition 2. *Let $id \in \texttt{string}$. For all state s, arithmetic expression e, and natural number n,*

$$id \notin \texttt{dom}(s) \wedge id \in_r e \Rightarrow \neg s \vdash e \rightarrow n$$

Proof. Let us define the property P as follows:

$$P(s, e, n) \equiv id \notin \texttt{dom}(s) \wedge id \in_r e \Rightarrow \neg s \vdash e \rightarrow n$$

Then, let us use rule induction to prove $P(s, e, n)$.

First, we must prove $P(s, n, n)$. Assuming $id \in_r n$, there is a contradiction as no rule instance defining the relation $\in_r$ includes the case where the considered expression is a natural number.

Then, we must prove $P(s, id', s(id'))$, assuming that $id' \in \texttt{dom}(s)$. We know that $id \in_r id'$, and thus $id = id'$. Then, there is a contradiction between $id \in \texttt{dom}(s)$ and $id \notin \texttt{dom}(s)$.

Finally, we must prove $P(s, e_0 + e_1, n + m)$, assuming that $s \vdash e_0 \rightarrow n$ and $P(s, e_0, n)$, and $s \vdash e_1 \rightarrow m$ and $P(s, e_1, m)$. We know that $id \in_r e_0 + e_1$; this hypothesis has either be constructed by applying Rule InRAddL or Rule InRAddR. If Rule InRAddL has been applied, then we know $id \in_r e_0$; thus, from $P(s, e_0, n)$, we can deduce $\neg s \vdash e_0 \rightarrow n$, which contradicts $s \vdash e_0 \rightarrow n$. We can perform the proof in a similar way if Rule InRAddR has been applied. □

1.3 The Coq proof assistant

In this section, we present the Coq proof assistant [22]. The Coq proof assistant constitutes our framework to encode the different semantics, programs and proofs involved in the verification of the HILECOP model-to-text transformation. Here, we give an overview of the different concepts underlying the Coq proof assistant. The aim is to give to the reader the tools to understand the different listings presenting Coq code in the following chapters. For a thorough presentation of the Coq proof assistant, the reader can refer to the Coq reference manual[3], or to [9, 19, 2].

[3] https://coq.inria.fr/distrib/current/refman/

1.3.1 The Calculus of Inductive Constructions (CIC)

The kernel of the Coq proof assistant implements the Calculus of Inductive Constructions (CIC) [11]. The CIC is a typed lambda-calculus extended with the possibility to define inductive types. Thus, the CIC permits to define programs and types in a similar way; both are *terms* of the language. A program is a term with a certain type, and a type is also a term with a certain type. The type of a type is called a *sort*. We can mention two basic sorts built in the Coq proof assistant: the `Prop` sort which is the type of logical formulas, and the `Set` sort which is the type of *small* sets.

The Coq proof assistant permits to express logic formulas and to interactively build proofs of these formulas by using a high-level tactic language. The sequence of tactics that builds a proof for a given formula is called a *proof script*. The execution of a proof script builds a proof term. In the CIC, a logic formula can be seen as a *type* and a proof of this formula is an *inhabitant* of the type denoted by the logic formula. Thus, when building a proof term by executing a proof script, the Coq kernel checks that the proof term is of the type of the logic formula by applying typing rules[4]. For instance, let us take two logical propositions `A` and `B`. In Coq, we can declare these propositions as elements of the `Prop` type in the Coq top-level loop[5]:

```
Coq < Variables A B : Prop.
```

The `Variables` keyword adds the propositions `A` and `B` to the global environment accessed by the Coq kernel. Now, say that we want to prove the *modus ponens* theorem expressed with the propositions `A` and `B`, namely that $\mathtt{A} \Rightarrow (\mathtt{A} \Rightarrow \mathtt{B}) \Rightarrow \mathtt{B}$. In Coq, we can express it as follows:

```
Coq < Theorem modus_ponens : A → (A → B) → A.
```

Here, we declare the modus ponens theorem as an element of type $\mathtt{A} \to (\mathtt{A} \to \mathtt{B}) \to \mathtt{A}$. The arrows represent functional arrows; in fact, $\mathtt{A} \to \mathtt{B}$ is a notation for the product type $\Pi x : \mathtt{A}.\mathtt{B}$ where x is not referenced in `B`. According to the Curry-Howard correspondence [14], there is an equivalence between a proof term and a program. Thus, a proof term of the logical implication $\mathtt{A} \Rightarrow \mathtt{B}$ is equivalent to a program, or a function, of type $\mathtt{A} \to \mathtt{B}$, i.e. a program that takes an element of type `A` and yields an element of type `B`. Thus, the type $\mathtt{A} \to (\mathtt{A} \to \mathtt{B}) \to \mathtt{A}$ is a valid encoding of the formula $\mathtt{A} \Rightarrow (\mathtt{A} \Rightarrow \mathtt{B}) \Rightarrow \mathtt{B}$.

The `Theorem` keyword triggers the interactive proof mode through which the user will build a proof term for the corresponding formula. A simple proof term for the modus ponens theorem is a function that takes an element x of type `A` and a function f of type $\mathtt{A} \to \mathtt{B}$ as inputs, and yields an element of type `B` by applying the function f to parameter x, i.e. $(f\ x)$. The function takes the form of the following term of the typed lambda-calculus:

$$\lambda(x : \mathtt{A}).\lambda(f : \mathtt{A} \to \mathtt{B}).(f\ x)$$

While passing this *lambda-term* as a proof term of the modus ponens theorem, the Coq kernel checks the well-typedness of the term by building the following derivation tree, which is a simplified version of the full derivation tree according to the typing rules of the CIC:

[4] https://coq.inria.fr/distrib/current/refman/language/cic.html

[5] Coq scripts can be either interpreted or compiled.

$$\dfrac{\dfrac{\dfrac{}{\mathtt{A\ B : Prop}[x : \mathtt{A}, f : \mathtt{A} \to \mathtt{B}] \vdash f\ :\ \mathtt{A} \to \mathtt{B}}\ \textsc{Var} \qquad \dfrac{}{\mathtt{A\ B : Prop}[x : \mathtt{A}, f : \mathtt{A} \to \mathtt{B}] \vdash x\ :\ \mathtt{A}}\ \textsc{Var}}{\mathtt{A\ B : Prop}[x : \mathtt{A}, f : \mathtt{A} \to \mathtt{B}] \vdash (f\ x)\ :\ \mathtt{B}}\ \textsc{App}}{\dfrac{\mathtt{A\ B : Prop}[\mathtt{x} : \mathtt{A}] \vdash \lambda(f : \mathtt{A} \to \mathtt{B}).(f\ x)\ :\ (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}{\mathtt{A\ B : Prop}[] \vdash \lambda(x : \mathtt{A}).\lambda(f : \mathtt{A} \to \mathtt{B}).(f\ x)\ :\ \mathtt{A} \to (\mathtt{A} \to \mathtt{B}) \to \mathtt{B}}\ \textsc{Lam}}\ \textsc{Lam}$$

In the above derivation tree, the global and the local environment are represented at the left of the thesis symbol $\vdash$. The local environment is represented by square brackets. The global environment is represented at the left of the local environment. At the root of the derivation tree, the global environment contains our two previously declared logical propositions `A` and `B`, whereas the the local environment is empty. The application of the LAM rule adds new entry to the local environment; the APP triggers the type-checking of the left and the right part of an application; the VAR rule checks that a term is well-typed if it is referenced as an element of the given type in the global or the local environment.

As said before, the `Theorem` keyword triggers the interactive proof mode. The interactive proof mode will accompany the user to an incremental building of a proof term for the current goal, i.e. the current logic formula we want to prove. Then, to prove the modus ponens theorem, the following interface is first presented to the user:

```
Coq < Theorem modus_ponens : A → (A → B) → B.
1 subgoal

============================
A → (A → B) → B
```

The term under the horizontal bar represents the current goal to prove, i.e. the current formula for which we are building a proof term. Above the horizontal bar are referenced the variables constituting the local environment. At the beginning of the proof, the local environment is empty – and so is the local environemnt at the root of the derivation tree presented above. To build a proof term in interactive mode, the user will then invoke commands called *tactics*. Each tactic invocation corresponds to the invocation of a typing rule of the CIC performed by the Coq kernel. To build a proof term for the modus ponens theorem, the first thing to do is to invoke the LAM rule; this is done by appealing to the `intros` tactic.

```
Coq < intros x.
1 subgoal

x : A
============================
(A → B) → B
```

Here, the user passes to the system the name of the variable that will be introduce in the local environment by the LAM rule, i.e the variable `x`. Then, we repeat the operation, applying the LAM rule a second time to introduce an element of type `A → B` in the environment.

```
Coq < intros f.
1 subgoal

x : A
f : A → B
============================
B
```

Then, based on the local environment, we can build an object of type `B` by applying `f` to the input `x`. We can do it by appealing to the `apply` tactic. The `apply` tactic invokes the APP rule.

```
Coq < apply (f x).
No more subgoals.

Coq < Qed.
```

After the invocation of the `apply` tactic, the proof term for the modus ponens theorem is completely built, thus, the Coq top-level loop displays the message that all goals are completed. Then, we can close the interactive proof mode and store the proof of the modus ponens theorem in the global environment by using the `Qed` keyword.

1.3.2 Inductive types

One of the major strength of the CIC, and therefore of the Coq proof assistant, is the possibility to enrich the global type system with the definition of inductive types. For instance, here is the definition of the type of natural numbers, named `nat`:

```
Inductive nat : Set :=
| O : nat
| S : nat → nat.
```

The `nat` type is of the `Set` sort (remember that the type of a type is called a sort). The `nat` type is defined through two constructors, represented by the pipe-separated entries. The `O` constructor states that zero is a natural number. The `S` constructor takes a natural number as input and yields the succesor to this natural number. This corresponds to the structural definition of natural numbers in Peano's arithmetic. Thus, in this setting, the number 2 is represented by `(S (S O))`, the number 3 by `(S (S (S O))`, etc. The result of the evaluation of an inductive type, declared through the `Inductive` keyword, is the addition of this type and each of its constructors to the global environment accesible by the Coq kernel. Also, a corresponding induction principle is generated at the evaluation of an inductive type definition. For instance, the `nat_ind` induction principle is generated at the evaluation of the `nat` type. The `nat_ind` induction principle is equivalent to the so-called mathematical induction presented in Section 1.2.1. It is a proof term of the logical formula denoting the mathematical induction, i.e.:

```
forall P : nat → Prop, P 0 → (forall n : nat, P n → P (S n)) → forall n : nat, P n
```

Then, the induction principle `nat_ind` can be used to perform mathematical induction in a proof involving natural numbers. For instance, say that we want to prove the following

theorem stating that a natural number elevated at the power 2 is always greater than or equal to itself. We can write as follows:

```
Coq < Theorem ge_pow2 : forall n : nat, n <= n * n.
1 subgoal

============================
forall n : nat, n <= n * n
```

Then, we can use the `nat_ind` induction principle to prove such a theorem. Most conveniently, the built-in `induction` tactic chooses the appropriate induction principle based on the type of its argument. Thus, the following command invokes the `nat_ind` induction principle over the universally quantified variable `n`:

```
Coq < induction n.
2 subgoals

============================
0 <= 0 * 0
subgoal 2 is:
S n <= S n * S n
```

The result of the invocation of the `induction` tactic is a branching in the proof tree. Thus, the system indicates that two subgoals must be prove to complete the proof of the `gt_pow2` theorem. These two subgoals corresponds to the proof of $P(0)$ and the proof that assuming $P(n)$ we can show $P(n+1)$, as agreed with mathematical induction. Here, the property P is defined by $P(n) \equiv n \leq n \times n$. We can use the built-in `lia` tactic, defined in the `Lia` module of the Coq standard library, to solve the two remaining subgoals. The `lia` tactic implements a whole decision procedure to prove theorems involving systems of equalities and inequalities over the set of natural numbers. We can combine the `induction` tactic with the `lia` tactic using the semi-colon operator. Then, the `lia` tactic is applied to all the subgoals generated by the `induction` tactic.

```
Coq < induction n; lia.
No more subgoals.
Coq < Qed.
```

The Coq proof assistant permits to define complex proof tactics, as for example the `lia` tactic, in order to automatize some proof tasks. The `Ltac` and `Ltac2` languages are the supports for the definition of these kind of tactics. These languages permit to compose sequences of tactics, to perform pattern matching over the local environemnt and the current goal in interactive proof mode, to define loops or recursive tactics, etc.

Leveraging the definition of inductive types, the syntactic constructs of programming languages are also easily implemented. Here is the implementation of the syntax of arithmetic expressions presented in the previous section:

```
Inductive e : Set :=
| enat : nat → e
| eid : string → e
| eadd e → e → e.
```

Each constructor corresponds to a construction case in the definition of arithmetic expressions in the Backus-Naur form. The Coq system also generates the induction principle following the structure of arithmetic expressions (thus, a structural induction principle). The induction principle is a proof term of the following logical formula:

```
forall P : e → Prop,
    (forall n : nat, P (enat n)) →
    (forall id : string, P (eid id)) →
    (forall e0 : e, P e0 → forall e1 : e, P e1 → P (eadd e0 e1)) →
    forall e : e, P e
```

The evaluation relation for arithmetic expressions is defined in a similar way:

```
Inductive evale (s : string → option nat) : e → nat → Prop :=
| evalnat : forall n : nat, evale s (enat n) n
| evalid : forall (id : string) (n : nat),
            s id = Some n →
            evale s (eid id) n
| evaladd : forall (e0 e1 : e) (n m : nat),
             evale s e0 n →
             evale s e1 m →
             evale s (eadd e0 e1) (n + m).
```

In the above listing, the state that yields the value of identifiers present in an arithmetic expression is a named parameter of the `evale` relation, i.e. the `s` parameter. Parameters which are not varying from one construction case to another can be passed as named parameters while defining an inductive type. The state `s` takes a string identifier as input and yields an `option` to a natural number. As so, the `option` type permits to represent partial functions. The identifiers that belong to the domain of state `s` will be associated with `Some` natural number, whereas the unreferenced identifiers will be associated with the `None` value of the `option` type. The `Some` and the `None` constructors are the two constructors of the `option` type which is defined in Coq as follows:

```
Inductive option (A : Type) : Type :=
| Some : A → option A
| None : option A.
```

The `option` type is parameterized by a type `A` that will set the type of elements passed to the `Some` constructor. As so, the `option` type is an example of generic type.

1.3.3 Functional programming

As told in the presentation of the CIC, the Coq proof assistant permits to write functional programs, including the definition of recursive functions. The definition of a recursive function is

performed with `Fixpoint` keyword. Here is an example of recursive function defined in Coq. The `pow` function takes two natural numbers `a` and `n` as inputs and yields `a` to the power `n`.

```
Fixpoint pow (a n : nat) {struct n} : nat :=
  match n with
  | O ⇒ 1
  | S m ⇒ a * pow a m
  end.
```

In the body of the `pow` function, the `match` construct performs pattern-matching over the structure of the input `n`. The input `n` is an element of the `nat` type, and thus it could either have been built with the `O` constructor or as the succesor of another element of the `nat` type, i.e. with the `S` constructor. The `match` construct enumerates all the possible construction cases for the given input. Each construction case leads to a pipe-separated entry; for each entry, the structure of the input appears at the left of the arrow, and the result returned appears at the right the arrow. In the above example, `1` is returned if `n` equals `O`, and the result of the multiplication of `a` with the recursive call `pow a m` is returned if `n` is the succesor of a certain `m`. In that case, we have $m = n - 1$, and then the recursive call `pow a m` can be read as `pow a` $(n - 1)$.

When declaring a recursive function, the user must specify which parameter is structurally decrementing through the recursive call. This is performed through {`struct` *id*} annotation, where *id* denotes one parameter of the declared function. This information permits to the Coq kernel to generate the fixpoint equation for the function, thus proving that the function is always terminating.

1.3.4 Dependent types

In the listings that the reader will find in the following chapters, and also in the code repository associated with this thesis, some data structures are *dependently-typed* structures. Thus, we introduce here the notion of dependent type and how it is expressible with the Coq proof assistant.

A type is said to be dependent when its expression depends on one or more elements of other types. To give an example of dependent type, let us take the definition of polymorphic lists that carry their own length. In Coq, these lists are defined as follows:

```
Inductive listn (A : Type) : nat → Set :=
| niln : listn A 0
| consn : forall n : nat, A → listn A n → listn A (S n).
```

The `listn` takes the type `A` of its elements as its first parameter, then its second parameter is an element of the `nat` type which represent the actual length of the list. Note that the first parameter, i.e. the `A` parameter, of the `listn` type alone is not sufficient to qualify `listn` as a dependent type. The `A` parameter is the expression of the polymorphism of the elements of the list involved in generic programming. Polymorphism relates to the fact that the `A` type is general enough to accept multiple types as the type of the list's elements. The `niln` constructor of the `listn`, i.e. the constructor of the empty list, has the type of lists of length 0. The `consn` constructor permits to add a new element at the head of an existing *tail* list to build a new list.

Thus, the type of the resulting list is the type of lists of length $n + 1$, where n is the length of the tail list.

To further illustrate the use of dependent types, let us say that we want to write a function that takes two natural numbers n and m as inputs, and yields $n - m$ only if $n \geq m$. Thus, the function takes two parameters n and m, and a third parameter which is the proof that $n \geq m$. This third parameter *depends* on the two previous parameters, and thus the function is said to be a dependently-typed function. In Coq, it would be written as follows:

```
Definition my_sub (n m : nat) (pf : m <= n) : nat := n − m.
```

Even though, in its definition body, the `my_sub` function simply appeals to the Coq built-in substraction function, passing a proof that m is less than or equal to n adds a constraint to the computation of the substraction. One can see how dependent types can help check that the parameters of programs meet some properties at definition time. Constraining the type of parameters during the definition of programs reduces the proof efforts afterwards, but adds programming complexities at the moment of the definition. Thus, there is a trade-off between using dependent types to constraint the structures and programs at the moment of their definition, or letting the structures and programs as constraint loose as possible at the cost of having to prove much more properties afterwards.

To conclude the subject of dependent types, we often use *sigma* types to define a type of elements that meet a given property. A sigma type expresses the dependence between a parameter and a proof of a given property that possibly depends on another parameter. As so, sigma types are useful to express intentional sets (cf. Section 1.1.2). In the Coq standard library, the definition of the sigma type is as follows:

```
Inductive sig (A:Type) (P: A → Prop) : Type := exist : forall x:A, P x → sig P.
```

The `sig` type only constructor takes an element `x` of type `A` along with a proof that `x` meets a certain property `P`. For instance, if we want to define the type of natural numbers that are strictly greater than zero, we can do it as follows:

```
Definition natstar := sig nat (fun n : nat ⇒ n > 0).
```

The property passed as the second argument of the `sig` type is expressed by a lambda abstraction (denoted by the `fun` keyword) that takes a parameter `n` of type `nat` and returns a proof that `n` is strictly greater than zero. The Coq standard library defines a notation to write sigma types as intensional sets. Thus, we can write the `natstar` type as follows:

```
Definition natstar2 := { x : nat | x > 0}.
```

We can leverage sigma types to rewrite the `my_sub` function presented above. In the following version, the type of the m parameter carries the proof that m is less than or equal to n:

```
Definition my_sub2 (n : nat) (m : { x : nat | x <= n }) : nat := n − (proj1_sig m).
```

Here, we can no longer directly substract n with m as the type of m is no longer `nat` but `{ x : nat | x <= n }`. We have to extract the first part of the m parameter with the help of the `proj1_sig` function. The first part of an element of the `{ x : nat | x <= n }` type corresponds to the natural number `x` verifying the following property `x <= n`, and the second part corresponds to the proof that `x` verifies the property.

Chapter 2

Proving semantic preservation in HILECOP

In this chapter, we present our semantic preservation theorem (or behavior preservation theorem, both denomination are equivalent) along with its informal "paper" proof. The written proof is about a hundred-page long after compilation of the LaTeX files. Therefore, we will only present here the "high-level" theorems and lemmas involved in the demonstration, and some points of our proof strategy. The full proof is available to the reader in Appendix A. The theorems and lemmas presented in this chapter will be referring to the lemmas of Appendix A. The structure of this chapter is as follows: in Section 2.1, we present our review of the literature pertaining to the proof of semantic preservation theorems for transformation functions; in Section 2.2, we detail our state similarity relation, i.e. the semantic bond between an SITPN and its $\mathcal{H}$-VHDL translation; in Section 2.3, we draw out our behavior preservation theorem; in Section 2.4, we detail a particular point of the proof related to the SITPN firing process, and leverage the opportunity to demonstrate our proof strategy; also, we show how this point of the proof has led to a bug detection in the code of the $\mathcal{H}$-VHDL transition design; in Section 2.5, we present some points of the mechanization of the proof with the Coq proof assistant.

2.1 Proofs of semantic preservation in the literature

In this section, we present a review of the literature about the verification of transformation functions. A transformation function is understood here as any kind of mapping from a source representation to a target representation, where the source and target representations possess a behavior of their own (i.e, they are executable). Here, we will focus on verification techniques based on the proof of semantic preservation theorems, with extra interest when the proofs are mechanized within the framework of a proof assistant. We are interested in how to prove that transformation functions are semantic preserving. Especially, we are interested in the expression of semantic preservation theorems, i.e, what does one mean by semantic preservation, and in seeking usual proof strategies.

The goal is to draw our inspiration from the literature and to see how far the correspondence holds between our specific case of transformation, and other cases of transformations. The material used for the literature review is divided into three categories. Each category covers a specific case of transformation function; the three categories are:

- Compilers for generic programming languages
- Compilers for hardware description languages
- Model-to-model and model-to-text transformations

In [15], X.Leroy presents the two points of major importance to express semantic preservation theorems for GPL compilers, and more generally to get the meaning of semantic preservation.

The first point is to clearly state how things are compared between the source and the target programs. It is to describe the runtime state of the source and the target and to draw a correspondence between the two. This is expressed through a state comparison relation.

The second point is to relate the execution of the source program to the execution of the target program through a *simulation* diagram, equivalently named *bisimulation* or *commuting* diagram. Figure 2.1, excerpt from [15], shows the different kinds of simulation diagrams possibly relating two programs together.

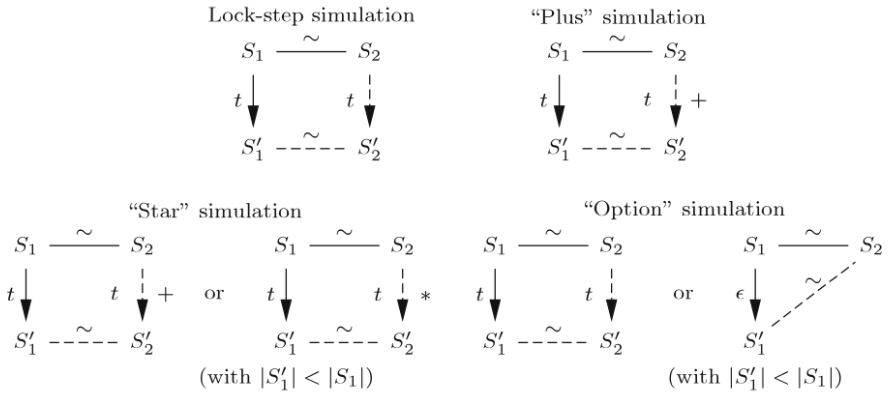

FIGURE 2.1: Simulation diagrams relating the execution of a source program to the execution of a target program; S_1 and S_2 are the initial states of the source and the target program, and S_1' and S_2' are the final states of the source and target program, i.e. the states resulting of the execution of the two programs. The $\sim$ symbol represents the state comparison relation between the source and target language states. The arrows represent the execution relation for the source and target program producing the observable execution trace t.

Choosing an adequate simulation diagram to express a semantic preservation theorem depends on the kind of possible behaviors that can exhibit a given program. In the case of GPL programs, X.Leroy lists three kinds of possible behaviors: either the program execution succeeds and returns a value, or the program execution fails and returns an error, or the program execution diverges. In the case where the source program execution succeeds, a theorem of semantic preservation takes the general form of Definition 1.

Definition 1 (General behavior preservation theorem)**.** *Consider a source programming language L_1 and a target programming language L_2, and a source program $P_1 \in L_1$ compiled into a target*

program $P_2 \in L_2$ by compiler `comp` $\in L_1 \rightarrow L_2$. *Consider an initial state S_1 for program P_1 and an initial state S_2 for program P_2 such that S_1 and S_2 are similar states w.r.t. to a given state comparison relation established between L_1 and L_2. Then, compiler* `comp` *is semantic preserving if it verifies the following property:*

If the execution of P_1 leads from state S_1 to final state S'_1, then there exists a final state S'_2 resulting of the execution of program P_2 from state S_2 such that S'_1 and S'_2 are similar w.r.t. the state comparison relation.

Compiler verification aims at proving the kind of theorem stated above. The other kind of task that can be applied to certify a compiler is to perform compiler validation. Compiler validation is interested in generating a proof of behavior preservation (or a counter-example showing that behaviors diverge) for a given input program alongside the compilation process. Thus, for a given input program, the compiler yields a target program and the proof that the input and target have the same behavior. Exhibiting a theorem of semantic preservation is stronger than building a proof of semantic preservation for each input program. Therefore, compiler verification is stronger than compiler validation. The thesis aims to perform compiler *verification* over the HILECOP methodology.

Now that we have clarified the meaning of semantic preservation for GPL compilers, we state that this definition of semantic preservation holds also for a more general case of transformation from a source representation to a target representation. The only condition to be able to verify that a transformation is semantic preserving is that the source and target representation must have an execution semantics (i.e, the instances of the source and target representations must be executable).

For each article used in the literature review and presenting a specific case of transformation, the following questions have been asked:

- What are the similarities/differences between source and target representations? May they be programs of GPLs, or models of a given model formalism.
- How is defined the runtime state for the source and target representations?
- How is expressed the state comparison relation?
- How is expressed the semantic preservation theorem?
- What is the employed proof strategy?

2.1.1 Compilers for generic programming languages

Taking the CompCert compiler as an example, the compilation pass from Clight programs to Cminor programs is described in [3, 15]. Clight is a subset of the C language, and Cminor is a low-level imperative language. The two languages are endowed with a big-step operational semantics. Here, the execution state of the source and target languages are memory models (of course, we are dealing with programming languages). The memory model consists of block references; each block has a lower and an upper bound. To access data, one has to specify the block reference along with the size of the accessed data (i.e, the data type) and the offset

from the start of the block reference (i.e, where to begin the data reading). About the proof of semantic preservation, the most difficult point is to relate the memory state of the source program to the memory state of the target program. To do so, the authors define a *memory injection* relation that binds the values of source and target together. They also establish a relation to compare execution environments, i.e, the environments holding the declaration of functions, global variables... The proof of semantic preservation is built incrementally. First, the authors prove a correctness lemma for the Clight expressions: if a Clight expression a evaluates to value v, then the translated Cminor expression $\lfloor a \rfloor$ evaluates to value v. Then, they prove a similar lemma for Clight statements, and finally for the entire Clight program. The proof strategy is to reason by induction over the evaluation relation of the Clight programs and to perform case analysis on the translation function.

The pattern to compiler verification for GPLs is more or less the same as presented above. May it be compilers for imperative languages [15, 20], or compilers for functional languages [8, 21], compiler verification proceeds as follows:

1. Establish a relationship between the memory models of the source and target languages, and between the global execution environments.

2. Prove correctness lemmas starting from simple constructs, and building up incrementally to consider entire programs.

3. Reason by induction over the evaluation relation of the source language, and the translation function.

Relating memory models is more difficult when the gap between the source and target languages is important (for instance, the translation of Cminor programs into RTL programs in [15]). As a consequence, the complexity of the memory model comparison relation increases.

2.1.2 Compilers for hardware description languages

In the case of HDL compilers, proving semantic preservation is very similar to the case of GPL compilers. Of course, the difference lies in the semantics of HDL languages and the description of execution states. The semantics of HDLs is intrinsically related to the notion of execution over time, or over multiple clock cycles; indeed, we are dealing with reactive systems. Therefore, the semantic preservation theorems are formulated w.r.t. the synchronous or the time-related semantics of the considered languages.

In [4, 6], the source language is a subset of the BlueSpec specification language for hardware synthesis, and the target language is an RTL representation of the circuit. The runtime state of the source and target programs are basically a mapping between registers to values. In [4], the execution state also holds a log of the read and write operations of the input program, and this log is compared to the log of the RTL representation. The semantic preservation theorem takes the general form of Definition 1, however, the final states refer to the states of source and target programs at the end of a clock cycle. Thus, the semantic preservation theorem states that starting from equal register stores after the execution of a source program and its RTL circuit after one clock cycle leads to equal register stores.

In [5], the source language is a subset of Lustre and the target language is an imperative language called Obc. A Lustre program is composed of nodes; each node treats a set of input streams and publishes output streams after the computation of its statement body. In its statement body, a Lustre node possibly refers to instances of other nodes. In the compilation process, each Lustre node is translated into an Obc class. An Obc class holds a vector of variables composing its internal memory and a vector of other Obc class instances. The authors define a data flow semantics for the Lustre language; judgments of the semantics describe how output streams are computed based on input streams. Furthermore, as we are dealing with hardware circuits, the semantics rules cover synchronous statements and combinational ones. On the side of the Obc language, the semantics define a function *step* that computes the execution of the Obc classes over one clock cycle. To prove the semantic preservation theorem, the state comparison relation binds the values of input and output streams on one side to the values of variables and Obc class instances on the other side. The semantic preservation theorem is as follows: if a Lustre node yields the output stream o from an input stream i, then the iterative execution of the *step* function for the corresponding Obc class incrementally builds the output stream o given the values of the input stream i. The proof is done by induction over the clock step count, and by induction over the evaluation relation for the Lustre statements composing the body of nodes.

In [16], the HDL compiler translates Verilog modules into netlists. The execution state of Verilog module holds the value of the variables declared in the module. The execution state of a netlist circuit holds the value of the registers declared in the circuit. Therefore, the state comparison relation, used to state the semantic preservation theorem, binds the values of variables on one side to the values of registers on the other side. The semantics of Verilog quite similar to the one of VHDL; a set of processes composing a module are executed w.r.t. the simulation semantics of the language, i.e, composed of synchronous and combinational execution steps. The semantics of netlists is set as a big-step operational semantics through an interpreter that runs a netlist list over n clock cycles. The semantic preservation theorem is as follows: Assuming that a module is transformed into a circuit, and that some well-formation hypotheses hold on the module, if the module executes without error, and yields a final state *venv*, then there exists a final state *cenv* yielded by the execution of the circuit over n clock cycles s.t. *venv* and *cenv* are similar according to the relation *verilog_netlist_rel*. Here, the *verilog_netlist_rel* is the state comparison relation.

In [26], the compiler transforms programs of the synchronous language SIGNAL into Synchronous Clock Guarded Actions programs (S-CGA programs). A SIGNAL program describes a set of processes; each process holds a set of equations describing the relation between signals. The equations can be synchronous equations (referring to a clock) or combinational ones. An S-CGA program defines a set of actions to be applied to some variables when some conditions (the guards) are met. The SIGNAL (resp. the S-CGA) language has been endowed with a trace semantics describing the computation of signal values (resp. variable values) over time. The authors describe a function to translate the traces of SIGNAL and S-CGA programs into a common trace model. Thus, the semantic preservation theorem is stated by comparing two traces of execution defined through the same model. The proof of the semantic preservation theorem is built incrementally. For each statement of a SIGNAL process, the authors exhibit a lemma proving that the trace resulting from the execution of the statement is equivalent to the

trace resulting of the execution of the corresponding guarded actions (obtained through the compilation). The proof is fully mechanized within the Coq proof assistant.

In [13], the authors verify a methodology to design hardware models with SystemC models. SystemC models describe hardware systems with modules; a module is a C++ class with ports, data members and methods. The methodology describes a transformation from SystemC models to Abstract State Machine (ASM) thus enabling to model-check the hardware models. ASMs are described in the language AsmL; in AsmL, an ASM is implemented by a class with data members and methods. A denotational (fixpoint) semantics for SystemC models is defined along with a denotational semantics for AsmL. The semantics is another variant of simulation cycle, similar to all other synchronous languages. There are two phases: evaluate and update and the gap between the two is called a delta-delay. The execution state of a SystemC model is divided into a signal store, mapping signal to value, and a variable store, mapping variable to value. The execution state of an AsmL class is only composed of a variable store. The theorem of semantic preservation states that, after translation, a SystemC model has the same *observational* behavior than its corresponding AsmL class. What is compared between a SystemC model and its corresponding AsmL class through their observational behavior is the activity of the processes of the first one and the activity of the methods of the second one. Processes and methods must be active at the same delta cycles. Therefore, what is compared here are not the values that the execution states hold, but rather the activity of the source and target programs.

2.1.3 Model transformations

Regarding model transformations, a lot of works consider semantic preservation as the preservation of structural properties in the transformed model [1, 7, 17].

Still, there are many cases where the source model and the target one have both an execution semantics. In these cases, the authors are interested in proving that the transformation is semantic preserving by showing that the computation of the source model and the target model follow a commuting diagram (see Figure 2.1).

In [10] and [25], the authors are interested in giving a translational semantics to a given model having itself a reference execution semantics. In [10], the source models are called xSpem models; they describe a set of *activities* that exchange resources and hold an internal state. The target models are PNs. Both xSpem models and PNs have a state transition semantics. The state comparison is performed by checking the correspondence between each current status of the activities describe in an xSpem model and the marking of the PN. Then, the authors prove a bisimulation theorem, illustrated in Figure 2.2.

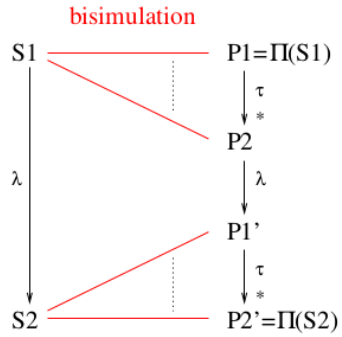

FIGURE 2.2: Bisimulation diagram relating an xSpem model execution and a Petri net execution

In Figure 2.2, one the right side of the diagram, i.e, the Petri net side, one can see that a Petri net possibly performs many internal actions (represented the arrow $\xrightarrow{\tau}^{*}$) before and after executing the computation step that is of interest for the proof (i.e, action λ). The proof is performed by reasoning by induction on the structure of the xSpem models, and then by reasoning of the state transition semantics of xSpem models and PNs.

In [25], the authors describe a transformation from a model of the AADL formalism (Architecture Analysis and Design Language) to a particular kind of Abstract State Machine (ASM) called Timed Abstract State Machines (TASM). To verify that the transformation is semantic preserving, the authors define the semantics of AADL models and TASMs through Timed Transition Systems (TTSs). Thus, the execution state of an AADL model is the execution state of the corresponding TTS, and the same holds for a TASM. Comparing the state of two TTSs is easier than comparing the state of two different models, thus having two different definitions. Then, the authors prove a strong bisimulation theorem to verify that the transformation is semantic preserving. The whole proof is mechanized within the Coq proof assistant.

In [12], the authors describe a transformation from LLVM-labelled Petri nets to LLVM programs, where LLVM is low-level assembly language. Precisely, the generated LLVM program implements the state space of the source Petri net (i.e, the graph of reachable markings). The authors want to verify if an LLVM program truly implements the PN state space, i.e if each marking present in the PN state space can be reached by running a specific $fire_t$ function on the generated LLVM program. The state of an LLVM program is defined by a memory model composed of a heap and a stack. The marking of an LLVM-labelled PN is defined in such a manner that the correspondence with the LLVM program memory model is straightforward. The PN model has classical firing semantics, and LLVM programs follow a small-step operational semantics. The semantic preservation theorem states that for all transition t being fired, leading from marking M to marking M', then applying running the $fire_t$ function over the generated LLVM program at state LM (such that LM implements marking M) leads to a new state LM', such that LM' implements marking M'. To prove this theorem, the authors proceed by induction on the number of places of the input Petri net.

2.1.4 Discussions on transformations and proof strategies

In this thesis, we are interested in the verification of a semantic preservation property for a given transformation function. To achieve this kind of proof task, the proceedings are quite similar, at least in the three cases of transformation presented above (i.e, GPL compilation, HDL compilation, and model transformations). Even though the source and target languages or models are different from one case of transformation to the other, however, semantic preservation theorems carry the same structure, i.e the one presented in Definition 1. The state comparison relation and the choice of the commuting diagram (i.e. how much computational steps of the target representation correspond to one computational step of the source representation) are the two angular stones of the process.

One can notice that when verifying the transformation of HDL programs, the semantic preservation theorems are expressed around a time-related computational step. It can either be a clock cycle or another kind of time step. The state equivalence checking is made at the end of this time-related computational step. This differs from the expression of behavior preservation theorems for GPLs, where a computational step is not related to time, but rather expresses the one-time computation of programs.

Concerning proof strategies, in the case of programming languages, proving the semantic preservation theorems are systematically done by induction over the semantics relations of the source and target languages, and by reasoning on the translation function. The semantics relations are themselves defined by following the inductive structure of the language ASTs. In the case of model transformations, when the source model permits it, the proofs are performed similarly by applying inductive reasoning over the structure of the input model. This enables compositional reasoning, i.e: to split the difficulty of proving the semantic preservation theorem into simpler lemmas about the execution of simpler programs or simple model structures.

2.2 The state similarity relation

Before presenting our behavior preservation theorem, we must clarify the meaning of semantic preservation between an SITPN and a $\mathcal{H}$-VHDL design. To do so, we must define:

1. What does semantic similarity mean between an SITPN state and a $\mathcal{H}$-VHDL state?
2. When, in the course of the execution of an SITPN and a $\mathcal{H}$-VHDL design, does this semantic similarity must hold?

We must relate the elements that constitute the execution state of an SITPN to the elements that constitute the execution state of a $\mathcal{H}$-VHDL design. An SITPN state is an abstract structure relating the places, transitions, actions, functions and conditions of a given SITPN to the values of certain domains (see Section **??**). A $\mathcal{H}$-VHDL design state is composed a signal store mapping signals to values, and of a component store mapping component instances to their own internal states (which are themselves design states). Thanks to the binder function γ generated alongside the transformation from an SITPN to a $\mathcal{H}$-VHDL design, we are able to relate the elements of the SITPN state structure to the component instance states and signal values of the $\mathcal{H}$-VHDL

design state. Thus, the state similarity relation, depending on a γ binder and expressing a semantic match between an SITPN state and a $\mathcal{H}$-VHDL design, is defined as follows:

Definition 2 (General state similarity)**.** *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, an elaborated design $\Delta \in ElDesign$, and a binder $\gamma \in WM(sitpn, d)$, an SITPN state $s \in S(sitpn)$ and a design state $\sigma \in \Sigma$ are similar, written $\gamma \vdash s \sim \sigma$ iff*

1. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s.M(p) = \sigma(id_p)(\texttt{s\_marking})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\big(upper(I_s(t)) = \infty \wedge s.I(t) \leq lower(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\texttt{s\_time\_counter})\big)$
   $\wedge\big(upper(I_s(t)) = \infty \wedge s.I(t) > lower(I_s(t)) \Rightarrow \sigma(id_t)(\texttt{s\_time\_counter}) = lower(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) > upper(I_s(t)) \Rightarrow \sigma(id_t)(\texttt{s\_time\_counter}) = upper(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) \leq upper(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\texttt{s\_time\_counter})\big)$.
3. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, s.reset_t(t) = \sigma(id_t)(\texttt{s\_reinit\_time\_counter})$.
4. $\forall c \in \mathcal{C}, id_c \in Ins(\Delta)\ s.t.\ \gamma(c) = id_c,\ s.cond(c) = \sigma(id_c)$.
5. $\forall a \in \mathcal{A}, id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a,\ s.ex(a) = \sigma(id_a)$.
6. $\forall f \in \mathcal{F}, id_f \in Outs(\Delta)\ s.t.\ \gamma(f) = id_f,\ s.ex(f) = \sigma(id_f)$.

In Item 1, based on the γ binder, we relate the marking value of a place p at state s to the value of the `s_marking` signal inside the internal state of the place component instance (PCI) id_p. The expression $\sigma(id_p)$ returns the internal state of PCI id_p by looking up the component store of state σ. Items 2 and 3 similarly relate the value of time counters (resp. reset orders) of transitions to the value of the signals `s_time_counter` (resp. `s_reinit_time_counter`) in the internal state of the corresponding transition component instances (TCIs). In item 4 (resp. 5 and 6), the boolean value of conditions (resp. actions and functions) are compared to the value of input (resp. output) ports of the $\mathcal{H}$-VHDL design, also based on the γ binder.

As one can observe in Item 2, the relation between the value of a time counter and the value of the `s_time_counter` signal is a particular. It is due to the definition domain of time intervals. In the definition of the SITPN structure, a time interval i is defined as follows: $i = [a, b]$ where $a \in \mathbb{N}^*$ and $b \in \mathbb{N}^* \sqcup \{\infty\}$. In the SITPN semantics, depending on certain conditions, a time counter possibly increments its value until it reaches the upper bound of the associated time interval. Therefore, a time counter associated to a time interval with an infinite upper bound will possibly increment its value indefinitely. While acceptable in the theoretical world, this is not acceptable is the world of hardware circuits where all dimensions and values are finite. On the $\mathcal{H}$-VHDL side, the signal `s_time_counter`, which value represents the value of a time counter, will stop its incrementation to the lower bound of the time interval in the case where the upper bound is infinite. As long as the value of the time counter is less than or equal to the lower bound of the time interval, we look for a perfect equality between the value of the time counter and the value of the `s_time_counter` signal. When the time counter reaches the lower bound, the values possibly diverge (i.e, the time counter value continues to be incremented while the value of the `s_time_counter` signal stalls). In that case, we are only interested in

knowing that the value of the `s_time_counter` signal is equal to the value of the lower bound of the time interval. The two last points of Item 2 are necessary to cover the case where a time counter has overreached the upper bound of its time interval. In that case, the time counter becomes *locked*. The `s_time_counter` signal can not overreached the upper bound of the time interval without causing an overflow. Thus, the value of the `s_time_counter` signal diverges from the value of its corresponding time counter when the time counter overreaches the upper bound of its time interval. While the time counter is less than or equal to the upper bound of its time interval, we look for a perfect equality between the value of the time counter and the value of the `s_time_counter` signal. When the time counter overreaches the upper bound, the value of the time counter stalls to upper bound plus one, and the value of `s_time_counter` stalls to upper bound. In that case, we are only interested in knowing that the value of the `s_time_counter` signal is equal to the value of the upper bound of the time interval.

The second question that we asked above was: when does the state similarity relation must hold in the course of the execution? The source and target representations are both synchronously executed. Thus, we find it natural to check that the state similarity relation holds at the end of a clock cycle. However, due to modifications resulting after a bug detection (see Section 2.4), the state similarity relation of Definition 2.2 does not hold at the end of a clock cycle. The equality between the value of reset orders and the value of the `s_reinit_time_counter` signals (Item 3) is not verified. However, this semantic divergence is without effect. New reset orders are computed at the beginning of a clock cycle such that the relation of Item 3 holds in the middle of the clock cycle (i.e, just before the falling edge of the clock). This is the only moment during the clock cycle where the `s_reinit_time_counter` signal is actually involved in the computation of other signals value. Thus, it is sufficient that Item 3 holds only in the middle of the clock cycle. However, we must now define two state similarity relation; one that checks the semantic similarity after the rising edge of the clock signal (i.e, in the middle of the clock cycle), and one that checks the semantic similarity after the falling edge of the clock signal (i.e, at the end of the clock cycle). The state similarity relation after a rising edge is defined as follows:

Definition 3 (Post rising edge state similarity)**.** *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, an elaborated design $\Delta \in ElDesign$, and a binder $\gamma \in WM(sitpn, d)$, an SITPN state $s \in S(sitpn)$ and a design state $\sigma \in \Sigma$ are similar after a rising edge, written $\gamma \vdash s \overset{\uparrow}{\sim} \sigma$ iff*

1. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s.M(p) = \sigma(id_p)(\texttt{s\_marking})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\big(upper(I_s(t)) = \infty \wedge s.I(t) \leq lower(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\texttt{s\_time\_counter})\big)$
   $\wedge\big(upper(I_s(t)) = \infty \wedge s.I(t) > lower(I_s(t)) \Rightarrow \sigma(id_t)(\texttt{s\_time\_counter}) = lower(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) > upper(I_s(t)) \Rightarrow \sigma(id_t)(\texttt{s\_time\_counter}) = upper(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) \leq upper(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\texttt{s\_time\_counter})\big)$.
3. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, s.reset_t(t) = \sigma(id_t)(\texttt{s\_reinit\_time\_counter})$.
4. $\forall a \in \mathcal{A}, id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a,\ s.ex(a) = \sigma(id_a)$.

5. $\forall f \in \mathcal{F}, id_f \in Outs(\Delta)\ s.t.\ \gamma(f) = id_f,\ s.ex(f) = \sigma(id_f)$.

Definition 3 is similar to Definition 2 in all points, except for the value of conditions. A condition of an SITPN is implemented by an input port in the resulting $\mathcal{H}$-VHDL top-level design. In the $\mathcal{H}$-VHDL semantics, the value of primary input ports (i.e, the input ports of the top-level design) are updated at each clock edge. In the SITPN semantics, the value of conditions are updated only at the falling edge of the clock. Consider that a given SITPN is executed at clock cycle τ; after the rising edge of the clock, the value of conditions are equal to their value at clock cycle $\tau - 1$, whereas the value primary input ports have been updated to fresh values. Thus, we will have to wait for the next falling edge to reach the equality between condition values and input port values. Therefore, there is a semantic divergence between the value of conditions and the value of input ports in the middle of the clock cycle, i.e. just before the next falling edge of the clock signal. However, similarly to the case of reset orders and `s_reinit_time_counter` signals, conditions and their corresponding input ports are only involved in computations at the falling edge of the clock cycle. Thus, it is sufficient that Item 4 holds only right after the falling of the clock signal.

The state similarity relation draws out a correspondence between the values hold by an SITPN state and the values of the signals declared in a $\mathcal{H}$-VHDL design state. However, to complete the proof of semantic preservation, we sometimes have to relate the value of signals to the value of expressions or predicates involved in the SITPN semantics. For instance, consider a given SITPN state s and a given $\mathcal{H}$-VHDL design state σ, and consider a transition t and its corresponding TCI id_t. It is useful to show that, after a rising edge, the value of signal `s_enabled` at state $\sigma(id_t)$, where $\sigma(id_t)$ denotes the internal state of component instance id_t at state σ, is equal to the predicate $t \in Sens(s.M)$ stating that the transition t is sensitized (or *enabled*) by the marking at state s (i.e, $s.M$). Thus, for the convenience of the proof, we enrich our definitions of the state similarity relations with formulas relating $\mathcal{H}$-VHDL signals to SITPN semantics predicates and expressions. Consequently, the *full* post rising edge state similarity relation is defined as follows:

Definition 4 (Full post rising edge state similarity)**.** *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, an elaborated design $\Delta \in ElDesign$, and a binder $\gamma \in WM(sitpn, d)$, a clock cycle count $\tau \in \mathbb{N}$, and an SITPN execution environment $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, an SITPN state $s \in S(sitpn)$ and a design state $\sigma \in \Sigma$ are fully similar after a rising edge happening at clock cycle count τ, written $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, if $\gamma \vdash s \overset{\uparrow}{\sim} \sigma$ (Definition 3) and*

1. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Sens(s.M) \Leftrightarrow \sigma(id_t)(\texttt{s\_enabled}) = \texttt{true}$.
2. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Sens(s.M) \Leftrightarrow \sigma(id_t)(\texttt{s\_enabled}) = \texttt{false}$.
3. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
$$\sigma(id_t)(\texttt{s\_condition\_combination}) = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau, c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}$$
where $conds(t) = \{c \in \mathcal{C} \mid \mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1\}$.

Definition 4 extends Definition 3 with the correspondence of the sensitization of transitions and the value of signal `s_enabled`, and the computation of the boolean product of condition values and the value of signal `s_condition_combination`.

Now, let us define the state similarity relation describing how things must be compared between an SITPN state and a $\mathcal{H}$-VHDL design state after the falling edge of a clock signal:

Definition 5 (Post falling edge state similarity)**.** *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, an elaborated design $\Delta \in ElDesign$, and a binder $\gamma \in WM(sitpn, d)$, an SITPN state $s \in S(sitpn)$ and a design state $\sigma \in \Sigma$ are similar after a falling edge, written $\gamma \vdash s \overset{\downarrow}{\sim} \sigma$, if*

1. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s.M(p) = \sigma(id_p)(\texttt{s\_marking})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\big(upper(I_s(t)) = \infty \wedge s.I(t) \leq lower(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\texttt{s\_time\_counter})\big)$
   $\wedge\big(upper(I_s(t)) = \infty \wedge s.I(t) > lower(I_s(t)) \Rightarrow \sigma(id_t)(\texttt{s\_time\_counter}) = lower(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) > upper(I_s(t)) \Rightarrow \sigma(id_t)(\texttt{s\_time\_counter}) = upper(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) \leq upper(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\texttt{s\_time\_counter})\big)$.
3. $\forall c \in \mathcal{C}, id_c \in Ins(\Delta)\ s.t.\ \gamma(c) = id_c,\ s.cond(c) = \sigma(id_c)$.
4. $\forall a \in \mathcal{A}, id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a,\ s.ex(a) = \sigma(id_a)$.
5. $\forall f \in \mathcal{F}, id_f \in Outs(\Delta)\ s.t.\ \gamma(f) = id_f,\ s.ex(f) = \sigma(id_f)$.

As explained above, Definition 5 is similar to Definition 2 except for the equality between reset orders and the value of the `s_reinit_time_counter` signals. The extended version of the post falling edge state similarity relation is defined as follows:

Definition 6 (Full post falling edge state similarity)**.** *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, an elaborated design $\Delta \in ElDesign$, and a binder $\gamma \in WM(sitpn, d)$, an SITPN state $s \in S(sitpn)$ and a design state $\sigma \in \Sigma$ are fully similar after a falling edge, written $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, if $\gamma \vdash s \overset{\downarrow}{\sim} \sigma$ (Definition 5) and*

1. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Firable(s) \Leftrightarrow \sigma(id_t)(\texttt{s\_firable}) = \texttt{true}$.
2. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Firable(s) \Leftrightarrow \sigma(id_t)(\texttt{s\_firable}) = \texttt{false}$.
3. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Fired(s) \Leftrightarrow \sigma(id_t)(\texttt{fired}) = \texttt{true}$.
4. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Fired(s) \Leftrightarrow \sigma(id_t)(\texttt{fired}) = \texttt{false}$.
5. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p, \sum_{t \in Fired(s)} pre(p,t) = \sigma(id_p)(\texttt{s\_output\_token\_sum})$.
6. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p, \sum_{t \in Fired(s)} post(t,p) = \sigma(id_p)(\texttt{s\_input\_token\_sum})$.

Definition 6 extends Definition 5 by drawing out a correspondence between:

- the firability of transitions and the value of the signal `s_firable`
- the firing status of transitions (i.e, transitions are fired or not) and the value of the output port `fired`
- the sum of tokens consumed by the firing process and the value of the signal `s_output_token_sum`
- the sum of tokens produced by the firing process and the value of the signal `s_input_token_sum`

2.3 Behavior preservation theorem

In this section, we lay out the major theorems and lemmas stating that the HILECOP transformation function is semantic preserving. We also present the informal proofs for these theorems and lemmas.

2.3.1 Proof notations

To add some readability to our proofs, we use the following notations:

- The most recent framed box above the point of reading denotes the current pending goal (what we are currently trying to prove): $\boxed{\forall n \in \mathbb{N},\ n > 0 \vee n = 0}$
- A red framed box denotes a completed goal (i.e. equivalent to QED): `true = true`
- A green framed box denotes the current induction hypothesis:

  $\forall n \in \mathbb{N},\ n + 1 > 0$

- The mention **CASE** directly follows an item bullet to denote a case during a proof by case analysis.

During a proof, we constantly refer to the names of the constants and signals declared in the $\mathcal{H}$-VHDL `place` and `transition` designs. Some constants and signals have very long names, and therefore we use aliases to refer to them in the following proofs. Table A.1 gives the full correspondence between constant and signal names and their aliases. Also, during a proof and when there is no ambiguity, id_p (resp. id_t) denotes the PCI (resp. TCI) identifier associated to a given place p (resp. transition t) through $\gamma(p) = id_p$ (resp. $\gamma(t) = id_t$), where γ is the binder returned by the transformation function. Similarly, id_c (resp. id_a and id_f) denotes the input port (resp. output port) identifier associated to a given condition c (resp. action a and function f) through $\gamma(c) = id_c$.

2.3.2 Preliminary definitions

We define here some relations that are necessary to formalize our theorem of behavior preservation.

In an SITPN, the conditions associated to transitions receive fresh Boolean values from an execution environment at each falling edge of the clock. During the simulation of a top-level design, the input ports of the design receive fresh values from a simulation environment at each clock event. The transformation function generates an input port in the top-level design that will reproduce the behavior of a given SITPN condition. The binder γ, generated alongside the top-level design, relates a given condition c to its corresponding input port identifier id_c. To compare the execution/simulation traces of an SITPN and a $\mathcal{H}$-VHDL design, we must assume that the execution/simulation environments assign similar values to conditions and to their corresponding input ports at a given clock cycle. Definition 7 states that the execution environment for a given SITPN and the simulation environment for a given $\mathcal{H}$-VHDL design are similar.

Definition 7 (Similar environments)**.** *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, a design store $\mathcal{D} \in entity\text{-}id \rightharpoonup design$, an elaborated version $\Delta \in ElDesign$ of design d, and a binder $\gamma \in WM(sitpn, d)$, the environment $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \rightarrow Ins(\Delta) \rightarrow value$, that yields the value of the primary input ports of Δ at a given simulation cycle and a given clock event, and the environment E_c, that yields the value of conditions of $sitpn$ at a given execution cycle, are similar, written $\gamma \vdash E_p \overset{env}{=} E_c$, if for all $\tau \in \mathbb{N}$, $clk \in \{\uparrow, \downarrow\}$, $c \in \mathcal{C}$, $id_c \in Ins(\Delta)$ s.t. $\gamma(c) = id_c$, $E_p(\tau, clk)(id_c) = E_c(\tau)(c)$.*

Definition 7 also states that every input port of the top-level design related to a SITPN condition by the γ binder has a stable boolean value during a whole clock cycle. That is to say, in the context of Definition 7, there exists no id_c such that $E_p(\tau, \uparrow)(id_c) \neq E_p(\tau, \downarrow)(id_c)$.

To prove that the behavior of an SITPN and a $\mathcal{H}$-VHDL design are similar, we want to compare the states composing their execution/simulation traces. As a reminder, an execution/simulation trace is a time-ordered list of states describing the evolution of a given SITPN or $\mathcal{H}$-VHDL design through a certain number of clock cycles. The relation presented in Definition 8 permits to compare such traces.

Definition 8 (Execution trace similarity)**.** *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, an elaborated design $\Delta \in ElDesign$, and a binder $\gamma \in WM(sitpn, d)$, the execution trace $\theta_s \in \texttt{list}(S(sitpn))$ and the simulation trace $\theta_\sigma \in \texttt{list}(\Sigma)$ are similar, written $\gamma \vdash \theta_s \overset{clk}{\sim} \theta_\sigma$, where $clk \in \{\uparrow, \downarrow\}$, according to the following rules:*

$$\frac{}{\gamma \vdash [\,] \overset{clk}{\sim} [\,]}\ \textsc{SimTraceNil}\ clk \in \{\uparrow, \downarrow\} \qquad \frac{\gamma \vdash s \overset{\uparrow}{\sim} \sigma \quad \gamma \vdash \theta_s \overset{\downarrow}{\sim} \theta_\sigma}{\gamma \vdash (s :: \theta_s) \overset{\uparrow}{\sim} (\sigma :: \theta_\sigma)}\ \textsc{SimTrace}\uparrow \qquad \frac{\gamma \vdash s \overset{\downarrow}{\sim} \sigma \quad \gamma \vdash \theta_s \overset{\uparrow}{\sim} \theta_\sigma}{\gamma \vdash (s :: \theta_s) \overset{\downarrow}{\sim} (\sigma :: \theta_\sigma)}\ \textsc{SimTrace}\downarrow$$

In Definition 8, the clock event symbol on top of the $\sim$ sign indicates the kind of clock event that led to the production of the states at the head of the traces. The execution trace similarity relation expects that the states composing the traces have been alternatively produced by a

rising edge step and then by a falling edge step. By construction, the traces must have the same length to respect the execution trace similarity relation.

To handle the case of an execution/simulation trace beginning by an initial state, that is, a state neither reached after a rising nor after a falling edge, we give a slightly different definition of the execution trace similarity relation in Definition 9.

Definition 9 (Full execution trace similarity)**.** *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, an elaborated design $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$, and a binder $\gamma \in WM(sitpn, d)$, the execution trace $\theta_s \in \texttt{list}(S(sitpn))$ and the simulation trace $\theta_\sigma \in \texttt{list}(\Sigma)$ are fully similar, written $\gamma \vdash \theta_s \sim \theta_\sigma$, according to the following rules:*

$$\frac{}{\gamma \vdash [\,] \sim [\,]}\ \textsc{FullSimTraceNil} \qquad \frac{\gamma \vdash s \sim \sigma \qquad \gamma \vdash \theta_s \overset{\uparrow}{\sim} \theta_\sigma}{\gamma \vdash (s :: \theta_s) \sim (\sigma :: \theta_\sigma)}\ \textsc{FullSimTraceCons}$$

The full execution trace similarity relation indicates that the head states of traces must verify the general state similarity relation, and that the tail of the traces must respect the execution state similarity relation starting with a rising edge step.

2.3.3 The behavior preservation theorem

Theorem 1 expresses our behavior preservation theorem. Theorem 1 states that the HILECOP transformation function is semantic preserving when the input model is a well-defined SITPN (see Definition **??**). As a complementary task, we could show that if the transformation function returns a couple $\mathcal{H}$-VHDL design and binder, and not an error, then the input SITPN is well-defined. To prove Theorem 1, we must first exhibit an elaborated version of the returned $\mathcal{H}$-VHDL design (Theorem 2), an initial state (Theorem 3), and a simulation trace over τ simulation cycles (Theorem 4). Finally, we can establish that the behaviors are similar by comparing the respective SITPN execution and $\mathcal{H}$-VHDL design simulation traces (Theorem 5). In this thesis, we are focusing on the proof that the execution/simulation traces are similar when they are produced by the SITPN execution relation and the $\mathcal{H}$-VHDL simulation relation over τ clock cycles. This corresponds to the proof of Theorem 5. For now, we choose to consider Theorems 2, 3 and 4 as axioms.

Theorem 1 (Behavior preservation)**.** *For all well-defined $sitpn \in SITPN$, an $\mathcal{H}$-VHDL design $d \in design$, a binder $\gamma \in WM(sitpn, d)$, a clock cycle count $\tau \in \mathbb{N}$, an execution environment $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$ and an execution trace $\theta_s \in \texttt{list}(S(sitpn))$ s.t.*

- *SITPN $sitpn$ translates into $\mathcal{H}$-VHDL design d and yields a binder γ: $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$*
- *SITPN $sitpn$ yields the execution trace θ_s after τ execution cycles in environment E_c:*
  $E_c, \tau \vdash sitpn \xrightarrow{full} \theta_s$

then there exist an elaborated design $\Delta \in ElDesign$ and a simulation trace $\theta_\sigma \in \texttt{list}(\Sigma)$ s.t.
for all simulation environment $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value$ verifying $\gamma \vdash E_p \overset{env}{=} E_c$ (simulation and execution environments are similar), we have:

- *In the context of the HILECOP design store $\mathcal{D}_\mathcal{H}$ and with an empty generic constant dimensioning function ($\emptyset$), design d elaborates into Δ and yields the simulation trace θ_σ after τ simulation cycles:*
  $\mathcal{D}_\mathcal{H}, \Delta, \emptyset, E_p, \tau \vdash \mathrm{d} \xrightarrow{full} \theta_\sigma$
- *Traces θ_s and θ_σ are fully similar: $\theta_s \sim \theta_\sigma$*

Proof. Given a $sitpn \in SITPN$, a $d \in design$, a $\gamma \in WM(sitpn, d)$, a $\tau \in \mathbb{N}$, an $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$ and a $\theta_s \in \mathtt{list}(S(sitpn))$, let us show that

$$\boxed{\exists \Delta, \theta_\sigma, \forall E_p,\ \gamma \vdash E_p \overset{env}{=} E_c \Rightarrow (\mathcal{D}_\mathcal{H}, \Delta, \emptyset, E_p, \tau \vdash \mathrm{d} \xrightarrow{full} \theta_\sigma) \wedge \theta_s \sim \theta_\sigma}$$

Appealing to Theorems 2, 3 and 4, let us take an elaborated design $\Delta \in ElDesign$, two design states $\sigma_e, \sigma_0 \in \Sigma$, and a simulation trace $\theta_\sigma \in \mathtt{list}(\Sigma)$ such that:

– Δ is the elaborated version of design d, and σ_e is the default design state of Δ:
  $\mathcal{D}_\mathcal{H}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e)$

– σ_0 is the initial simulation state: $\mathcal{D}_\mathcal{H}, \Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0$

– Design d yields the simulation trace θ_σ after τ simulation cycles, starting from initial state σ_0:
  $\mathcal{D}_\mathcal{H}, E_p, \Delta, \tau, \sigma_0 \vdash \mathrm{d.cs} \to \theta_\sigma$

Let us use this Δ and this θ_σ to prove the current goal. Given an E_p such that $\gamma \vdash E_p \overset{env}{=} E_c$, it remains to be proved that:

$$\boxed{(\mathcal{D}_\mathcal{H}, \Delta, \emptyset, E_p, \tau \vdash \mathrm{d} \xrightarrow{full} \theta_\sigma) \wedge \theta_s \sim \theta_\sigma}$$

First, we must prove that $\boxed{(\mathcal{D}_\mathcal{H}, \Delta, \emptyset, E_p, \tau \vdash \mathrm{d} \xrightarrow{full} \theta_\sigma)}$ holds. By definition of the $\mathcal{H}$-VHDL full simulation relation, we have:

$$\begin{aligned} \mathcal{D}_\mathcal{H}, \Delta, \emptyset, E_p, \tau \vdash \mathrm{d} \xrightarrow{full} \theta_\sigma \equiv \exists \sigma_e, \sigma_0 \in \Sigma(\Delta),\ & \mathcal{D}_\mathcal{H}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e) \\ \wedge & \mathcal{D}_\mathcal{H}, \Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0 \\ \wedge & \mathcal{D}_\mathcal{H}, E_p, \Delta, \tau, \sigma_0 \vdash \mathrm{d.cs} \to \theta_\sigma \end{aligned} \tag{2.1}$$

Rewriting the goal with (2.1):

$$\boxed{\exists \sigma_e, \sigma_0 \text{ s.t. } \mathcal{D}_\mathcal{H}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e) \wedge \mathcal{D}_\mathcal{H}, \Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0 \wedge \mathcal{D}_\mathcal{H}, E_p, \Delta, \tau, \sigma_0 \vdash \mathrm{d.cs} \to \theta_\sigma}$$

To prove the goal, let us use $\sigma_e, \sigma_0 \in \Sigma$ previously introduced by the invocation of Theorems 2, 3 and 4. Then, the three first points of the goal are previously assumed hypotheses.
Finally, appealing to Theorem 5, we can prove final point of theorem, i.e. $\boxed{\theta_s \sim \theta_\sigma}$.

□

Theorem 2 states that every $\mathcal{H}$-VHDL design returned by the HILECOP transformation function can be elaborated. The elaboration relation verifies that a given $\mathcal{H}$-VHDL design is well-typed and well-formed w.r.t. to the VHDL language standards, and builds an elaborated version of the $\mathcal{H}$-VHDL design that will act as a simulation environment. Thus, Theorem 2 states that the HILECOP transformation function produces *acceptable* code, for instance, code that could be the input to a simulator program.

Theorem 2 (Elaboration). *For all well-defined* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$ *s.t.*

- $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$

then there exists an elaborated design $\Delta \in ElDesign$ *and a design state* $\sigma_e \in \Sigma$ *s.t.* Δ *is the elaborated version of design* d*, and* σ_e *is the default design state of* Δ*:* $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e)$.

Theorem 3 states that one can always build an initial state for every $\mathcal{H}$-VHDL design returned by the HILECOP transformation function.

Theorem 3 (Initialization). *For all well-defined* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign$, $\sigma_e \in \Sigma(\Delta)$ *s.t.*

- $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$ *and* $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e)$

then there exists a design state $\sigma_0 \in \Sigma(\Delta)$ *s.t.* σ_0 *is the initial simulation state:* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0$.

Theorem 4 states that one can always build a simulation trace over τ clock cycles for every $\mathcal{H}$-VHDL design returned by the HILECOP transformation function. This means that the simulation of an $\mathcal{H}$-VHDL design never fails when it is the result of the transformation of a well-defined SITPN.

Theorem 4 (Simulation). *For all well-defined* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign$, $\sigma_e, \sigma_0 \in \Sigma$ *s.t.*

- $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$ *and* $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e)$ *and* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0$

then there exists a simulation trace $\theta_\sigma \in \texttt{list}(\Sigma)$ *such that for all simulation environment* $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \rightarrow Ins(\Delta) \rightarrow value$*, and simulation cycle count* $\tau \in \mathbb{N}$*, design* d *yields the simulation trace* θ_σ *after* τ *simulation cycles, starting from initial state* σ_0*:*
$\mathcal{D}_{\mathcal{H}}, E_p, \Delta, \tau, \sigma_0 \vdash \text{d.cs} \rightarrow \theta_\sigma$

2.3.4 The bisimulation theorem

Here, we present the bisimulation theorem. The bisimulation theorem states that if an SITPN and its corresponding $\mathcal{H}$-VHDL design are executed/simulated over τ execution/simulation cycles, then the produced traces are semantically similar, i.e they verify the full execution trace similarity relation of Definition 9. In this thesis, we proved this particular theorem, and as said

before, we left the proofs of Theorems 2, 3 and 4 for later. We choose to focus our work on the bisimulation theorem, because it directly addresses the semantic preservation property of HILECOP's transformation function.

In the proof of Theorem 5, in the case where $\tau > 0$, we must show that the state similarity relation holds between the states produced by the first execution cycle, and then use Lemma 1 to complete the proof of similarity between the tail traces. First, we must show that the initial states of both SITPN and $\mathcal{H}$-VHDL design verify the general state similarity relation (Definition 2); this is done by appealing to Lemma 5. The first execution cycle is particular because, by definition of the SITPN full execution relation, no transitions are fired during the first rising edge. Therefore, after the first rising edge, the SITPN state is still equal to its initial state s_0. We prove that the post rising edge similarity relation is verified after the first rising edge by appealing to Lemma 15. The detailled proofs for Lemmas 5 and 15 are given in Sections A.1 and A.2.

Theorem 5 (Full bisimulation). *For all* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\tau \in \mathbb{N}$, $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $\theta_s \in \texttt{list}(S(sitpn))$, $\Delta \in ElDesign$, $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value$, $\theta_\sigma \in \texttt{list}(\Sigma)$ *s.t.*

- $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$
- $\gamma \vdash E_p \overset{env}{=} E_c$
- $E_c, \tau \vdash sitpn \xrightarrow{full} \theta_s$
- $\mathcal{D}_{\mathcal{H}}, \Delta, \emptyset, E_p, \tau \vdash \mathrm{d} \xrightarrow{full} \theta_\sigma$

then $\gamma \vdash \theta_s \sim \theta_\sigma$

Proof. Assuming the above hypotheses, let us show $\boxed{\gamma \vdash \theta_s \sim \theta_\sigma}$.
Let us perform case analysis on τ; there are two cases:

- **CASE** $\tau = 0$. By definition of the SITPN full execution and the $\mathcal{H}$-VHDL full simulation relations, we have:
  - $E_c, 0 \vdash sitpn \xrightarrow{full} [s_0]$ and $\theta_s = [s_0]$
  - $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e)$ and $\Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0$ and $\mathcal{D}_{\mathcal{H}}, E_p, \Delta, 0, \sigma_0 \vdash d.cs \to [\,]$ and $\theta_\sigma = [\sigma_0]$

  Rewriting θ_s as $[s_0]$, and θ_σ as $[\sigma_0]$, and by definition of the full execution trace similarity relation, what is left to prove is: $\boxed{\gamma \vdash s_0 \sim \sigma_0}$

  Appealing to Lemma 5, we can show $\gamma \vdash s_0 \sim \sigma_0$.

- **CASE** $\tau > 0$. By definition of the SITPN full execution relation (i.e, $E_c, \tau \vdash sitpn \xrightarrow{full} \theta_s$) and the $\mathcal{H}$-VHDL full simulation relation (i.e, $\mathcal{D}_{\mathcal{H}}, \Delta, \emptyset, E_p, \tau \vdash \mathrm{d} \xrightarrow{full} \theta_\sigma$), we have:

– $E_c, \tau \vdash s_0 \xrightarrow{\uparrow_0} s_0$ and $E_c, \tau \vdash s_0 \xrightarrow{\downarrow} s$ and $E_c, \tau - 1 \vdash sitpn, s \to \theta$ and $\theta_s = s_0 :: s_0 :: s :: \theta$

– $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e)$ and $\Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0$ and $E_p, \Delta, \tau, \sigma_0 \vdash \text{d.cs} \to \theta'$ and $\theta_\sigma = \sigma_0 :: \theta'$

Rewriting θ_s and θ_σ, the new goal is: $\boxed{\gamma \vdash (s_0 :: s_0 :: s :: \theta) \sim (\sigma_0 :: \theta')}$

By definition of the $\mathcal{H}$-VHDL simulation relation (i.e. $E_p, \Delta, \tau, \sigma_0 \vdash \text{d.cs} \to \theta'$), we have:

$E_p, \Delta, \tau, \sigma_0 \vdash d.cs \xrightarrow{\uparrow\!\!\!\!\rightsquigarrow\downarrow} \sigma, \sigma'$ and $E_p, \Delta, \tau - 1, \sigma' \vdash \text{d.cs} \to \theta''$ and $\theta' = \sigma :: \sigma' :: \theta''$

Rewriting θ', the new goal is: $\boxed{\gamma \vdash (s_0 :: s_0 :: s :: \theta) \sim (\sigma_0 :: \sigma :: \sigma' :: \theta'')}$

By definition of the full execution trace similarity relation, there are four points to prove:

1. $\boxed{\gamma \vdash s_0 \sim \sigma_0.}$ Appealing to Lemma 5, we can show $\gamma \vdash s_0 \sim \sigma_0$.
2. $\boxed{\gamma, E_c, \tau \vdash s_0 \overset{\uparrow}{\sim} \sigma.}$ Appealing to Lemma 15, we have $\gamma, E_c, \tau \vdash s_0 \overset{\uparrow}{\approx} \sigma$.

   By definition of $\gamma, E_c, \tau \vdash s_0 \overset{\uparrow}{\approx} \sigma$, we can show $\gamma, E_c, \tau \vdash s_0 \overset{\uparrow}{\sim} \sigma.$
3. $\boxed{\gamma \vdash s \overset{\downarrow}{\sim} \sigma'.}$ Appealing to Lemma 15 and 3, we have $\gamma \vdash s \overset{\downarrow}{\approx} \sigma'$.

   By definition of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma'$, we can show $\gamma \vdash s \overset{\downarrow}{\sim} \sigma'.$
4. $\boxed{\gamma \vdash \theta \overset{\uparrow}{\sim} \theta''.}$

   Appealing to Lemma 15 and 3, we have $\gamma \vdash s \overset{\downarrow}{\approx} \sigma'$.

   Then, we can appeal to Lemma 1 to show $\gamma \vdash \theta \overset{\uparrow}{\sim} \theta''.$

□

Lemma 1 is similar to Theorem 5 excepts that the execution/simulation traces are not produced starting from the initial states, but starting from two states verifying the full post falling edge state similarity relation (i.e, $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$). The SITPN execution relation and the $\mathcal{H}$-VHDL simulation relation execute one computational step at clock count τ and then decrement the clock count and call themselves recursively to produce the rest of the execution/simulation traces. Therefore, the proof of Lemma 1 is naturally done by induction over the clock count τ.

Lemma 1 (Bisimulation). *For all $sitpn$, d, γ, E_p, E_c, τ, s, θ_s, σ, θ_σ, Δ, σ_e, assume that:*

- $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$ *and* $\gamma \vdash E_p \overset{env}{=} E_c$ *and* $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} \Delta, \sigma_e$
- *Starting states are fully similar as intended after a falling edge:* $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$
- $E_c, \tau \vdash sitpn, s \to \theta_s$

- $E_p, \Delta, \tau, \sigma \vdash d.cs \rightarrow \theta_\sigma$

then $\gamma \vdash \theta_s \overset{\uparrow}{\sim} \theta_\sigma$.

Proof. Given a $sitpn$, d, γ, E_p, E_c, τ, and assuming $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$ and $\gamma \vdash E_p \overset{env}{=} E_c$ and $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} \Delta, \sigma_e$, let us show

$\forall s, \sigma, \theta_s, \theta_\sigma$ s.t. $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$ and $E_c, \tau \vdash sitpn, s \rightarrow \theta_s$ and $E_p, \Delta, \tau, \sigma \vdash d.cs \rightarrow \theta_\sigma$ then $\gamma \vdash \theta_s \overset{\uparrow}{\approx} \theta_\sigma$.

Let us reason by induction on τ.

- **BASE CASE**: $\tau = 0$. Then, $\sigma_s = \sigma_\sigma = [\,]$ and by definition of the execution trace similarity relation, we can show $\gamma \vdash [\,] \overset{\uparrow}{\approx} [\,]$.
- **INDUCTION CASE**: Assuming the following induction hypothesis

  $\forall s, \sigma, \theta_s, \theta_\sigma$ s.t. $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$ and $E_c, \tau \vdash sitpn, s \rightarrow \theta_s$ and $E_p, \Delta, \tau\sigma \vdash d.cs \rightarrow \theta_\sigma$ then $\gamma \vdash \theta_s \overset{\uparrow}{\approx} \theta_\sigma$.

  we must prove the goal at $\tau + 1$, i.e.:

  $\forall s, \sigma, \theta_s, \theta_\sigma$ s.t. $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$ and $E_c, \tau + 1 \vdash sitpn, s \rightarrow \theta_s$ and $E_p, \Delta, \tau + 1, \sigma \vdash d.cs \rightarrow \theta_\sigma$ then $\gamma \vdash \theta_s \overset{\uparrow}{\approx} \theta_\sigma$.

  Given $s, \sigma, \theta_s, \theta_\sigma$ such that $(\gamma \vdash s \overset{\downarrow}{\approx} \sigma)$ and $(E_c, \tau + 1 \vdash sitpn, s \rightarrow \theta_s)$ and $(E_p, \Delta, \tau + 1, \sigma \vdash d.cs \rightarrow \theta_\sigma)$, let us show $\gamma \vdash \theta_s \overset{\uparrow}{\approx} \theta_\sigma$.
  By definition of $(E_c, \tau + 1 \vdash sitpn, s \rightarrow \theta_s)$ and $(E_p, \Delta, \tau + 1, \sigma \vdash d.cs \rightarrow \theta_\sigma)$, we have:

  - $E_c, \tau + 1 \vdash s \xrightarrow{\uparrow} s'$ and $E_c, \tau + 1 \vdash s' \xrightarrow{\downarrow} s''$ and $E_c, \tau \vdash sitpn, s'' \rightarrow \theta$.
  - $\texttt{Inject}_{\uparrow}(\sigma, E_p, \tau + 1, \sigma_i)$ and $\Delta, \sigma_i \vdash \text{d.cs} \xrightarrow{\uparrow} \sigma_{\uparrow}$ and $\Delta, \sigma_{\uparrow} \vdash \text{d.cs} \xrightarrow{\rightsquigarrow} \sigma'$
  - $\texttt{Inject}_{\downarrow}(\sigma', E_p, \tau + 1, \sigma'_i)$ and $\Delta, \sigma'_i \vdash \text{d.cs} \xrightarrow{\downarrow} \sigma_{\downarrow}$ and $\Delta, \sigma_{\downarrow} \vdash \text{d.cs} \xrightarrow{\rightsquigarrow} \sigma''$
  - $E_p, \Delta, \tau, \sigma'' \vdash d.cs \rightarrow \theta'$
  - $\theta_s = s' :: s'' :: \theta$ and $\theta_\sigma = \sigma' :: \sigma'' :: \theta'$

Then, the new goal is: $\boxed{\gamma \vdash (s' :: s'' :: \theta) \overset{\uparrow}{\sim} (\sigma' :: \sigma'' :: \theta')}$.

By definition of the execution trace similarity relation, there are three points to prove:

1. $\boxed{\gamma \vdash s' \overset{\uparrow}{\sim} \sigma'.}$ Appealing to Lemma 3, we have $\gamma \vdash s' \overset{\uparrow}{\approx} \sigma'$.

   By definition of $\gamma \vdash s' \overset{\uparrow}{\approx} \sigma'$, we can show $\gamma \vdash s' \overset{\uparrow}{\sim} \sigma'$.

2. $\boxed{\gamma \vdash s'' \overset{\downarrow}{\sim} \sigma''.}$ Appealing to Lemmas 3 and 2, we have $\gamma, E_c, \tau \vdash s' \overset{\downarrow}{\approx} \sigma'$.

   By definition of $\gamma, E_c, \tau \vdash s' \overset{\downarrow}{\approx} \sigma'$, we can show $\gamma \vdash s' \overset{\downarrow}{\sim} \sigma'$.

3. $\boxed{\gamma \vdash \theta \overset{\uparrow}{\sim} \theta'.}$

   We can apply the induction hypothesis with $s = s''$, $\sigma = \sigma''$, $\theta_s = \theta$ and $\theta_\sigma = \theta'$. Then, what is left to prove is: $\boxed{\gamma \vdash s'' \overset{\downarrow}{\approx} \sigma''}$

   Appealing to Lemmas 3 and 2, we can show $\gamma \vdash s'' \overset{\downarrow}{\approx} \sigma''$.

□

To prove the semantic preservation property, we want to prove that a given SITPN and its translated $\mathcal{H}$-VHDL version follow the bisimulation diagram of Figure 2.3.

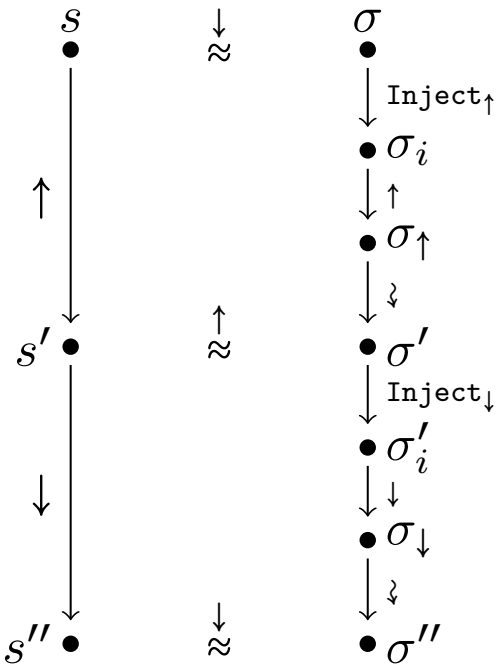

FIGURE 2.3: Bisimulation diagram over one clock cycle for a source SITPN and a target $\mathcal{H}$-VHDL design; the left part of the diagram presents the execution of an SITPN over one clock cycle, and the right part of the diagram presents the simulation of an $\mathcal{H}$-VHDL design over one clock cycle; the upper part of the diagram corresponds to the rising edge phase of the clock cycle, and the lower part illustrates the falling edge phase of the clock cycle.

The upper part of the diagram is proved by Lemma 2. First, we assume that the starting SITPN state and the starting $\mathcal{H}$-VHDL design state verify the full post falling edge state similarity relation at the beginning of the clock cycle (i.e, $s \overset{\downarrow}{\approx} \sigma$ in Figure 2.3). Then, Lemma 2 states that after the computation of a rising edge step on the SITPN part and on the $\mathcal{H}$-VHDL part the resulting states verify the full post rising edge state similarity relation. The lower part of the diagram is proved by Lemma 3. First, we assume that the starting SITPN state and the starting $\mathcal{H}$-VHDL state verify the full post rising edge state similarity relation (i.e, $s' \overset{\downarrow}{\approx} \sigma'$ in Figure 2.3). Then, Lemma 2 states that after the computation of a falling edge step on the SITPN part and on the $\mathcal{H}$-VHDL part the resulting states verify the full post falling edge state similarity relation.

Here, we present Lemma 2 and Lemma 3, along with their proofs. In the two lemmas, we added an extra hypothesis about the starting state of the $\mathcal{H}$-VHDL design: $\mathcal{D}_\mathcal{H}, \Delta, \sigma \vdash \text{d.cs} \xrightarrow{comb} \sigma$. This hypothesis states that all signal values are stable at the beginning of the considered clock phase. This means that the execution of the combinational part of the $\mathcal{H}$-VHDL design does not change the value of signals anymore. This hypothesis is mandatory to determine the expression associated to combinational signals, i.e. the *combinational equations*, at the beginning of the clock phase (see Section 2.4 for more details about combinational equations).

To prove Lemmas 2 and 3, one must show that every point of the state similarity relation in the conclusion holds. For each point, the proof is given as a separate lemma that the reader will find in Appendix A. The proof strategy to show the equalities or equivalences laid out in the state similarity relation follows the same two-fold pattern:

- First, reason on the SITPN structure and on the transformation function to determine the content of the target $\mathcal{H}$-VHDL design.
- Then, reason on the SITPN state transition relation and the $\mathcal{H}$-VHDL "simulation" relations (i.e, the $\texttt{Inject}_{clk}$, $\uparrow$, $\downarrow$ and $\rightsquigarrow$ relations) to establish the equality between the values coming from the SITPN world (i.e, marking, time counters, reset orders, etc. and also predicates) and the values of the signals declared in the $\mathcal{H}$-VHDL design and in its internal component instances.

The application of this proof strategy will be detailed in Section 2.4.

Lemma 2 (Rising edge). *For all* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $\Delta \in ElDesign$, $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value$, $\tau \in \mathbb{N}$, $s, s' \in S(sitpn)$, $\sigma_e, \sigma, \sigma_i, \sigma_\uparrow, \sigma' \in \Sigma$, *assume that:*

- $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$ *and* $\gamma \vdash E_p \overset{env}{=} E_c$ *and* $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash \mathrm{d} \xrightarrow{elab} \Delta, \sigma_e$
- $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$
- $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$
- $\texttt{Inject}_{\uparrow}(\sigma, E_p, \tau, \sigma_i)$ *and* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma_i \vdash \mathrm{d.cs} \xrightarrow{\uparrow} \sigma_\uparrow$ *and* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma_\uparrow \vdash \mathrm{d.cs} \xrightarrow{\rightsquigarrow} \sigma'$
- *State* σ *is a stable design state:* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma \vdash \mathrm{d.cs} \xrightarrow{comb} \sigma$

then $\gamma, E_c, \tau \vdash s' \overset{\uparrow}{\approx} \sigma'$.

Proof. By definition of the Full post rising edge state similarity relation, there are 8 points to prove:

1. $\forall p \in P, id_p \in Comps(\Delta)\ s'.t.\ \gamma(p) = id_p,\ s'.M(p) = \sigma'(id_p)(\texttt{s\_marking})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\big(upper(I_s(t)) = \infty \wedge s'.I(t) \leq lower(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\texttt{s\_time\_counter})\big)$
   $\wedge \big(upper(I_s(t)) = \infty \wedge s'.I(t) > lower(I_s(t)) \Rightarrow \sigma'(id_t)(\texttt{s\_time\_counter}) = lower(I_s(t))\big)$
   $\wedge \big(upper(I_s(t)) \neq \infty \wedge s'.I(t) > upper(I_s(t)) \Rightarrow \sigma'(id_t)(\texttt{s\_time\_counter}) = upper(I_s(t))\big)$
   $\wedge \big(upper(I_s(t)) \neq \infty \wedge s'.I(t) \leq upper(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\texttt{s\_time\_counter})\big)$.
3. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, s'.reset_t(t) = \sigma'(id_t)(\texttt{s\_reinit\_time\_counter})$.
4. $\forall a \in \mathcal{A}, id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a,\ s'.ex(a) = \sigma'(id_a)$.

5. $\forall f \in \mathcal{F}, id_f \in Outs(\Delta)\ s.t.\ \gamma(f) = id_f,\ s'.ex(f) = \sigma'(id_f)$.
6. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Sens(s'.M) \Leftrightarrow \sigma'(id_t)(\texttt{s\_enabled}) = \texttt{true}$.
7. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Sens(s'.M) \Leftrightarrow \sigma'(id_t)(\texttt{s\_enabled}) = \texttt{false}$.
8. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\sigma'(id_t)(\texttt{s\_condition\_combination}) = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}$
   where $conds(t) = \{c \in \mathcal{C} \mid \mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1\}$.

Each point is proved by a separate lemma:

– Apply the Rising edge equal marking lemma to solve 1.

– Apply the Rising edge equal time counters lemma to solve 2.

– Apply the Rising edge equal reset orders lemma to solve 3.

– Apply the Rising edge equal action executions lemma to solve 4.

– Apply the Rising edge equal function executions lemma to solve 5.

– Apply the Rising edge equal sensitized lemma to solve 6.

– Apply the Rising edge equal not sensitized lemma to solve 7.

– Apply the Rising edge equal condition combination lemma to solve 8.

□

Lemma 3 (Falling edge). *For all* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $\Delta \in ElDesign$, $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value$, $\tau \in \mathbb{N}$, $s, s' \in S(sitpn)$, $\sigma_e, \sigma, \sigma_i, \sigma_\downarrow, \sigma' \in \Sigma$, *assume that:*

- $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$ *and* $\gamma \vdash E_p \overset{env}{=} E_c$ *and* $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash \mathrm{d} \xrightarrow{elab} \Delta, \sigma_e$
- $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$
- $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$
- $\texttt{Inject}_\downarrow(\sigma, E_p, \tau, \sigma_i)$ *and* $\Delta, \sigma_i \vdash \mathrm{d.cs} \xrightarrow{\downarrow} \sigma_\downarrow$ *and* $\Delta, \sigma_\downarrow \vdash \mathrm{d.cs} \xrightarrow{\rightsquigarrow} \sigma'$
- *State* σ *is a stable design state:* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma \vdash \mathrm{d.cs} \xrightarrow{comb} \sigma$

then $\gamma \vdash s' \overset{\downarrow}{\approx} \sigma'$.

Proof. By definition of the Post falling edge state similarity relation, there are 11 points to prove:

1. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s'.M(p) = \sigma'(id_p)(\texttt{s\_marking})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\big(upper(I_s(t)) = \infty \wedge s'.I(t) \leq lower(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\texttt{s\_time\_counter})\big)$
   $\wedge\big(upper(I_s(t)) = \infty \wedge s'.I(t) > lower(I_s(t)) \Rightarrow \sigma'(id_t)(\texttt{s\_time\_counter}) = lower(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s'.I(t) > upper(I_s(t)) \Rightarrow \sigma'(id_t)(\texttt{s\_time\_counter}) = upper(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s'.I(t) \leq upper(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\texttt{s\_time\_counter})\big)$.
3. $\forall c \in \mathcal{C}, id_c \in Ins(\Delta)\ s.t.\ \gamma(c) = id_c,\ s'.cond(c) = \sigma'(id_c)$.
4. $\forall a \in \mathcal{A}, id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a,\ s'.ex(a) = \sigma'(id_a)$.
5. $\forall f \in \mathcal{F}, id_f \in Outs(\Delta)\ s.t.\ \gamma(f) = id_f,\ s'.ex(f) = \sigma'(id_f)$.
6. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Firable(s') \Leftrightarrow \sigma'(id_t)(\texttt{s\_firable}) = \texttt{true}$.
7. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Firable(s') \Leftrightarrow \sigma'(id_t)(\texttt{s\_firable}) = \texttt{false}$.
8. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Fired(s') \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}$.
9. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Fired(s') \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{false}$.
10. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,$
    $\sum_{t \in Fired(s')} pre(p,t) = \sigma'(id_p)(\texttt{s\_output\_token\_sum})$.
11. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,$
    $\sum_{t \in Fired(s')} post(t,p) = \sigma'(id_p)(\texttt{s\_input\_token\_sum})$.

Each point is proved by a separate lemma:

- Apply the Falling edge equal marking lemma to solve 1.
- Apply the Falling edge equal time counters lemma to solve 2.
- Apply the Falling edge equal condition values lemma to solve 3.
- Apply the Falling edge equal action executions lemma to solve 4.
- Apply the Falling edge equal function executions lemma to solve 5.
- Apply the Falling edge equal firable lemma to solve 6.
- Apply the Falling edge equal not firable lemma to solve 7.
- Apply the Falling edge equal fired lemma to solve 8.
- Apply the Falling edge equal not fired lemma to solve 9.

– Apply the Falling Edge Equal Output Token Sum lemma to solve 10.

– Apply the Falling Edge Equal Input Token Sum lemma to solve 11.

□

2.4 A detailled proof: equivalence of fired transitions

The goal of this section is to present the overall proof strategy that we adopted to establish the semantic preservation property for the HILECOP model-to-text transformation. We use the proof of Lemma 4, involved in the proof of Lemma 3, to illustrate our demonstration technics. The proof of Lemma 4 has been one complex part of the overall demonstration; we believe it is worth to be mentioned. Also, it has led to a bug detection. We give a full account on this bug detection, and on how we manage to correct it, at the end of the section.

2.4.1 An accompanied journey along the proof

The proof of Lemma 4 pertains to the set of fired transitions. In an SITPN, the firing process, based on the set of fired transitions, is responsible for the computation of the new marking, the reset orders, and the execution of functions during the rising edge phase. To prove the semantic preservation property, we must have the equivalence between the set of fired transitions as defined on the SITPN side and the set of fired transitions as defined on the $\mathcal{H}$-VHDL side. The equivalence must hold at the beginning of the rising edge phase, i.e. when the set of fired transitions will be used to compute a new SITPN state. Thus, the falling edge phase prepares the field so that the equivalence between the set of fired transitions holds at the beginning of the next rising edge phase. To express Lemma 4, we must first define the hypotheses stating that a falling edge phase happened in the course of the execution of an SITPN and its corresponding $\mathcal{H}$-VHDL design, plus some hypotheses about the similarity of the states at the beginning of the falling edge phase:

Definition 10 (Falling edge hypotheses). *Given a* $sitpn \in SITPN, d \in design, \gamma \in WM(sitpn, d)$, $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $\Delta \in ElDesign$, $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value$, $\tau \in \mathbb{N}$, $s, s' \in S(sitpn)$, $\sigma_e, \sigma, \sigma_i, \sigma_\downarrow, \sigma' \in \Sigma$, *assume that:*

- *SITPN* $sitpn$ *translates into* $\mathcal{H}$*-VHDL design* d *and yields a binder* γ: $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$
- *Simulation/Execution environments are similar:* $\gamma \vdash E_p \overset{env}{=} E_c$
- Δ *is the elaborated version of design* d*, and* σ_e *is the default design state of* Δ: $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash \mathrm{d} \xrightarrow{elab} \Delta, \sigma_e$
- *Starting states are similar according to the full post rising edge similarity relation:* $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$
- *On the SITPN side, the execution of a falling edge phase starting from state* s *leads to state* s'*:* $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$

- *On the $\mathcal{H}$-VHDL side, the simulation of a falling edge phase starting from state σ leads to state σ':* $\texttt{Inject}_{\downarrow}(\sigma, E_p, \tau, \sigma_i)$ *and* $\Delta, \sigma_i \vdash \text{d.cs} \xrightarrow{\downarrow} \sigma_{\downarrow}$ *and* $\Delta, \sigma_{\downarrow} \vdash \text{d.cs} \xrightarrow{\rightsquigarrow} \sigma'$
- *State σ is a stable design state:* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma \vdash \text{d.cs} \xrightarrow{comb} \sigma$

The hypotheses of Definition 10 are used in all the lemmas expressing some properties about the falling edge phase. Therefore, Definition 10 enables the conciser expression of these lemmas. Then, we can express Lemma 4 as follows:

Lemma 4 (Falling edge equal fired)**.** *For all $sitpn$, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_{\downarrow}$, σ' that verify the hypotheses of Definition 10, then for all $t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $t \in Fired(s') \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}$.*

Now, let us detail the proof of Lemma 4. To prove Lemma 4, we must reason on a given transition t of the input SITPN $sitpn$ and a TCI id_t in the output $\mathcal{H}$-VHDL design d. Transition t and TCI id_t are bound together through the γ binder returned by the transformation function. This means that the TCI id_t structurally represents the transition t in the output $\mathcal{H}$-VHDL design d. In this setting, we want to prove that t is in the set of fired transitions at the end of the falling edge phase if and only if the `fired` port of id_t equals `true` at the end of the falling edge phase. Formally, we want to prove: $\boxed{t \in Fired(s') \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}.}$

As a reminder, the expression $Fired(s')$ qualifies the set of fired transitions at the SITPN state s', and $\sigma'(id_t)(\texttt{fired})$ denotes the value of the `fired` port of TCI id_t at design state σ'. The expression $\sigma'(id_t)$ denotes the internal state, i.e. a design state, of TCI id_t at state σ'.

To prove the equivalence, we must first look at the definition of the set of fired transitions on the SITPN side and on the $\mathcal{H}$-VHDL side, and then think of a way to relate the two definitions.

On the SITPN side, the set of fired transitions receives an intentional and recursive definition (see Definition **??**) depending on a given SITPN state. In Lemma 4, we are interested in the definition of the set of fired transitions at state s', i.e. the state at the end of the falling edge phase. A transition belongs to the set of fired transitions if it is *firable* (see Definition **??**) and sensitized by the *residual* marking (see Sections **??** and **??**) at the considered SITPN state. Figure 2.4 gives the set of fired transitions, i.e. $Fired(s)$, on an example SITPN at a given state s. Here, transitions t_a, t_b and t_c are all firable at state s; however, only transition t_c is sensitized by the residual marking.

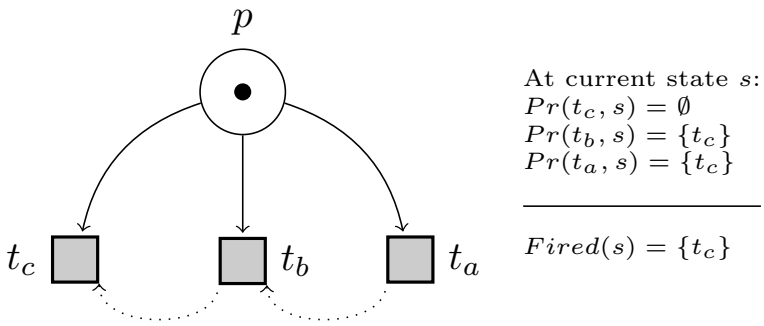

FIGURE 2.4: The set of fired transitions on an example SITPN at a given SITPN state s; on the left side, the dotted arrows indicates the priority relation between the three transitions (t_c is the top-priority transition); on the right side, each transition is associated to its Pr set, which is necessary to compute the residual marking.

The computation of the residual marking involves the Pr sets, which are, for a given transition t and a state s, the set of transitions with a higher firing priority than t which are actually fired at s. This is where the recursive definition of the set of fired transitions begins. The definition is correct, i.e. the recursion ends, if the priority relation is a strict order over the set of transitions, and therefore, there are always transitions of top-priority (e.g, t_c in Figure 2.4). The condition of the priority relation being a strict order over the set of transitions is part of the definition of a well-defined SITPN (see Definition **??**). By definition, top-priority transitions have an empty Pr set. Indeed, there exist no transition with a higher firing priority than a top-priority transition. Thus, a top-priority transition that is firable is also fired. Note that one can not determine the Pr set of a transition before having determined the firing status of all the transitions with a higher firing priority. For instance, in Figure 2.4, it is impossible to know the content of $Pr(t_a, s)$ before having determined if transition t_b is fired or not. To know if t_b is fired or not, we must determine the content of $Pr(t_b, s)$. To do so, we must first determine the firing status of t_c. Even though the definition of the set of fired transitions is very declarative, this hints at a natural way to establish an algorithm to build the set of fired transitions at a given SITPN state.

On the $\mathcal{H}$-VHDL side, the set of fired transitions is defined through the value of the `fired` port of TCIs. The transition design declares an output port of the Boolean type with the identifier `fired`. What we want to prove in Lemma 4 is that, at the end of the falling edge phase (i.e. at state σ'), the value of the `fired` port of a TCI reflects the firing status of the corresponding transition. The `fired` port is a combinational signal. This means that its value depends on an equation that is verified when all signals are stable, i.e. at the end of the stabilization phases happening during the simulation. In the point of view of the circuit synthesis, this equation reflects the wiring of the port in the described hardware circuit. Figure 2.5 shows a part of the transition design architecture describing how the `fired` port is connected to the other internal signals.

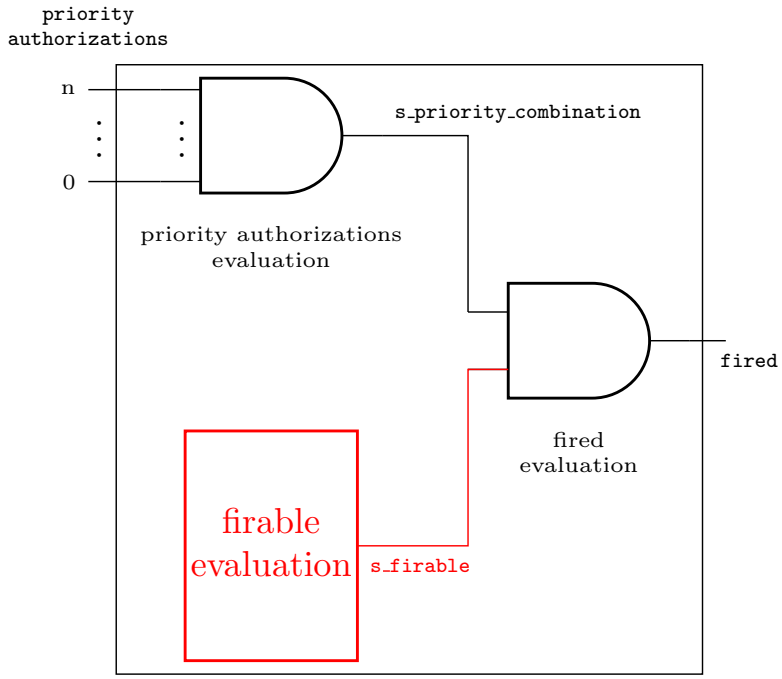

FIGURE 2.5: Wiring of the `fired` output port in the transition design architecture; on the left side is the input interface of the transition design; on the right side is the output interface of the transition design, with the `fired` port; in red are the parts of the architecture that depend on synchronous logic and in black are the parts that are purely combinational.

In Figure 2.5, the labels underneath the *and* logic ports and inside the block denote the names of the processes defined in the transition design architecture as VHDL code. As a matter of fact, Figure 2.5 is a graphic transcription of the code defining the transition design architecture. Therefore, by looking at the VHDL code, we are able to determine the combinational equation associated to the `fired` port. Given a TCI id_t in a top-level design and a state σ denoting a current stable state of the design (remember that a combinational equation holds when all signal values are stable), the `fired` port equation at σ is:

$$\sigma(id_t)(\texttt{fired}) = \sigma(id_t)(\texttt{s\_firable}) \,.\, \sigma(id_t)(\texttt{s\_priority\_combination}) \tag{2.2}$$

Equation (2.2) states that the value of the `fired` port is a simple "and" expression[1] between the value of the internal signal `s_firable` and `s_priority_combination`.

Remark 1 (Signals and combinational equations)**.** *In the proceeding of the proof, a lot of combinational equations are established (e.g, Equation (2.2)). These equations relate the value of a given signal to the value of other signals or expressions. All these equations are deduced by applying the $\mathcal{H}$-VHDL semantics rules on the internal behavior (i.e., the processes) of the `transition` and the `place` designs. A combinational equation is always the result of a signal assignment statement happening inside the*

[1] To differentiate the formulas of the intuitionistic logic from the expressions of the Boolean logic, we use (".", "+") to denote the *and* and *or* operators in Boolean expressions, and ($\wedge$,$\vee$) to denote the conjunction and the disjunction in the intuitionistic formulas.

statement body of a process. For instance, in the transition design, the `fired_evaluation` *process, presented in Listing 2.1, assigns the* `fired` *output port. Reasoning on the* `fired_evaluation` *process statement body and on the $\mathcal{H}$-VHDL semantics rules permits us to deduce Equation (2.2).*

```
fired_evaluation : process (s_firable, s_priority_combination)
begin
   fired ⇐ s_firable and s_priority_combination;
end process fired_evaluation;
```

LISTING 2.1: The `fired_evaluation` process in the transition design architecture; its statement body assigns the `fired` output port; symbol ⇐ is the signal assignment operator.

Listing 2.2 presents the `priority_authorizations_evaluation` *process, responsible for the assignment of the* `s_priority_combination` *in the transition design.*

```
priority_authorization_evaluation : process(priority_authorizations)
   variable v_priority_combination : std_logic;
begin
   v_priority_combination := '1';

   for i in 0 to input_arcs_number − 1 loop
     v_priority_combination := v_priority_combination and priority_authorizations(i);
   end loop;

   s_priority_combination ⇐ v_priority_combination; -- Assignment of the result
end process priority_authorization_evaluation;
```

LISTING 2.2: The `priority_authorizations_evaluation` process in the transition design's architecture. The local variable `v_priority_combination` accumulates the product of the subelements of the `priority_authorizations` input port in the `for` loop; then the last statement assigns the value of `v_priority_combination` to the `s_priority_combination` internal signal.

Equation (2.3) gives the combinational equation deduced from the execution of the `priority_authorizations_evaluation` *process for a given TCI id_t in a top-level design d. State σ denotes the current state of d, and $\sigma(id_t)$ denotes the internal state of id_t at state σ. The elaborated design Δ is the elaborated version of design d, and $\Delta(id_t)$ is the elaborated version of id_t.*

$$\sigma(id_t)(spc) = \prod_{i=0}^{\Delta(id_t)(input\_arcs\_number)-1} \sigma(id_t)(priority\_authorizations)[i] \tag{2.3}$$

In Equation (2.3), `spc` is an alias for the `s_priority_combination` *signal. The* `for` *loop of the* `priority_authorization_evaluation` *process has been converted into a product expression where the index i follows the bounds of the loop. The* `priority_authorizations` *signal is an input port of type* `array`*, thus we use the bracketed notation $a[i]$ to access the element of index i in array a. Also, we know that* `input_arcs_number` *identifies a generic constant of the transition design, thus, we can retrieve its value in the elaborated design $\Delta(id_t)$.*

*In the proofs laid out in Appendix A and in this chapter, we do not detail how the execution of processes' statement body permit to deduce combinational equations. We find that the proofs are easier to follow without entering in so much details. We let aside the task of proving that these equations hold until the time of the mechanization with the Coq proof assistant. For now, the reader can convince himself/herself that an equation holds by looking at the code of the `place` and the `transition` designs (see Appendices **??** and **??**).*

Now that we know which combinational equation is attached to the value of the output port `fired` for a given TCI, we must relate this equation to the definition of the set of fired transitions on the SITPN side. By definition of the set of fired transitions, we know that $t \in Fired(s')$ is equivalent to $t \in Firable(s') \wedge t \in Sens(s'.M - \sum_{t_i \in Pr(t,s')} pre(t_i))$ where $Pr(t,s') = \{t' \mid t' \succ t \wedge t' \in Fired(s')\}$. By definition of the `fired` port equation, we know that $\sigma'(id_t)(\texttt{fired}) = \sigma'(id_t)(\texttt{s\_firable}) \,.\, \sigma'(id_t)(\texttt{s\_priority\_combination})$. Using these definitions to rewrite the terms of the current goal, the new goal to prove is:

$$t \in Firable(s') \wedge t \in Sens(s'.M - \sum_{t_i \in Pr(t,s')} pre(t_i)) \Leftrightarrow$$
$$\sigma'(id_t)(\texttt{s\_firable}) \,.\, \sigma'(id_t)(\texttt{s\_priority\_combination}) = \texttt{true}$$

Thanks to Lemma 39, we know that $t \in Firable(s')$ iff $\sigma'(id_t)(\texttt{s\_firable}) = \texttt{true}$. Then, we can get rid of these two terms in the current goal; what is left to prove is:

$$t \in Sens(s'.M - \sum_{t_i \in Pr(t,s')} pre(t_i)) \Leftrightarrow \sigma'(id_t)(\texttt{s\_priority\_combination}) = \texttt{true}$$

Based on Equation (2.3), we can replace the value of the `s_priority_combination` signal by its equivalent product expression:

$$t \in Sens(s'.M - \sum_{t_i \in Pr(t,s')} pre(t_i)) \Leftrightarrow$$
$$\Big(\prod_{i=0}^{\Delta(id_t)(\texttt{input\_arcs\_number})-1} \sigma'(id_t)(\texttt{priority\_authorizations})[i]\Big) = \texttt{true}$$

Then, the proof is in two parts:

1. Assuming $t \in Sens(s'.M - \sum_{t_i \in Pr(t,s')} pre(t_i))$, let us show that
$$\Big(\prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i]\Big) = \texttt{true}.$$
2. Assuming $\Big(\prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i]\Big) = \texttt{true}$, let us show that
$$t \in Sens(s'.M - \sum_{t_i \in Pr(t,s')} pre(t_i)).$$

Let us prove the first point. Assuming that $t \in Sens(s'.M - \sum_{t_i \in Pr(t,s')} pre(t_i))$, let us show

$$\boxed{\left(\prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i]\right) = \texttt{true}.}$$

To prove the current goal, we can equivalently show that:

$$\boxed{\forall i \in [0, \Delta(id_t)(\texttt{ian}) - 1],\ \sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}.}$$

For a given $i \in [0, \Delta(id_t)(\texttt{ian}) - 1]$, let us show that $\boxed{\sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}.}$ As shown in Figure 2.5, the `priority_authorizations` signal is an input port of the `transition` design. Therefore, to know what is the value of the element i-th element of the `priority_authorizations` port at state $\sigma'(id_t)$, we must know how the `priority_authorizations` port is connected in the top-level design. Basing ourselves on the transformation function, the connection of the `priority_authorizations` port for the TCI id_t depends on the set of input places of the transition t. If the set of input places of t is empty, then, all elements of the `priority_authorizations` port are connected to the constant `true`, and proving the goal is trivial. If the set of input places of t is not empty, then, the connection of the i-th element of the `priority_authorizations` port reflects the connection of some place p to the transition t by an input arc. Then, we must reason on the nature of the input arc connecting p to t. The interested case happens when p and t are connected by a `basic` arc, and when the conflicts in the output transitions of p are handled by the priority relation. In that case, the i-th element of the `priority_authorizations` input port of the TCI id_t is connected to the j-th element of the `priority_authorizations` output port of the PCI id_p. Figure 2.6 shows the connection of the `priority_authorizations` port between the component instances id_p and id_t.

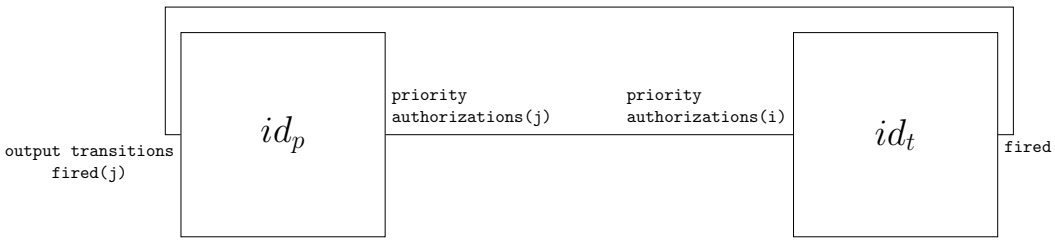

FIGURE 2.6: Connection of the j-th element of the `priority_authorizations` output port of the PCI id_p to the i-th element of the `priority_authorizations` input port of the TCI id_t; also the `fired` output port of id_t is connected to the j-th element of the `output_transitions_fired` input port of id_p.

Thus, we know that the value of the i-th element of the `priority_authorizations` input port of id_t is bound to the value of the j-th element of the `priority_authorizations` output port of id_p. Therefore, to show that $\boxed{\sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}}$, we must now show that $\boxed{\sigma'(id_p)(\texttt{pauths})[j] = \texttt{true}.}$ We must now look at the architecture of the `place` design to determine the combinational equation associated to the j-th element of the `priority_authorizations` output port. Figure 2.7 illustrates the wiring of the `priority_authorizations` output port in a `place` design.

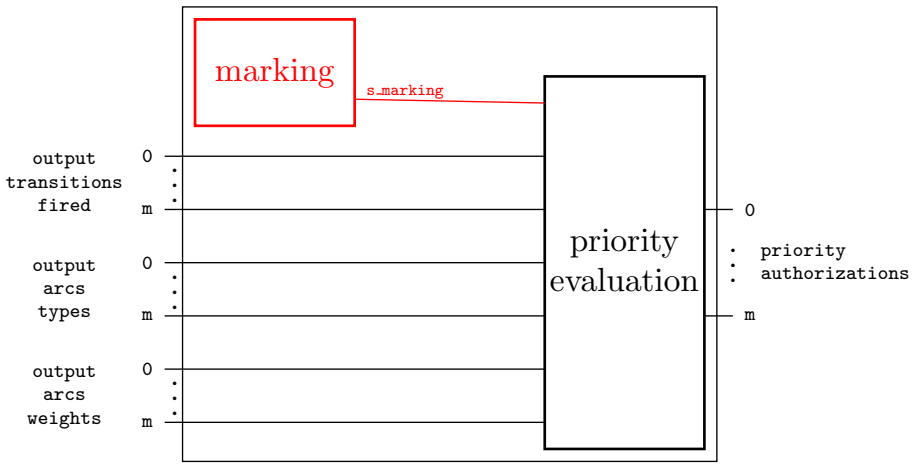

FIGURE 2.7: Wiring of the `priority_authorizations` output port in the architecture of the place design; the input port interface is on the left side and the output port interface is on the right side; the synchronous parts are in red and the combinational ones are in black.

Figure 2.7 shows that the value of the elements of the `priority_authorizations` output port is computed by the `priority_evaluation` process. This process reads the value of the `s_marking` signal, assigned by the synchronous process `marking`. It also reads the value of the `output_transitions_fired`, `output_arcs_types` and `output_arcs_weights` input ports. In Figure 2.7, the ports of the input and output interface are composite ports (i.e., of the array type) with an upper bound index equal to `m`. The number `m` is equal to the expression `output_arcs_number`-1, where `output_arcs_number` is a generic constant of the place design. The value of the `output_arcs_number` constant is set at the generation of the generic map of a PCI id_p, and is equal to the number of output transitions of place p. Listing 2.3 presents the code of the `priority_evaluation` process defined in the architecture of the `place` design.

```
1  priority_evaluation : process (output_transitions_fired, s_marking, output_arcs_types,
       output_arcs_weights)
2    variable v_saved_output_token_sum : local_weight_t;
3  begin
4    v_saved_output_token_sum := 0;
5
6    for k in 0 to output_arcs_number - 1 loop
7
8      priority_authorizations(k) <= (s_marking - v_saved_output_token_sum >=
           output_arcs_weights(k));
9
10     if (output_transitions_fired(k) = '1') and (output_arcs_types(k) = arc_t(BASIC)) then
11       v_saved_output_token_sum := v_saved_output_token_sum + output_arcs_weights(k);
12     end if;
13
14   end loop;
15 end process priority_evaluation;
```

LISTING 2.3: The `priority_evaluation` process in the place design's architecture.

In the statement body of the `priority_evaluation` process, each subelement of the `priority_authorizations` output port is assigned at Line 8 inside the `for` loop. The statement of Line 8 assigns the result of the test `s_marking - v_saved_output_token_sum >= output_arcs_weights(k)` to the k-th element of `priority_authorizations`. The test checks that the value of the `s_marking` signal, representing the current marking of the PCI, minus the value of the local variable `v_saved_output_token` is greater than or equal to the value of the k-th element of the `output_arcs_weights` signal. The test corresponds to the test of sensitization by the residual marking for the TCI connected through index k.

Getting back to our proof, the following combinational equation holds for the j-th element of the `priority_authorizations` port at state σ':

$$\sigma'(id_p)(\texttt{pauths})[j] = (\sigma'(id_p)(\texttt{s\_marking}) - \texttt{vsots} \geq \sigma'(id)(\texttt{output\_arcs\_weights})[j]) \quad (2.4)$$

Then, rewriting the goal with Equation (2.4), the new goal is:

$$\boxed{(\sigma'(id_p)(\texttt{s\_marking}) - \texttt{vsots} \geq \sigma'(id)(\texttt{output\_arcs\_weights})[j]) = \texttt{true}.}$$

Here $\geq$ denotes a Boolean operator, i.e. $\geq \in \mathbb{N} \to \mathbb{N} \to \mathbb{B}$. As the $\geq \subseteq (\mathbb{N} \times \mathbb{N})$ relation is decidable for all pairs of natural numbers, we can interchange an expression $a \geq b = \texttt{true}$ with $a \geq b$ where $a, b \in \mathbb{N}$. We will generalize this practice to every Boolean operator having a corresponding decidable relation. Thus, the new goal is:

$$\boxed{\sigma'(id_p)(\texttt{s\_marking}) - \texttt{vsots} \geq \sigma'(id)(\texttt{output\_arcs\_weights})[j].}$$

Here, the term `vsots` corresponds to the value of the local variable `v_saved_output_token_sum` at the moment of the assignment in the `for` loop. By looking at the code of Listing 2.3 (Lines 10 to 12), we can deduce the value of the `vsots` variable:

$$\texttt{vsots} = \sum_{l=0}^{j-1} \begin{cases} \sigma'(id_p)(\texttt{oaw})[l] \texttt{ if } \sigma'(id_p)(\texttt{otf})[l] \,.\, (\sigma'(id_p)(\texttt{oat})[l] = \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases} \quad (2.5)$$

The `vsots` term is equal to the sum of the output arc weights for all TCIs, representing output transitions of p, connected through an index l comprised between 0 and $j-1$. An output arc weight is taken into account in the sum only if the TCI connected through index l has a `fired` port equals to `true` (i.e. the `output_transitions_fired` input port of id_p equals `true` at index l) and is linked to the place p through a `basic` input arc (i.e. the `output_arcs_types` input port of id_p equals `basic` at index l).

Based on the fact that all conflicts in the output transitions of place p are handled with the priority relation, then, as a property deduced from the HILECOP transformation function, we know that the order of the indexes from 0 to `output_arcs_number` -1 reflects the priority order of the output transitions of place p. Therefore, the indexes from 0 to $j-1$ are linked to transitions with a higher firing priority than the transition connected to the index j. Figure 2.8 reuses the SITPN of Figure 2.4 to illustrate how the indexes are ordered when the connection

between the PCI id_p and its output TCIs id_{t_a}, id_{t_b} and id_{t_c} is set (i.e., in the course of the model-to-text transformation).

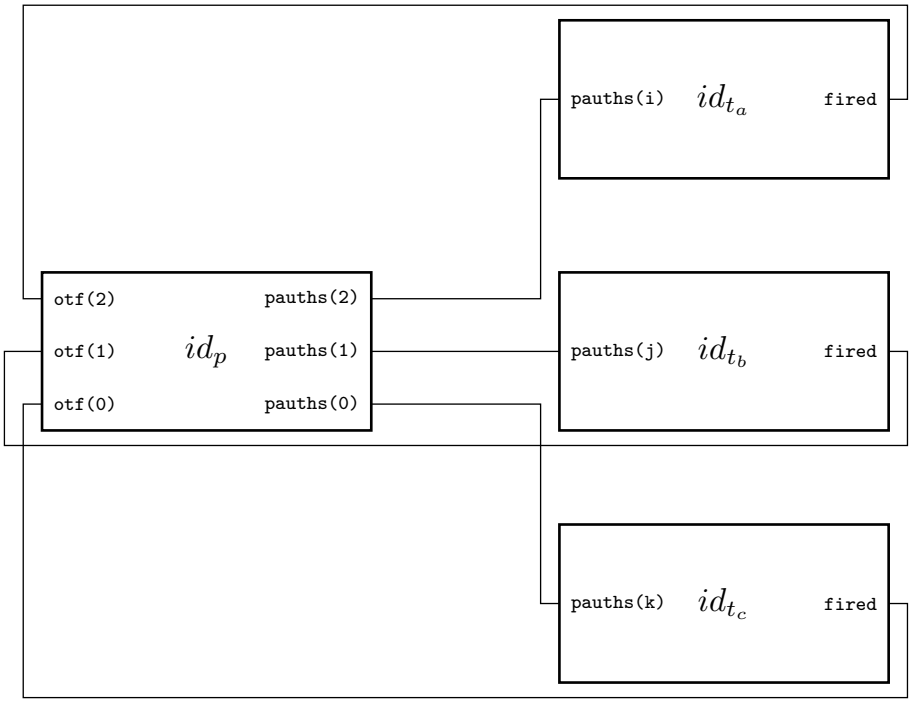

FIGURE 2.8: Connection between the `priority_authorizations` output port of PCI id_p and the `priority_authorizations` input port of TCIs id_{t_a}, id_{t_b} and id_{t_c}, and between the `output_transitions_fired` input port of id_p and the `fired` ports of id_{t_a}, id_{t_b} and id_{t_c}. `pauths` stands for `priority_authorizations` and `otf` stands for `output_transitions_fired`.

In Figure 2.8, the indexes in the interface of id_p respect the priority order of the output transitions. The index increases as the priority level of the connected TCI decreases. Thus, id_{t_c} is connected to index 0 as transition t_c is the top-priority transition in the output transitions of p, id_{t_b} is connected to index 1 as $t_c \succ t_b$, etc.

As a reminder, the current goal to prove is:

$$\sigma'(id_p)(\texttt{s\_marking}) - \texttt{vsots} \geq \sigma'(id)(\texttt{output\_arcs\_weights})[j].$$

The current goal is the $\mathcal{H}$-VHDL implementation of the test that the residual marking in place p enables transition t. At the beginning of the proof, we assumed that transition t is sensitized by the residual marking for all its input places, i.e. $t \in Sens(s'.M - \sum_{t_i \in Pr(t,s')} pre(t_i))$. By looking at the definition of the *Sens* set (see Definition **??**), and knowing that a basic arc of weight ω connects place p to transition t, we can deduce that $s'.M(p) - \sum_{t_i \in Pr(t,s')} pre(p, t_i) \geq \omega$. Now, we must relate the terms of the latter formula to the terms of the goal. We can easily show, appealing to Lemma 32, that $s'.M(p)$ equals $\sigma'(id_p)(\texttt{s\_marking})$. Then, by construction,

and knowing that TCI id_t is connected to PCI id_p through the index j, we can deduce that the j-th element of the `output_arcs_weights` input port denotes the weight of the arc between place p and transition t, i.e. the natural number ω. The last thing to show is the equality between the two sum terms:

$$\sum_{t_i \in Pr(t,s')} \begin{cases} \omega \texttt{ if } pre(p,t_i) = (\omega, \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases} = \sum_{l=0}^{j-1} \begin{cases} \sigma'(id_p)(\texttt{oaw})[l] \texttt{ if } \sigma'(id_p)(\texttt{otf})[l] \,.\, (\sigma'(id_p)(\texttt{oat})[l] = \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases}$$

On the upper part of the equality, we have unfolded term $\sum_{t_i \in Pr(t,s')} pre(t_i)$ to its full definition (see Notation **??** in Section **??**). On the lower part is the full definition of the `vsots` term. Let us rewrite the two sum terms in a manner that will come handy to prove the equality. Let us define the Pr_p set, which is the set of fired transitions with a higher priority than t that are conflicting with t through the place p:

$$t_i \in Pr_p \equiv t_i \succ t \wedge \exists \omega \textit{ s.t. } pre(p,t_i) = (\omega, \texttt{basic}) \wedge t_i \in Fired(s')$$

Let us also define the IPr_p set, which the set of indexes from 0 to $j-1$ for which the `otf` port of id_p equals `true` at state σ' and the `oat` port of id_p equals `true` at state σ':

$$l \in IPr_p \equiv l \in [0, j-1] \wedge (\sigma'(id_p)(\texttt{oat})[l] = \texttt{basic}) \wedge (\sigma'(id_p)(\texttt{otf})[l] = \texttt{true})$$

We can equivalently rewrite the goal as follows: $\sum_{t_i \in Pr_p} \texttt{fst}(pre(p,t_i)) = \sum_{l \in IPr_p} \sigma'(id_p)(\texttt{oaw})[l]$

In the left sum term, the $pre(p,t_i)$ always returns a couple $(\omega, \texttt{basic})$ as t_i verifies that there exists an ω such that $pre(p,t_i) = (\omega, \texttt{basic})$. Thus, the expression $\texttt{fst}(pre(p,t_i))$ denotes the first element of the couple returned by $pre(p,t_i)$, i.e. the weight ω.

Then, to prove the above equality, we must show that there exists a bijection β between Pr_p and IPr_p such that for all $t_i \in Pr_p$, we have $\texttt{fst}(pre(p,t_i)) = \sigma'(id_p)(\texttt{oaw})[\beta(t_i)]$. By property of the transformation function, we know that the order of the indexes in the `priority_authorizations` output port of id_p reflects the priority order of the conflicting output transitions of place p (see Figure 2.8). Then, we can exhibit a bijection β_0 between the output transitions of p with a higher priority than t and the indexes l of interval $[0, j-1]$ verifying $\sigma'(id_p)(\texttt{oat})[l] = basic$. However, to build a complete bijection β from Pr_p to IPr_p, we need to know that for a transition t_i that is a conflicting transition of place p with a higher priority than t, we have $t_i \in Fired(s') \Leftrightarrow \sigma'(id_p)(\texttt{otf})[\beta_0(t_i)] = \texttt{true}$. By property of the transformation function, we know that the element of index $\beta_0(t_i)$ of the `otf` input port of PCI id_p is in fact connected to the `fired` output port of TCI id_{t_i}. Thus, what we must assume to build a bijection from Pr_p to IPr_p is $t_i \in Fired(s') \Leftrightarrow \sigma'(id_{t_i})(\texttt{fired}) = \texttt{true}$. This is exactly the property of equivalence between the set of fired transitions and the value of the `fired` output ports that we are currently trying to prove.

Thus, to carry out the proof, we need a strong hypothesis stating that the equivalence between the set of fired transitions and the `fired` ports holds for all transitions with a higher firing priority than t, thus including the ones that are in conflict with t through place p. Therefore, we must think of a way to build the set of fired transitions iteratively such that the previous hypothesis becomes an invariant over the many iterations. The recursive definition of the set of fired transitions naturally calls for a proof by structural induction over the set of fired transitions. As stated before, the actual definition of the set of fired transitions is very declarative. However, we can easily convert it into an algorithm that will build the set iteratively. The result is Algorithm 1.

Algorithm 1: fired(s)

Data: An SITPN state s
Result: Returns the set of fired transitions at state s

1 $F \leftarrow \emptyset$
2 $T_s \leftarrow T$
3 **while** $T_s \neq \emptyset$ **do**
4 $\quad t \leftarrow$ `GetTopPriorityTransition(`T_s`,` $\succ$`)`
5 $\quad$ **if** $t \in Firable(s)$ ***and*** $t \in Sens(s.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$ **then** $F \leftarrow F \cup \{t\}$
6
7 $\quad T_s \leftarrow T_s \setminus \{t\}$
8 **return** F

Algorithm 1 builds the set of fired transitions at state s by iterating over the set of transitions T. Local variables are initialized in the two first lines. Variable F carries the set of fired transitions, which is initially empty. Variable T_s represents the set of transitions still to be processed; T_s is equal to T at the beginning of the algorithm. At Line 3, the `while` loop iterates until all transitions of the T_s set have been elected to be fired or have been discarded. At Line 4, function `GetTopPriorityTransition` returns a top-priority transition of T_s, i.e. a transition t for which there exists no transition t' in T_s such that $t' \succ t$. The statement of Line 5 tests if the top-priority transition t is firable at state s and is sensitized by the residual marking computed by the expression $s.M - \sum_{t_i \in Pr(t,F)} pre(t_i)$. Here, $Pr(t,F)$ is the set of transitions with the higher priority than t that are in the set F, i.e.: $Pr(t,F) = \{t_i \mid t_i \succ t \wedge t_i \in F\}$. We know that the following property holds: all fired transitions with a higher firing priority than t and that have been elected to be fired are inside the set F. Therefore, F contains all the transitions necessary to compute the residual marking that is necessary to elect the transition t as a fired transition; if t passes the test of Line 5 then it joins the set F. The statement of Line 7 withdraws the transition t from the set T_s before beginning another iteration. Because the priority relation $\succ$ is a strict order over the set of transitions T, we can always find a top-priority transition in T_s. Thus, there can be no iteration where T_s is not decreasing. Thus, the algorithm always terminates and returns the set of fired transitions at state s.

Being more accustomed to handle relations while performing a proof, we make a relational definition of Algorithm 1 through the definition of the $IsFiredSet$ and the $IsFiredSetAux$ relations given in Definition 12 and 13. Definition 11 states that a given transition is fired in relation

to the *IsFiredSet* relation.

Definition 11 (Fired). *A transition $t \in T$ is said to be fired at the SITPN state $s = <M, I, reset_t, ex, cond>$, iff there exists a subset $Fset \subseteq T$ such that $IsFiredSet(s, Fset)$ and $t \in Fset$.*

Definition 12 (IsFiredSet). *Given an $sitpn \in SITPN$, a SITPN state $s \in S(sitpn)$, and a subset $Fset \subseteq T$, the IsFiredSet relation is defined as follows:*
$IsFiredSet(s, Fset) \equiv IsFiredSetAux(s, T, \emptyset, Fset)$

Definition 13 (IsFiredSetAux). *The $IsFiredSetAux$ relation is defined by the following rules:*

$$\textsc{FSetEmp}\ \frac{}{IsFiredSetAux(s, \emptyset, F, F)}$$

$$\textsc{FSetFired}\ \frac{t \in Firable(s) \quad t \in Sens(s.M - \sum_{t_i \in Pr(t,F)} pre(t_i)) \quad IsFiredSetAux(s, T_s, F \cup \{t\}, Fset)}{IsFiredSetAux(s, T_s \cup \{t\}, F, Fset)} \quad \begin{array}{l} \nexists t' \in T_s \text{ s.t. } t' \succ t \\ Pr(t,F) = \{t' \mid t' \succ t \wedge t' \in F\} \end{array}$$

$$\textsc{FSetNotFirable}\ \frac{t \notin Firable(s) \quad IsFiredSetAux(s, T_s, F, Fset)}{IsFiredSetAux(s, T_s \cup \{t\}, F, Fset)} \quad \nexists t' \in T_s \text{ s.t. } t' \succ t$$

$$\textsc{FSetNotSens}\ \frac{t \notin Sens(s.M - \sum_{t_i \in Pr(t,F)} pre(t_i)) \quad IsFiredSetAux(s, T_s, F, Fset)}{IsFiredSetAux(s, T_s \cup \{t\}, F, Fset)} \quad \begin{array}{l} \nexists t' \in T_s \text{ s.t. } t' \succ t \\ Pr(t,F) = \{t' \mid t' \succ t \wedge t' \in F\} \end{array}$$

We are now satisfied with the definition of the set of fired transitions provided by the *IsFiredSet* relation and the *IsFiredSetAux* relation. Therefore, we give a new expression to Lemma 4 by using the *IsFiredSet* relation to qualify the set of fired transitions instead of using the first declarative definition. The result is Lemma 43.

The full formal proof of Lemma 43 is given in Section A.4 of Appendix A. The inductive definition of the *IsFiredSetAux* relation permits us to express the hypothesis that we lacked to perform the proof of Lemma 4. The hypothesis saying that for a given transition t, the "fired" equivalence holds for all transitions with a higher firing priority. This is stated in the "extra" hypothesis used in Lemma 44.

2.4.2 A report on a bug detection

In the previous section, we showed the equivalence between fired transitions and `fired` port values at the end of the falling edge phase. In a previous definition of the SITPN state, preceding the bug detection, the set of fired transitions was a member of the SITPN state record. For a given $sitpn \in SITPN$, we defined an SITPN state s by the record $s = <Fired, M, I, cond, ex>$ where *Fired* was the set of fired transitions. The *Fired* set was involved in the computation of time counter values during the falling edge phase. Thus, we needed the proof that the equivalence between the set of fired transitions and the value of the `fired` ports was effective at the

beginning of the falling edge phase. In the previous SITPN semantics, the set of fired transitions stayed the same during the rising edge phase. Therefore, between two SITPN states s, s' verifying the rising edge state transition relation, i.e. $s \xrightarrow{\uparrow} s'$, we had $s.Fired = s'.Fired$. However, we showed that it wasn't the case on the $\mathcal{H}$-VHDL side, i.e. the values of the `fired` ports in TCIs would not stay the same during the rising edge phase. Thus, the equivalence fired transitions and `fired` port values at the end of the falling edge phase. The consequence was a divergence between the value of time counters and the value of the `s_time_counter` signals, both computed during the falling edge phase. Figure 2.9 shows a case of divergence between time counters and `s_time_counter` signals values in the course of an execution.

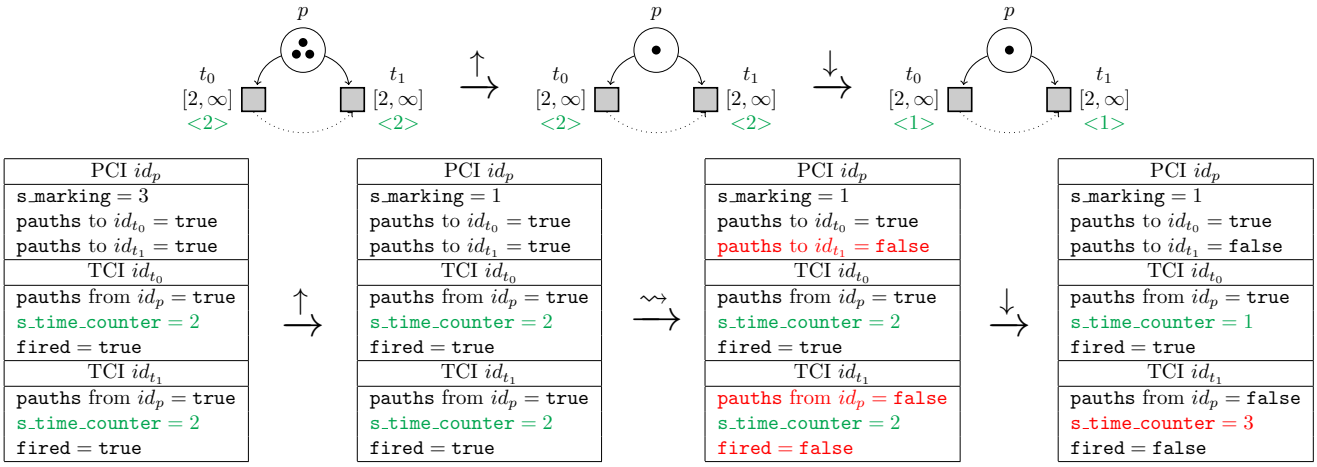

FIGURE 2.9: Bug detection: divergence between the value of time counters and the value of the `s_time_counter` signals due to the loss of the firing status information during the stabilization phase; the value of time counters and of the `s_time_counter` signals are in green; the value of diverging signals are in red.

In Figure 2.9, during the stabilization phase coming right after the rising edge of the clock, the value of the `fired` port of TCI id_{t_1} passes to `false`. After the update of the `s_marking` signal value during the rising edge phase, PCI id_p computes new priority authorizations for its output TCIs. As the marking is only sufficient to fire transition t_0 but not transition t_1, PCI id_p indicates to TCI id_{t_1} that it no longer has the authorization to fire. Consequently, through the connection of `priority_authorizations` ports, the value of the `fired` port of id_{t_1} is set to `false`. Following the rules of the SITPN semantics, on the next falling edge, the value of time counters must be reset for transition t_0 and t_1, because both were fired at the previous rising edge. As a part of the behavior of a TCI, the `time_counter` process, executed at the falling edge of the clock, resets the value of the `s_time_counter` signal given that the value of the `fired` port is true. Thus, as the value of the `fired` port of TCI id_{t_1} is false at the falling edge, the `time_counter` process increments the value of the `s_time_counter` signal instead of resetting its value. The consequence is a divergence between the value of the time counter of transition t_1 and the value of the `s_time_counter` signal in TCI id_{t_1}.

As demonstrated above, the `time_counter` process can not rely on the value of the `fired` ports to determine if the value of the `s_time_counter` signal must be reset or not. We proved

that there is an equivalence between the fired transitions and the value of the `fired` ports at the end of a falling edge phase. We need a way to memorize the value of `fired` ports at the moment where the equivalence hold (i.e. at the end of the falling edge phase) so that the `time_counter` process can use this information to reset the `s_time_counter` signal. To do so, we have modified the SITPN semantics and the behavior of the transition design. In the actual version of the SITPN semantics, if a transition is fired at the beginning of the rising edge phase then a reset order is sent to the transition. As a consequence, the time counter associated to this transition will be reset at the next falling edge. In the actual version of the transition design behavior, the value of the `fired` port is involved in the computation of the `s_reinit_time_counter` signal; the `s_reinit_time_counter` signal value follows the value of the reset order assigned to a given transition. Thus, as the equivalence between reset orders and the value of the `s_reinit_time_counter`signal holds at the beginning of the falling edge phase, the `time_counter` process can rely on the value of the `s_reinit_time_counter` signal to reset the value of the `s_time_counter` signal. As a consequence, the set of fired transitions is no longer involved in any the SITPN semantics rules happening during the falling edge phase. Therefore, we chose to withdraw the *Fired* set from the definition of the SITPN state record. We opted for an intentional definition of the set of fired transitions at given SITPN state (i.e., Definition **??**). After these changes, we were able to prove that there were no more divergence between the time counters and the value of the `s_time_counter` signals in the course of the execution (see Lemmas 26 and 35 about the equivalence of time counters).

2.5 Mechanized verification of the proof

The work of mechanizing the proof of the Full bisimulation theorem is an ongoing task. At the time of the writing, we have only verified thirty per cent of the proof concerning the Similar initial states lemma. However, the effort to achieve this thirty per cent of the verification amounts to three months of work. In this section, we give metrics to measure the gap between the size of the "paper" proof (see Appendix A) and the size of the computer-checked proof written in Coq. We point out some of the reasons that may explain the gap, and comment some employed techniques to reduce the size of proof scripts. As a remainder, the full code including specifications and proof scripts is available at `https://github.com/viampietro/ver-hilecop`.

Listing 2.4 presents the Coq implementation of Theorem 5 along with the sequence of tactics constituting its proof. We also declared the Behavior preservation theorem, and the Elaboration, Initialization, Simulation theorems as axioms in the `Soundness.v` file under the `soundness` folder of the Git repository.

```
1 Theorem sitpn2hvhdl_full_bisim :
2   forall τ sitpn decpr id_ent id_arch E_c θ_s d E_p mm θ_σ γ Δ,
3
4     (* sitpn is well-defined. *)
5     IsWellDefined sitpn →
6
7     (* sitpn translates into (d, γ). *)
8     sitpn_to_hvhdl sitpn decpr id_ent id_arch mm = (inl (d, γ)) →
```

```
9
10    (* Environments are similar. *)
11    SimEnv sitpn γ E_c E_p →
12
13    (* SITPN sitpn yields execution trace θ_s after τ execution cycles. *)
14    SitpnFullExec sitpn E_c γ θ_s →
15
16    (* Design d yields simulation trace θ_σ after τ simulation cycles. *)
17    hfullsim E_p τ Δ d θ_σ →
18
19    (* ** Conclusion: traces are similar. ** *)
20    SimTrace γ θ_s θ_σ.
21  Proof.
22    (* Case analysis on τ *)
23    destruct τ;
24      intros *;
25      inversion_clear 4;
26      inversion_clear 1;
27
28    (* - CASE τ = 0, GOAL γ ⊢ s0 ∼ σ0. Solved with sim_init_states lemma.
29       - CASE τ > 0, GOAL γ ⊢ (s0 :: s0 :: s :: θ) ∼ (σ0 :: σ :: σ' :: θ'').
30       Solved with [first_cycle] and [simulation] lemmas. *)
31      lazymatch goal with
32      | [ Hsimloop: simloop _ _ _ _ _ _ _ _ | _ ] ⇒
33        inversion_clear Hsimloop; constructor; eauto with hilecop
34      end.
35  Qed.
```

LISTING 2.4: Coq implementation of the Full bisimulation theorem and the mechanized version of its proof.

The proof laid out in Listing 2.4 follows the structure of the informal proof of Theorem 5. First, we perform case analysis on the structure of the τ variable through the `destruct` tactic. Then, the `intros` $*$ introduces all universally-bound variables in the proof context. Then, at Lines 25 and 26, we use a variant of the `inversion` tactic (i.e. `inversion_clear`) to unfold the definition of the SITPN full execution relation and the $\mathcal{H}$-VHDL full simulation relations. The number passed as an argument to the `inversion_clear` tactic refers to the index of the premise in the arrow-separated list of premises constituting the declaration of the theorem. At Line 31, we perform pattern matching on the proof context and on the conclusion to be proved. This permits to identify the hypothesis associated to the $\mathcal{H}$-VHDL simulation relation; we name it `Hsimloop`. This hypothesis has been introduced in the context of the proof as a side effect of the inversion `tactic` used at Line 26. Then, we introduce in the proof context new hypotheses based on the definition of the `Hsimloop` hypothesis (i.e. the definition of the $\mathcal{H}$-VHDL simulation relation) by invoking `inversion_clear` tactic on `Hsimloop`. Then, the `constructor` tactic builds sub-goals to be proved based on the definition of the full trace similarity relation (i.e.). We let the `eauto` tactic decide which lemma apply to solve the sub-goals generated by the

constructor tactics. We give a hint to the eauto tactic so that it looks in the user-defined hilecop database of theorems and lemmas to solve the sub-goals. The hilecop database contains the Coq implementation of all the theorems and lemmas used to prove the Full bisimulation theorem.

Robustness to change

The proof laid out in Listing 2.4 is representative of our strategy to keep our mechanized proofs robust to change. The robustness criterion is important for multiple reasons. First, in the proceeding of the proof, we can always realize that some case is missing in the expression of the transformation function or discover that the semantics of the SITPNs or the $\mathcal{H}$-VHDL language is incomplete or incorrect. Therefore, we want to structure our proofs in a way that will lower the impact of correcting the transformation function or completing the semantics. Second, we know that the SITPN structure and the $\mathcal{H}$-VHDL code of the place and transition designs will be evolving in the future. Therefore, we want to be able to adapt our proofs with a minimum effort. To reach robustness to change, we follow the indications laid out in [8]. Mainly, we make an important use of the pattern matching constructs, such as lazymatch or match, to seek hypotheses in the current proof context. Also, we build hint databases and rely as much as possible on the use of the auto and eauto to solve the conclusions.

Automation

To shorten the size of proofs, we develop user-defined tactics using the Coq Ltac language. The tactic that most contributed to the reduction of the size of the proof scripts is the minv tactic (see StateAndErrorMonadTactics.v under the common folder). The minv tactic automate the proof of certain lemmas regarding the properties of the HILECOP transformation function in the context of the state-and-error monad. Our Coq implementation of the HILECOP transformation function implements the state-and-error monad. This monad simulates imperative language traits into functional languages. All functions involve in the HILECOP transformation function carry a compile-time state, defined as the Coq type CompileTimeState. Each function either return a value, modify the compile-time state or do both. To give an example of the use of the minv tactic, Listing 2.5 shows the implementation of the generate_place_comp_inst function involved in HILECOP transformation function. The generate_place_comp_inst function generates a $\mathcal{H}$-VHDL PCI statement from a place p passed as a parameter. As a side effect, the generate_place_comp_inst function adds the PCI statement to the behavior of the top-level design currently built in the compile-time state.

```
1 Definition generate_place_comp_inst (p : P sitpn) : CompileTimeState unit :=
2
3   do id        ← get_nextid;
4   do _         ← bind_place p id;
5   do pcomp     ← get_pcomp p;
6   do pcomp_inst ← HComponent_to_comp_inst id place_entid pcomp;
7   add_cs pcomp_inst.
```

LISTING 2.5: Coq implementation of the `generate_place_comp_inst` function; the function takes an SITPN place p as a parameter, and modifies the compile-time state without returning a value (i.e. the function return type is `unit`)

In its definition body, function `generate_place_comp_inst` sequentially calls to functions that sometimes modify the compile-time state (e.g. the `bind_place` function adds a binding between p and id in the generated γ binder, i.e. $\gamma(p) = id$ after the call to `bind_place`), or sometimes simply return a value without modifying the state (e.g. `get_pcomp` returns an intermediate structure representing the place component instance associated to place p in the compile-time state). During the mechanization of the proof, we often need to prove that some properties hold between the input compile-time state and the output compile-time after the call to a certain function. For example, after calling the `generate_place_comp_inst` function on a given place p and for a given input state s, let us say that a new compile-time state s' is returned. We want to show that the part of the γ binder pertaining to the binding of transitions to TCI identifiers has not changed between state s and state s'[2]. To perform the proof, we need to show that each function call composing the sequence of the `generate_place_comp_inst` function returns a compile-time state verifying the wanted property. Proving simple property like verifying that part of the compile-time states are equal through the multiple invocation of functions is highly automatable. We adapt the tactic `monadInv` defined in the CompCert project [15] to automate proof for such properties. The result is the tactic `minv` massively used in the proofs pertaining to state invariants[3].

Gap between informal and formal proof

There is a huge gap of size between the informal proof of the Full bisimulation theorem given in this Chapter and in Appendix A and the machine-checked formal proof. Right now, the Coq proof wins the size competition. The most significant distance between the size of the informal and the formal proof comes from the two following points: the statement of the combinational equations defining the value of $\mathcal{H}$-VHDL signals and the statement of properties about the HILECOP transformation function. Stating that a combinational equation holds for a given signal in the context of an informal proof is a one-line sentence. The same goes when invoking the properties of the PCIs and TCIs populating the top-level design behavior based on the definition of the transformation function. However, proving these statements represents a tremendous mechanization effort within the Coq proof assistant. To give an example, we begin the proof of Lemma 6 by taking a place p and a PCI identifier id_p linked through the γ binder returned by the transformation function. Then, we state the existence of a PCI statement, identified by id_p and with an associated generic map, input port map and output port map, in the behavior of the top-level design returned by the transformation function. To do so, we use the following the sentence:

[2]Remember that the γ binder is part of the compile-time state record type.

[3]State invariance lemmas are to be found in the `GenerateInfosInvs.v`, `GenerateArchitectureIns.v`, `GeneratePortsInvs.v` and `GenerateHVhdlInvs.v` under the `sitpn2hvhdl` folder of the Git repository.

"Let us take a $p \in P$ and an $id_p \in Comps(\Delta)$ such that $\gamma(p) = id_p$. By construction, there exist gm_p, ipm_p, opm_p s.t. $\texttt{comp}(id_p, "place", gm_p, ipm_p, opm_p) \in d.cs$."

The expression "by construction" is a shorthand for "knowing how the target $\mathcal{H}$-VHDL design is constructed by the transformation function", or "based on the definition of the transformation function". In Coq, proving the lemma that states the existence of a PCI for a given place p amounts to 1500 lines of proof script. The lemmas regarding properties of PCI and TCI statements deduced from the transformation function tend to have complicated proofs. We believe that the implementation of the HILECOP transformation function could be more straightforward in order to simplify this kind of proof. By straightforward, we mean that the number of steps separating a given place or a given transition from the generation of their corresponding PCI or TCI could be diminished, maybe at the cost of time performance. Right now, ease of proof is more important than time performance, considering that our goal is to prove the semantic preservation theorem in a reasonable amount of time. Still, the major complexity of the transformation function, i.e. what makes the proofs so hard, lies in the generation of the interconnections between PCIs and TCIs. Some engineering effort could be spent to simplify this particular of the transformation.

Also, we spent a lot of time proving some uninteresting, however necessary, properties about the $\mathcal{H}$-VHDL design states and the $\mathcal{H}$-VHDL simulation relations. For instance, we proved a lot of lemmas pertaining the preservation of identifiers through the simulation phases (e.g if a signal id is present in a design state at the beginning of a stabilization phase, then it is still present at the end of the phase). We also proved a lot of uninteresting properties about the $\mathcal{H}$-VHDL elaborated designs and the $\mathcal{H}$-VHDL elaboration relation. For instance, properties on the uniqueness of identifiers in design states, in elaborated designs...We believe that a more systematic use of dependent types, especially to implement the $\mathcal{H}$-VHDL design state and the elaborated design structure, could prevent us from proving this kind of lemmas.

2.6 Conclusion

In this chapter, the aim was to present the behavior preservation theorem stating that the HILECOP transformation is semantic preserving along with its informal proof. By presenting the work of the literature pertaining to the verification of *transformation functions* through the proof of behavior preservation theorems, we wanted to convince the reader that the expression of our semantic preservation theorem is "correct", i.e. it follows a common expression pattern. We saw that the expression of semantic preservation theorems is quite similar in its form even when considered transformations are not of the same nature (i.e. GPL compilers, HDL compilers and model transformations). Our semantic preservation theorem takes the form of a state similarity checking between the states composing the execution traces of our source model and our target program. At each point of the execution (i.e. at each clock signal event), the state of the input model and the state of the output representation must be similar to ensure the behavior preservation property. This definition of the behavior preservation property is particular to reactive systems, i.e. we are dealing with systems that are executing over time, and that are synchronized with a clock signal. Naturally, the behavior preservation theorem must ensure that the behaviors are similar, independently of the number of execution cycles performed.

Hopefully, leveraging the inductive reasoning, proving such a thing comes down to proving that behaviors are preserved through a clock cycle.

The study of the literature showed that the state comparison relation, i.e. the relation that describe how things are compared between the source and the target representation, is a significant element in the expression of the behavior preservation theorem. Especially, in our case, the state structure of the source and target representations are quite different. Indeed, we are dealing with an abstract set of data in the SITPN world, while in the $\mathcal{H}$-VHDL representation all is converted into signal values and internal states of component instances. Thus, relating these two kind of states is not straightforward, and we thoroughly presented our state similarity relation in Section 2.2.

In this chapter, we wanted to stress another point pertaining no more to the "how" but to the "when" the states of the input and output representations must be compared in the course of the execution. Here, we are dealing with to kind of models that are synchronously executed. However, the synchronous execution of an SITPN stays at a level that is quite abstract compared to the concrete execution of a synchronous hardware system. Indeed, the execution of a synchronous hardware system is related to the rules of the combinational and the synchronous logic, while it is not the case at the SITPN level. Thus, a $\mathcal{H}$-VHDL design goes through a lot more different states in the proceeding of a clock cycle compared to its corresponding SITPN. Figure 2.3 illustrates when the state comparison must be performed in the course of a clock cycle.

While presenting the proof of Theorem 1, we used certain theorems declared as axioms (Theorems 2, 3 and 4). These theorems express the fact that we can always derive a simulation trace from the execution of a $\mathcal{H}$-VHDL design resulting of a succesful HILECOP transformation. It means that the execution of a $\mathcal{H}$-VHDL design resulting from the HILECOP transformation never results into an error at some point of the simulation. We chose not to represent errors in the $\mathcal{H}$-VHDL semantics due to the fact that the concept of error is nonexistent in the SITPN semantics. However, we argue that proving a theorem stating the existence of a simulation trace, independently of the number of simulation cycles considered, is a way to rectify the lack of error representation in our semantics. By presenting Theorems 2, 3 and 4 as axioms, we chose to prove the theorem of semantic preservation in the case where a simulation trace has been produced for the generated $\mathcal{H}$-VHDL design. This is the setting of Theorem 5 for which the full proof is detailed in this chapter and in Appendix A. However, we are not giving up on the proof of Theorems 2, 3 and 4. Indeed, proving a theorem stating the similarity of execution traces is useless if the execution a generated $\mathcal{H}$-VHDL design always fails at some point while the execution of the corresponding SITPN goes on. However, we are confident in the fact that if the execution of a generated $\mathcal{H}$-VHDL design fails, then it can only reflect a divergence in relation to the behavior of the input SITPN. Thus, proving that the execution traces are similar contributes to the proof that we can always derive an execution trace for a generated $\mathcal{H}$-VHDL design.

The informal "paper" proof of Theorem 5 given in this chapter and Appendix A is long; about a hundred pages. However, as we explained in Section 2.4, the strategy used in the overall proof is pretty much the same. To prove that the behavior of an SITPN and its corresponding $\mathcal{H}$-VHDL design is preserved through an execution cycle, we must reason on the execution relations ruling both worlds. But first, to relate the execution of our input and output

representations, we must structurally relate the SITPN to the translated $\mathcal{H}$-VHDL design. In the proceeding of the proof, we will first reason on the structure of the input SITPN; based on the structure of the SITPN and by property of the HILECOP transformation, we can determine the structure of the top-level $\mathcal{H}$-VHDL design. Once we know the structure of the SITPN and the $\mathcal{H}$-VHDL design, we can unfold their execution rules to prove that their behavior are the same; i.e. at the end of a computational step, states are similar.

The mechanization of the proof of Theorem 5 is at its very beginning in terms of completion. However, we have already spent three months on it. Thus, the mechanization is a very slow process. We explain the hardness of the mechanization task by pointing out the two points where the distance between informal and formal proof is most important. The first point corresponds to the statement of the construction of the $\mathcal{H}$-VHDL design based on the structure of the SITPN and the HILECOP transformation function. Reasoning on the transformation function is not an easy task as the transformation itself is not as straightforward as the transformation from a source program of a GPL to a target program of another GPL. In Section 2.5, we pointed out the distance between a property of the transformation function expressed in one sentence in the informal proof and the thousands of lines that it represents in the formal proof. The second point digging the distance between the informal and the formal proof comes from the establishment of the synchronous and combinational equations that are verified by the internal signals of the PCIs and TCIS. This also results in one sentence statement in the informal proof while representing thousands of lines of code in the formal proof. The De Bruijn factor [23], that permits to measure the distance in terms of number of characters between an informal proof and its machine-checked version (i.e. the formal program), is tremendously high in our case when considering these intermediary results.

Appendix A

Semantic preservation proof

Constants and signals reference			
Full name	*Alias*	*Category*	*Type*
input_arcs_number	ian	generic constant (T)	$\mathbb{N}$
transition_type	tt	generic constant (T)	{not_temp, temp_a_b, temp_a_a, temp_a_inf}
conditions_number	cn	generic constant (T)	$\mathbb{N}$
maximal_time_counter	mtc	generic constant (T)	$\mathbb{N}$
time_A_value	A	input port (T)	$\mathbb{N}$
time_B_value	B	input port (T)	$\mathbb{N}$
input_conditions	ic	input port (T)	array of $\mathbb{B}$
reinit_time	rt	input port (T)	array of $\mathbb{B}$
input_arcs_valid	iav	input port (T)	array of $\mathbb{B}$
priority_authorizations	pauths	input port (T)	array of $\mathbb{B}$
fired	f	output port (T)	$\mathbb{B}$
s_condition_combination	scc	internal signal (T)	$\mathbb{B}$
s_reinit_time_counter	srtc	internal signal (T)	$\mathbb{B}$
s_priority_combination	spc	internal signal (T)	$\mathbb{B}$
s_firable	sfa	internal signal (T)	$\mathbb{B}$
s_enabled	se	internal signal (T)	$\mathbb{B}$
s_time_counter	stc	internal signal (T)	$\mathbb{N}$
s_firing_condition	sfc	internal signal (T)	$\mathbb{B}$
input_arcs_number	ian	generic constant (P)	$\mathbb{N}$
output_arcs_number	oan	generic constant (P)	$\mathbb{N}$
maximal_marking	mm	generic constant (P)	$\mathbb{N}$
initial_marking	im	input port (P)	$\mathbb{N}$
output_arcs_types	oat	input port (P)	array of {basic, test, inhib}
output_arcs_weights	oaw	input port (P)	array of $\mathbb{N}$
output_transitions_fired	otf	input port (P)	array of $\mathbb{B}$
input_arcs_weights	iaw	input port (P)	array of $\mathbb{N}$
input_transitions_fired	itf	input port (P)	array of $\mathbb{B}$
output_transitions_fired	otf	output port (P)	array of $\mathbb{B}$

`reinit_transitions_time`	`rtt`	output port (P)	array of $\mathbb{B}$
`priority_authorizations`	`pauths`	output port (P)	array of $\mathbb{B}$
`s_marking`	`sm`	internal signal (P)	$\mathbb{N}$
`s_output_token_sum`	`sots`	internal signal (P)	$\mathbb{N}$
`s_input_token_sum`	`sits`	internal signal (P)	$\mathbb{N}$

TABLE A.1: Constants and signals reference for the $\mathcal{H}$-VHDL transition and place designs. In the *Category* column, T (resp. P) indicates a generic constant, input port, output port or internal signal defined in the `transition` (resp. `place`) design.

A.1 Initial States

Definition 14 (Initial state hypotheses). *Given an* $sitpn \in SITPN, d \in design, \gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}}), \sigma_e, \sigma_0 \in \Sigma(\Delta)$, *assume that:*

- *SITPN* $sitpn$ *translates into design* d: $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$
- Δ *is the elaborated version of* d, σ_e *is the default state of* Δ, *i.e, state of* Δ *where all signals have their default value:*

  $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e)$
- σ_0 *is the initial state of* Δ: $\Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0$

Lemma 5 (Similar initial states). *For all* $sitpn \in SITPN, d \in design, \gamma \in WM(sitpn, d), \Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}}), \sigma_e, \sigma_0 \in \Sigma(\Delta)$ *that verify the hypotheses of Definition 14, then* $\gamma \vdash s_0 \sim \sigma_0$.

Proof. By definition of the General state similarity relation, there are 6 points to prove.

1. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s_0.M(p) = \sigma_0(id_p)(\text{"s\_marking"})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $(upper(I_s(t)) = \infty \wedge s_0.I(t) \leq lower(I_s(t)) \Rightarrow s_0.I(t) = \sigma_0(id_t)(\text{"s\_time\_counter"}))$
   $\wedge (upper(I_s(t)) = \infty \wedge s_0.I(t) > lower(I_s(t)) \Rightarrow \sigma_0(id_t)(\text{"s\_time\_counter"}) = lower(I_s(t)))$
   $\wedge (upper(I_s(t)) \neq \infty \wedge s_0.I(t) > upper(I_s(t)) \Rightarrow \sigma_0(id_t)(\text{"s\_time\_counter"}) = upper(I_s(t)))$
   $\wedge (upper(I_s(t)) \neq \infty \wedge s_0.I(t) \leq upper(I_s(t)) \Rightarrow s_0.I(t) = \sigma_0(id_t)(\text{"s\_time\_counter"}))$.
3. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, s_0.reset_t(t) = \sigma_0(id_t)(\text{"s\_reinit\_time\_counter"})$.
4. $\forall c \in \mathcal{C}, id_c \in Ins(\Delta)\ s.t.\ \gamma(c) = id_c,\ s_0.cond(c) = \sigma_0(id_c)$.
5. $\forall a \in \mathcal{A}, id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a,\ s_0.ex(a) = \sigma_0(id_a)$.
6. $\forall f \in \mathcal{F}, id_f \in Outs(\Delta)\ s.t.\ \gamma(f) = id_f,\ s_0.ex(f) = \sigma_0(id_f)$.

- Apply the Initial states equal marking lemma to solve 1.

- Apply the Initial states equal time counters lemma to solve 2.
- Apply the Initial states equal reset orders lemma to solve 3.
- Apply the Initial states equal condition values lemma to solve 4.
- Apply the Initial states equal action executions lemma to solve 5.
- Apply the Initial states equal function executions lemma to solve 6.

□

A.1.1 Initial states and marking

Lemma 6 (Initial states equal marking). *For all* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}}), \sigma_e, \sigma_0 \in \Sigma(\Delta)$ *that verify the hypotheses of Definition 14, then* $\forall p \in P, id_p \in Comps(\Delta)$ *s.t.* $\gamma(p) = id_p$, $s_0.M(p) = \sigma_0(id_p)(\text{"}s\_marking\text{"})$.

Proof. Given a $p \in P$ and an $id_p \in Comps(\Delta)$ s.t. $\gamma(p) = id_p$, let us show that $\boxed{s_0.M(p) = \sigma_0(id_p)(\text{"}s\_marking\text{"}).}$
By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.
By property of the $\mathcal{H}$-VHDL initialization relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `marking` process defined in the place design architecture, we can deduce $\sigma_0(id_p)(\text{"}s\_marking\text{"}) = \sigma_0(id_p)(\text{"}initial\_marking\text{"})$.
Rewriting $\sigma_0(id_p)(\text{"}sm\text{"})$ as $\sigma_0(id_p)(\text{"}initial\_marking\text{"})$, $\boxed{\sigma_0(id_p)(\text{"}initial\_marking\text{"}) = s_0.M(p).}$
By construction, $<\texttt{initial\_marking} \Rightarrow M_0(p)> \in ipm_p$.
By property of the $\mathcal{H}$-VHDL initialization relation, and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, then $\sigma_0(id_p)(\text{"}initial\_marking\text{"}) = M_0(p)$. Rewriting $\sigma_0(id_p)(\text{"}initial\_marking\text{"})$ as $M_0(p)$ in the current goal: $\boxed{M_0(p) = s_0.M(p).}$
By definition of s_0, we can rewrite $s_0.M(p)$ as $M_0(p)$ in the current goal, tautology.

□

Lemma 7 (Null input token sum at initial state). *For all* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}}), \sigma_e, \sigma_0 \in \Sigma(\Delta)$ *that verify the hypotheses of Definition 14, then* $\forall p \in P, id_p \in Comps(\Delta)$ *s.t.* $\gamma(p) = id_p$, $\sigma_0(id_p)(\text{"}s\_input\_token\_sum\text{"}) = 0$.

Proof. Given a p and an id_p s.t. $\gamma(p) = id_p$, let us show that $\boxed{\sigma_0(id_p)(\text{"}s\_input\_token\_sum\text{"}) = 0.}$
By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.
By property of the initialization relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `input_tokens_sum` process defined in the place design architecture, we can deduce:

$$\sigma_0(id_p)(\text{"}sits\text{"}) = \sum_{i=0}^{\Delta(id_p)(\text{"}ian\text{"})-1} \begin{cases} \sigma_0(id_p)(\text{"}iaw\text{"})[i] \texttt{ if } \sigma_0(id_p)(\text{"}itf\text{"})[i] \\ 0 \textit{ otherwise} \end{cases} \tag{A.1}$$

Rewriting the goal with Equation (A.1):

$$\sum_{i=0}^{\Delta(id_p)("ian")-1} \begin{cases} \sigma_0(id_p)("iaw")[i] \texttt{ if } \sigma_0(id_p)("itf")[i] \\ 0 \textit{ otherwise} \end{cases} = 0.$$

Let us perform case analysis on $input(p)$; there are two cases:

1. $input(p) = \emptyset$:

   By construction, $<\texttt{input\_arcs\_number} \Rightarrow 1> \in gm_p$, $<\texttt{input\_transitions\_fired(0)} \Rightarrow \texttt{true}> \in ipm_p$, and $<\texttt{input\_arcs\_weights(0)} \Rightarrow 0> \in ipm_p$.

   By property of the elaboration relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and $<\texttt{input\_arcs\_number} \Rightarrow 1> \in gm_p$, we can deduce $\Delta(id_p)("ian") = 1$.

   By property of the initialization relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, $<\texttt{input\_transitions\_fired(0)} \Rightarrow \texttt{true}> \in ipm_p$ and $<\texttt{input\_arcs\_weights(0)} \Rightarrow 0> \in ipm_p$, we can deduce $\sigma_0(id_p)("itf")[0] = \texttt{true}$ and $\sigma_0(id_p)("iaw")[0] = 0$.

   Rewriting the goal with $\Delta(id_p)("ian") = 1$, $\sigma_0(id_p)("itf")[0] = \texttt{true}$, $\sigma_0(id_p)("iaw")[0] = 0$ and simplifying the goal, tautology.

2. $input(p) \neq \emptyset$:

   By construction, $<\texttt{input\_arcs\_number} \Rightarrow |input(p)|> \in gm_p$, and by property of the elaboration relation, and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\Delta(id_p)("ian") = |input(p)|$.

   Let us reason by induction on the sum term of the goal.

   - **BASE CASE**: The sum term equals 0, then tautology.
   - **INDUCTION CASE**:

     $$\sum_{i=1}^{\Delta(id_p)("ian")-1} \begin{cases} \sigma_0(id_p)("iaw")[i] \texttt{ if } \sigma_0(id_p)("itf")[i] \\ 0 \textit{ otherwise} \end{cases} = 0$$

     $$\begin{cases} \sigma_0(id_p)("iaw")[0] \texttt{ if } \sigma_0(id_p)("itf")[0] \\ 0 \textit{ otherwise} \end{cases} + \sum_{i=1}^{\Delta(id_p)("ian")-1} \begin{cases} \sigma_0(id_p)("iaw")[i] \texttt{ if } \sigma_0(id_p)("itf")[i] \\ 0 \textit{ otherwise} \end{cases}$$

     Using the induction hypothesis to rewrite the goal:

     $$\begin{cases} \sigma_0(id_p)("iaw")[0] \texttt{ if } \sigma_0(id_p)("itf")[0] \\ 0 \textit{ otherwise} \end{cases} = 0$$

     Since $input(p) \neq \emptyset$, by construction, there exist an $id_t \in Comps(\Delta), gm_t, ipm_t, opm_t$ s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, $id_{ft} \in Sigs(\Delta)$ s.t. $<\texttt{fired} \Rightarrow id_{ft}> \in opm_t$ and $<\texttt{input\_transitions\_fired(0)} \Rightarrow \texttt{id}_{\texttt{ft}}> \in ipm_p$.

By property of the initialization relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, $<\texttt{fired} \Rightarrow id_{ft}> \in opm_t$ and $<\texttt{input\_transitions\_fired(0)} \Rightarrow \texttt{id}_{\texttt{ft}}> \in ipm_p$, we can deduce $\sigma_0(id_p)(''itf'')[0] = \sigma_0(id_t)(''fired'')$.
Rewriting the goal with $\sigma_0(id_p)(''itf'')[0] = \sigma_0(id_t)(''fired'')$:

$$\boxed{\begin{cases} \sigma_0(id_p)(''iaw'')[0] \texttt{ if } \sigma_0(id_t)(''fired'') \\ 0 \textit{ otherwise} \end{cases} = 0}$$

Appealing to Lemma 14, we can deduce $\sigma_0(id_t)(''fired'') = \texttt{false}$.
Rewriting the goal with $\sigma_0(id_t)(''fired'') = \texttt{false}$, and simplifying the goal, tautology.

□

Lemma 8 (Null output token sum at initial state). *For all $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$, $\sigma_e, \sigma_0 \in \Sigma(\Delta)$ that verify the hypotheses of Definition 14, then $\forall p \in P, id_p \in Comps(\Delta)$ s.t. $\gamma(p) = id_p$, $\sigma_0(id_p)(''s\_output\_token\_sum'') = 0$.*

Proof. The proof is similar to the proof of Lemma 7. □

A.1.2 Initial states and time counters

Lemma 9 (Initial states equal time counters). *For all $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$, $\sigma_e, \sigma_0 \in \Sigma(\Delta)$ that verify the hypotheses of Definition 14, then $\forall t \in T_i, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$,*
$upper(I_s(t)) = \infty \wedge s_0.I(t) \leq lower(I_s(t)) \Rightarrow s_0.I(t) = \sigma_0(id_t)(''s\_time\_counter'') \wedge$
$upper(I_s(t)) = \infty \wedge s_0.I(t) > lower(I_s(t)) \Rightarrow \sigma_0(id_t)(''s\_time\_counter'') = lower(I_s(t)) \wedge$
$upper(I_s(t)) \neq \infty \wedge s_0.I(t) > upper(I_s(t)) \Rightarrow \sigma_0(id_t)(''s\_time\_counter'') = upper(I_s(t)) \wedge$
$upper(I_s(t)) \neq \infty \wedge s_0.I(t) \leq upper(I_s(t)) \Rightarrow s_0.I(t) = \sigma_0(id_t)(''s\_time\_counter'')$.

Proof. Given a $t \in T_i$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, let us show that:

1. $\boxed{upper(I_s(t)) = \infty \wedge s_0.I(t) \leq lower(I_s(t)) \Rightarrow s_0.I(t) = \sigma_0(id_t)(''s\_time\_counter'')}$
2. $\boxed{upper(I_s(t)) = \infty \wedge s_0.I(t) > lower(I_s(t)) \Rightarrow \sigma_0(id_t)(''s\_time\_counter'') = lower(I_s(t))}$
3. $\boxed{upper(I_s(t)) \neq \infty \wedge s_0.I(t) > upper(I_s(t)) \Rightarrow \sigma_0(id_t)(''s\_time\_counter'') = upper(I_s(t))}$
4. $\boxed{upper(I_s(t)) \neq \infty \wedge s_0.I(t) \leq upper(I_s(t)) \Rightarrow s_0.I(t) = \sigma_0(id_t)(''s\_time\_counter'')}$

By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.
Then, let us show the 4 previous points.

1. Assuming that $upper(I_s(t)) = \infty \wedge s_0.I(t) \leq lower(I_s(t))$, then let us show $\boxed{s_0.I(t) = \sigma_0(id_t)(''s\_time\_counter'').}$

Rewriting $s_0.I(t)$ as 0, by definition of s_0, $\boxed{\sigma_0(id_t)(\text{"s\_time\_counter"}) = 0.}$

By property of the $\mathcal{H}$-VHDL initialization relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `time_counter` process defined in the transition design architecture, we can deduce $\sigma_0(id_t)(\text{"s\_time\_counter"}) = 0.$

2. Assuming that $upper(I_s(t)) = \infty$ and $s_0.I(t) > lower(I_s(t))$, let us show $\boxed{\sigma_0(id_t)(\text{"s\_time\_counter"}) = lower(I_s(t))}$.

   By definition, $lower(I_s(t)) \in \mathbb{N}^*$ and $s_0.I(t) = 0$. Then, $lower(I_s(t)) < 0$ is a contradiction.

3. Assuming that $upper(I_s(t)) \neq \infty$ and $s_0.I(t) > upper(I_s(t))$, let us show $\boxed{\sigma_0(id_t)(\text{"s\_time\_counter"}) = upper(I_s(t))}$.

   By definition, $upper(I_s(t)) \in \mathbb{N}^*$ and $s_0.I(t) = 0$. Then, $upper(I_s(t)) < 0$ is a contradiction.

4. Assuming that $upper(I_s(t)) \neq \infty$ and $s_0.I(t) \leq upper(I_s(t))$, let us show $\boxed{s_0.I(t) = \sigma_0(id_t)(\text{"s\_time\_counter"})}$.

   Rewriting $s_0.I(t)$ as 0, by definition of s_0, $\boxed{\sigma_0(id_t)(\text{"s\_time\_counter"}) = 0.}$

   By property of the $\mathcal{H}$-VHDL initialization relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `time_counter` process defined in the transition design architecture, we can deduce $\sigma_0(id_t)(\text{"s\_time\_counter"}) = 0.$

□

A.1.3 Initial states and reset orders

Lemma 10 (Initial states equal reset orders). *For all $sitpn \in SITPN, d \in design, \gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_\mathcal{H}), \sigma_e, \sigma_0 \in \Sigma(\Delta)$ that verify the hypotheses of Definition 14, then $\forall t \in T_i, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $s_0.reset_t(t) = \sigma_0(id_t)(\text{"s\_reinit\_time\_counter"})$.*

Proof. Given a $t \in T_i$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, let us show that $\boxed{s_0.reset_t(t) = \sigma_0(id_t)(\text{"s\_reinit\_time\_counter"}).}$

Rewriting $s_0.reset_t(t)$ as `false`, by definition of s_0, $\boxed{\sigma_0(id_t)(\text{"s\_reinit\_time\_counter"}) = \texttt{false}.}$

By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.

By property of the $\mathcal{H}$-VHDL initialization relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `reinit_time_counter_evaluation` process defined in the transition design architecture

we can deduce $\sigma_0(id_t)(\text{"s\_reinit\_time\_counter"}) = \prod_{i=0}^{\Delta(id_t)(\text{"ian"})-1} \sigma_0(id_t)(\text{"rt"})[i]$.

Rewriting $\sigma_0(id_t)(''s\_reinit\_time\_counter'')$ as $\prod_{i=0}^{\Delta(id_t)(''ian'')-1} \sigma_0(id_t)(''rt'')[i]$,

$\boxed{\prod_{i=0}^{\Delta(id_t)(''ian'')-1} \sigma_0(id_t)(''rt'')[i] = \texttt{false}.}$

For all $t \in T$ (resp. $p \in P$), let $input(t)$ (resp. $input(p)$) be the set of input places of t (resp. input transitions of p), and let $output(t)$ (resp. $output(p)$) be the set of output places of t (resp. output transitions of p).
Let us perform case analysis on $input(t)$; there are 2 cases:

- **CASE** $input(t) = \emptyset$.

  By construction, $<\texttt{input\_arcs\_number} \Rightarrow 1> \in gm_t$, and by property of the elaboration relation, and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, we can deduce $\Delta(id_t)(''ian'') = 1$.

  By construction, $< \texttt{reinit\_time(0)} \Rightarrow \texttt{false} > \in ipm_t$, and by property of the initialization relation and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, we can deduce $\sigma_0(id_t)(''rt'')[0] = \texttt{false}$.

  Rewriting $\Delta(id_t)(''ian'')$ as 1 and $\sigma_0(id_t)(''rt'')[0]$ as $\texttt{false}$, tautology.

- **CASE** $input(t) \neq \emptyset$.

  To prove the current goal, we can equivalently prove that
  $\boxed{\exists i \in [0, \Delta(id_t)(''ian'') - 1] \; s.t. \; \sigma_0(id_t)(''rt'')[i] = \texttt{false}.}$

  Since $input(t) \neq \emptyset$, $\exists p \; s.t. \; p \in input(t)$. Let us take such a $p \in input(t)$.

  By construction, for all $p \in P$, there exist $id_p \; s.t. \; \gamma(p) = id_p$.

  By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.

  By construction, there exist $i \in [0, |input(t)| - 1]$, $j \in [0, |output(p)| - 1]$, $id_{ji} \in Sigs(\Delta)$ s.t. $<\texttt{reinit\_transitions\_time(j)} \Rightarrow id_{ji}> \in opm_p$ and $<\texttt{reinit\_time(i)} \Rightarrow id_{ji}> \in ipm_t$. Let us take such a i, j and id_{ji}.

  By construction and $input(t) \neq \emptyset$, $<\texttt{input\_arcs\_number} \Rightarrow |input(t)|> \in gm_t$.

  By property of the $\mathcal{H}$-VHDL elaboration relation and $<\texttt{input\_arcs\_number} \Rightarrow |input(t)|> \in gm_t$, we can deduce $\Delta(id_t)(''ian'') = |input(t)|$.

  Since $\Delta(id_t)(''ian'') = |input(t)|$ and we have an $i \in [0, |input(t)| - 1]$, then, we have an $i \in [0, \Delta(id_t)(''ian'') - 1]$. Let us take that i to prove the goal.

  Then, we must show $\boxed{\sigma_0(id_t)(''rt'')[i] = \texttt{false}.}$

  By property of the $\mathcal{H}$-VHDL initialization relation and $<\texttt{reinit\_time(i)} \Rightarrow id_{ji}> \in ipm_t$, we can deduce $\sigma_0(id_t)(''rt'')[i] = \sigma_0(''id_{ji}'')$.

  Rewriting $\sigma_0(id_t)(''rt'')[i]$ as $\sigma_0(''id_{ji}'')$, $\boxed{\sigma_0(''id_{ji}'') = \texttt{false}.}$

  By property of the $\mathcal{H}$-VHDL initialization relation and $< \texttt{reinit\_transitions\_time(j)} \Rightarrow id_{ji} > \in opm_p$, we can deduce $\sigma_0(''id_{ji}'') = \sigma_0(id_p)(''rtt'')[j]$.

Rewriting $\sigma_0("id_{ji}")$ as $\sigma_0(id_p)("rtt")[j]$, $\boxed{\sigma_p^0("rtt")[j] = \texttt{false}.}$

Since $t \in output(p)$, then we know that $output(p) \neq \emptyset$.

Then, by construction, $<\texttt{output\_arcs\_number} \Rightarrow |output(p)|> \in gm_p$.

By property of the elaboration relation and $<\texttt{output\_arcs\_number} \Rightarrow |output(p)|> \in gm_p$, we can deduce that $\Delta(id_p)("oan") = |output(p)|$.

Since $\Delta(id_p)("oan") = |output(p)|$ and $j \in [0, |output(p)| - 1]$, then $j \in [0, \Delta(id_p)("oan") - 1]$.

By property of the $\mathcal{H}$-VHDL initialization relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, through the examination of the `reinit_transitions_time_evaluation` process defined in the place design architecture, and since $j \in [0, \Delta(id_p)("oan") - 1]$, $\sigma_0(id_p)("rtt")[j] = \texttt{false}.$

□

A.1.4 Initial states and condition values

Lemma 11 (Initial states equal condition values)**.** *For all* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_\mathcal{H}), \sigma_e, \sigma_0 \in \Sigma(\Delta)$ *that verify the hypotheses of Definition 14, then* $\forall c \in \mathcal{C}, id_c \in Ins(\Delta)$ *s.t.* $\gamma(c) = id_c$, $s_0.cond(c) = \sigma_0(id_c)$.

Proof. Given a $c \in \mathcal{C}$ and an $id_c \in Ins(\Delta)$ *s.t.* $\gamma(c) = id_c$, let us show that $\boxed{s_0.cond(c) = \sigma_0(id_c).}$

Rewriting $s_0.cond(c)$ as `false`, by definition of s_0, $\boxed{\sigma_0(id_c) = \texttt{false}.}$
By construction, id_c is an input port identifier of Boolean type in the $\mathcal{H}$-VHDL design d, and thus, by property of the $\mathcal{H}$-VHDL elaboration relation, we can deduce $\sigma_e(id_c) = \texttt{false}$.
By property of the $\mathcal{H}$-VHDL initialization relation and $id_c \in Ins(\Delta)$, we can deduce $\sigma_e(id_c) = \sigma_0(id_c)$.
Rewriting $\sigma_0(id_c)$ as $\sigma_e(id_c)$ and $\sigma_e(id_c)$ as `false`, tautology.

□

A.1.5 Initial states and action executions

Lemma 12 (Initial states equal action executions)**.** *For all* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_\mathcal{H}), \sigma_e, \sigma_0 \in \Sigma(\Delta)$ *that verify the hypotheses of Definition 14, then* $\forall a \in \mathcal{A}, id_a \in Outs(\Delta)$ *s.t.* $\gamma(a) = id_a$, $s_0.ex(a) = \sigma_0(id_a)$.

Proof. Given a $a \in \mathcal{A}$ and an $id_a \in Outs(\Delta)$ *s.t.* $\gamma(a) = id_a$, let us show that $\boxed{s_0.ex(a) = \sigma_0(id_a).}$

Rewriting $s_0.ex(a)$ as `false`, by definition of s_0, $\boxed{\sigma_0(id_a) = \texttt{false}.}$
By construction, id_a is an output port identifier of Boolean type in the $\mathcal{H}$-VHDL design d. Moreover, we know that the output port identifier id_a is assigned to `false` in the generated `action` process during the initialization phase (i.e. the assignment is a part of a *reset* block). Thus, we can deduce that $\sigma_0(id_a) = \texttt{false}$.

Rewriting $\sigma_0(id_a)$ as `false`, tautology.

□

A.1.6 Initial states and function executions

Lemma 13 (Initial states equal function executions). *For all* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}}), \sigma_e, \sigma_0 \in \Sigma(\Delta)$ *that verify the hypotheses of Definition 14, then* $\forall f \in \mathcal{F}, id_f \in Outs(\Delta)$ *s.t.* $\gamma(f) = id_f$, $s_0.ex(f) = \sigma_0(id_f)$.

Proof. Given a $f \in \mathcal{F}$ and an $id_f \in Outs(\Delta)$ *s.t.* $\gamma(f) = id_f$, let us show that $\boxed{s_0.ex(f) = \sigma_0(id_f).}$

Rewriting $s_0.ex(f)$ as `false`, by definition of s_0, $\boxed{\sigma_0(id_f) = \texttt{false}.}$
By construction, id_f is an output port identifier of Boolean type in the $\mathcal{H}$-VHDL design d, and thus, by property of the $\mathcal{H}$-VHDL elaboration relation, we can deduce $\sigma_e(id_f) = \texttt{false}$.
By construction, and by property of the initialization relation, we know that the output port identifier id_f is assigned to `false` in the generated `function` process during the initialization phase (i.e. the assignment is a part of a *reset* block). Thus, we can deduce $\sigma_0(id_f) = \texttt{false}$.
Rewriting $\sigma_0(id_f)$ as `false`, tautology.

□

A.1.7 Initial states and fired transitions

Lemma 14 (No fired at initial state). $\forall d \in design, \Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}}), \sigma_e, \sigma_0 \in \Sigma(\Delta), id_t \in Comps(\Delta), gm_t, ipm_t, opm_t$ *s.t. :*

- $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d.cs \xrightarrow{elab} \sigma_0$
- $\Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0$
- $\texttt{comp}(id_t, "transition", gm_t, ipm_t, opm_t) \in d.cs$

then $\sigma_0(id_t)("fired") = \texttt{false}$.

Proof. Assuming all the above hypotheses, let us show $\boxed{\sigma_0(id_t)("fired") = \texttt{false}.}$
By property of the initialization relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `fired_evaluation` process defined in the transition design architecture, we can deduce:

$$\sigma_0(id_t)("fired") = \sigma_0(id_t)("s\_firable") \,.\, \sigma_0(id_t)("s\_priority\_combination") \tag{A.2}$$

Rewriting the goal with Equation (A.2): $\boxed{\sigma_0(id_t)("sfa") \,.\, \sigma_0(id_t)("spc") = \texttt{false}.}$
By property of the initialization relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `firable` process defined in the transition design architecture, we can deduce $\sigma_0(id_t)("sfa") = \texttt{false}$.
Rewriting the goal with $\sigma_0(id_t)("sfa") = \texttt{false}$ and simplifying the goal, tautology. □

A.2 First Rising Edge

Definition 15 (First rising edge hypotheses). *Given an* $sitpn \in SITPN, d \in design, \gamma \in WM(sitpn, d), \Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$, $\sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma \in \Sigma(\Delta)$, $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value$, $\tau \in \mathbb{N}$, *assume that:*

- $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$ *and* $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e)$ *and* $\gamma \vdash E_p \overset{env}{=} E_c$
- σ_0 *is the initial state of* Δ: $\Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0$
- $E_c, \tau \vdash s_0 \xrightarrow{\uparrow_0} s_0$
- $\texttt{Inject}_\uparrow(\sigma_0, E_p, \tau, \sigma_i)$ *and* $\Delta, \sigma_i \vdash \text{d.cs} \xrightarrow{\uparrow} \sigma_\uparrow$ *and* $\Delta, \sigma_\uparrow \vdash \text{d.cs} \xrightarrow{\theta} \sigma$

Lemma 15 (First rising edge). *For all* $sitpn, d, \gamma, \Delta, \sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma, E_c, E_p, \tau$ *that verify the hypotheses of Definition 15, then* $\gamma, E_c, \tau \vdash s_0 \overset{\uparrow}{\approx} \sigma$.

Proof. By definition of the Full post rising edge state similarity relation, there are 8 points to prove.

1. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s_0.M(p) = \sigma(id_p)(\text{"s\_marking"})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $(upper(I_s(t)) = \infty \wedge s_0.I(t) \leq lower(I_s(t)) \Rightarrow s_0.I(t) = \sigma(id_t)(\text{"s\_time\_counter"}))$
   $\wedge (upper(I_s(t)) = \infty \wedge s_0.I(t) > lower(I_s(t)) \Rightarrow \sigma(id_t)(\text{"s\_time\_counter"}) = lower(I_s(t)))$
   $\wedge (upper(I_s(t)) \neq \infty \wedge s_0.I(t) > upper(I_s(t)) \Rightarrow \sigma(id_t)(\text{"s\_time\_counter"}) = upper(I_s(t)))$
   $\wedge (upper(I_s(t)) \neq \infty \wedge s_0.I(t) \leq upper(I_s(t)) \Rightarrow s_0.I(t) = \sigma(id_t)(\text{"s\_time\_counter"}))$.
3. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, s_0.reset_t(t) = \sigma(id_t)(\text{"s\_reinit\_time\_counter"})$.
4. $\forall a \in \mathcal{A}, id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a,\ s_0.ex(a) = \sigma(id_a)$.
5. $\forall f \in \mathcal{F}, id_f \in Outs(\Delta)\ s.t.\ \gamma(f) = id_f,\ s_0.ex(f) = \sigma(id_f)$.
6. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Sens(s_0.M) \Leftrightarrow \sigma(id_t)(\text{"s\_enabled"}) = \texttt{true}$.
7. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Sens(s_0.M) \Leftrightarrow \sigma(id_t)(\text{"s\_enabled"}) = \texttt{false}$.
8. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\sigma(id_t)(\text{"s\_condition\_combination"}) = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau, c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}$
   where $conds(t) = \{c \in \mathcal{C} \mid \mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1\}$.

- Apply the First rising edge equal marking lemma to solve 1.

- Apply the First rising edge equal time counters lemma to solve 2.
- Apply the First rising edge equal reset orders lemma to solve 3.
- Apply the First rising edge equal action executions lemma to solve 4.
- Apply the First rising edge equal function executions lemma to solve 5.
- Apply the First rising edge equal sensitized lemma to solve 6.
- Apply the First rising edge not equal sensitized lemma to solve 7.
- Apply the First rising edge equal condition combination lemma to solve 8.

□

A.2.1 First rising edge and marking

Lemma 16 (First rising edge equal marking). *For all* $sitpn, d, \gamma, \Delta, \sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma, E_c, E_p, \tau$ *that verify the hypotheses of Definition 15, then* $\forall p \in P, id_p \in Comps(\Delta)$ *s.t.* $\gamma(p) = id_p$, $s_0.M(p) = \sigma(id_p)(\text{"}s\_marking\text{"})$.

Proof. Given a p and an id_p s.t. $\gamma(p) = id_p$, let us show that $\boxed{s_0.M(p) = \sigma(id_p)(\text{"}s\_marking\text{"}).}$
By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.
By property of the $\texttt{Inject}_\uparrow$ relation, the $\mathcal{H}$-VHDL rising edge relation, the stabilize relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `marking` process defined in the place design architecture, we can deduce:

$$\sigma(id_p)(\text{"}sm\text{"}) = \sigma_0(id_p)(\text{"}sm\text{"}) + \sigma_0(id_p)(\text{"}sits\text{"}) - \sigma_0(id_p)(\text{"}sots\text{"}) \tag{A.3}$$

Rewriting the goal with Equation (A.3):
$\boxed{s_0.M(p) = \sigma_0(id_p)(\text{"}sm\text{"}) + \sigma_0(id_p)(\text{"}sits\text{"}) - \sigma_0(id_p)(\text{"}sots\text{"}).}$
Appealing to Lemmas 7 and 8, we can deduce $\sigma_0(id_p)(\text{"}sits\text{"}) = 0$ and $\sigma_0(id_p)(\text{"}sots\text{"}) = 0$.
Rewriting the goal with $\sigma_0(id_p)(\text{"}sits\text{"}) = 0$ and $\sigma_0(id_p)(\text{"}sots\text{"}) = 0$, $\boxed{s_0.M(p) = \sigma_0(id_p)(\text{"}sm\text{"}).}$

Appealing to Lemma 6, $s_0.M(p) = \sigma_0(id_p)(\text{"}sm\text{"})$. □

A.2.2 First rising edge and time counters

Lemma 17 (First rising edge equal time counters). *For all* $sitpn, d, \gamma, \Delta, \sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma, E_c, E_p, \tau$ *that verify the hypotheses of Definition 15, then*
$\forall t \in T_i, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t$,
$upper(I_s(t)) = \infty \wedge s_0.I(t) \leq lower(I_s(t)) \Rightarrow s_0.I(t) = \sigma(id_t)(\text{"}s\_time\_counter\text{"}) \wedge$
$upper(I_s(t)) = \infty \wedge s_0.I(t) > lower(I_s(t)) \Rightarrow \sigma(id_t)(\text{"}s\_time\_counter\text{"}) = lower(I_s(t)) \wedge$
$upper(I_s(t)) \neq \infty \wedge s_0.I(t) > upper(I_s(t)) \Rightarrow \sigma(id_t)(\text{"}s\_time\_counter\text{"}) = upper(I_s(t)) \wedge$
$upper(I_s(t)) \neq \infty \wedge s_0.I(t) \leq upper(I_s(t)) \Rightarrow s_0.I(t) = \sigma(id_t)(\text{"}s\_time\_counter\text{"}).$

Proof. Given a $t \in T_i$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, let us show that:

1. $\boxed{upper(I_s(t)) = \infty \wedge s_0.I(t) \leq lower(I_s(t)) \Rightarrow s_0.I(t) = \sigma(id_t)(\text{"s\_time\_counter"})}$
2. $\boxed{upper(I_s(t)) = \infty \wedge s_0.I(t) > lower(I_s(t)) \Rightarrow \sigma(id_t)(\text{"s\_time\_counter"}) = lower(I_s(t))}$
3. $\boxed{upper(I_s(t)) \neq \infty \wedge s_0.I(t) > upper(I_s(t)) \Rightarrow \sigma(id_t)(\text{"s\_time\_counter"}) = upper(I_s(t))}$
4. $\boxed{upper(I_s(t)) \neq \infty \wedge s_0.I(t) \leq upper(I_s(t)) \Rightarrow s_0.I(t) = \sigma(id_t)(\text{"s\_time\_counter"})}$

By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
Then, let us show the 4 previous points:

1. Assuming that $upper(I_s(t)) = \infty$ and $s_0.I(t) \leq lower(I_s(t))$, let us show $\boxed{s_0.I(t) = \sigma(id_t)(\text{"stc"}).}$

   By property of the $\texttt{Inject}_{\uparrow}$ relation, the $\mathcal{H}$-VHDL rising edge and stabilize relations, and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, we can deduce $\sigma(id_t)(\text{"stc"}) = \sigma_0(id_t)(\text{"stc"})$.

   Rewriting $\sigma(id_t)(\text{"stc"})$ as $\sigma_0(id_t)(\text{"stc"})$, $\boxed{s_0.I(t) = \sigma_0(id_t)(\text{"stc"}).}$

   Appealing to Lemma 9, $s_0.I(t) = \sigma_0(id_t)(\text{"stc"}).$

2. Assuming that $upper(I_s(t)) = \infty$ and $s_0.I(t) > lower(I_s(t))$, let us show $\boxed{\sigma(id_t)(\text{"stc"}) = lower(I_s(t)).}$

   By definition, $lower(I_s(t)) \in \mathbb{N}^*$ and $s_0.I(t) = 0$. Then, $lower(I_s(t)) < 0$ is a contradiction.

3. Assuming that $upper(I_s(t)) \neq \infty$ and $s_0.I(t) > upper(I_s(t))$, let us show $\boxed{\sigma(id_t)(\text{"stc"}) = upper(I_s(t))}$.

   By definition, $upper(I_s(t)) \in \mathbb{N}^*$ and $s_0.I(t) = 0$. Then, $upper(I_s(t)) < 0$ is a contradiction.

4. Assuming that $upper(I_s(t)) \neq \infty$ and $s_0.I(t) \leq upper(I_s(t))$, let us show $\boxed{s_0.I(t) = \sigma(id_t)(\text{"stc"}).}$

   By property of the $\texttt{Inject}_{\uparrow}$ relation, the $\mathcal{H}$-VHDL rising edge and stabilize relations, and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, we can deduce $\sigma(id_t)(\text{"stc"}) = \sigma_0(id_t)(\text{"stc"})$.

   Rewriting $\sigma(id_t)(\text{"stc"})$ as $\sigma_0(id_t)(\text{"stc"})$, $\boxed{s_0.I(t) = \sigma_0(id_t)(\text{"stc"}).}$

   Appealing to Lemma 9, $s_0.I(t) = \sigma_0(id_t)(\text{"stc"}).$

□

A.2.3 First rising edge and reset orders

Lemma 18 (First rising edge equal reset orders). *For all* $sitpn, d, \gamma, \Delta, \sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma, E_c, E_p, \tau$ *that verify the hypotheses of Definition 15, then*
$\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t, s_0.reset_t(t) = \sigma(id_t)(''s\_reinit\_time\_counter'')$.

Proof. Given a $t \in T$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, let us show that
$\boxed{s_0.reset_t(t) = \sigma(id_t)(''srtc'').}$
By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
By property of the stabilize relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `reinit_time_counter_evaluation` process defined in the transition design architecture, we can deduce:

$$\sigma(id_t)(''srtc'') = \sum_{i=0}^{\Delta(id_t)(''input\_arcs\_number'')-1} \sigma(id_t)(''reinit\_time'')[i] \tag{A.4}$$

Rewriting the goal with Equation (A.4): $\boxed{s_0.reset_t(t) = \sum_{i=0}^{\Delta(id_t)(''ian'')-1} \sigma(id_t)(''rt'')[i].}$
Let us perform case analysis on $input(t)$; there are two cases:

- **CASE** $input(t) = \emptyset$:

  By construction, $<\texttt{input\_arcs\_number} \Rightarrow 1> \in gm_t$, and by property of the $\mathcal{H}$-VHDL elaboration relation, we can deduce $\Delta(id_t)(''ian'') = 1$.

  By construction, $< \texttt{reinit\_time}(0) \Rightarrow \texttt{false} > \in ipm_t$, and by property of the $\mathcal{H}$-VHDL stabilize relation, $\sigma(id_t)(''rt'')[0] = \texttt{false}$.

  Rewriting the goal with $\Delta(id_t)(''ian'') = 1$ and $\sigma(id_t)(''rt'')[0] = \texttt{false}$, $\boxed{s_0.reset_t(t) = \texttt{false}.}$

  By definition of s_0, $s_0.reset_t(t) = \texttt{false}.$

- **CASE** $input(t) \neq \emptyset$:

  By construction, $<\texttt{input\_arcs\_number} \Rightarrow |input(t)|> \in gm_t$, and by property of the $\mathcal{H}$-VHDL elaboration relation, we can deduce $\Delta(id_t)(''ian'') = |input(t)|$.

  Rewriting $\Delta(id_t)(''ian'')$ as $|input(t)|$, $\boxed{s_0.reset_t(t) = \sum_{i=0}^{|input(t)|-1} \sigma(id_t)(''rt'')[i].}$

  By definition of s_0, $s_0.reset_t(t) = \texttt{false}$. Rewriting $s_0.reset_t(t)$ as $\texttt{false}$,
  $\boxed{\sum_{i=0}^{|input(t)|-1} \sigma(id_t)(''rt'')[i] = \texttt{false}.}$

  Given a $i \in [0, |input(t)| - 1]$, let us show $\boxed{\sigma(id_t)(''rt'')[i] = \texttt{false}.}$

By construction, and since $input(t) \neq \emptyset$, there exist a $p \in input(t)$, an $id_p \in Comps(\Delta)$ s.t. $\gamma(p) = id_p$, a gm_p, an ipm_p, an opm_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and there exist a $j \in [0, |output(p)| - 1]$ and an $id_{ji} \in Sigs(\Delta)$ s.t. $<\texttt{reinit\_transition\_time(j)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in opm_p$ and $<\texttt{reinit\_time(i)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in ipm_t$.

By property of the stabilize relation, $<\texttt{reinit\_transition\_time(j)} \Rightarrow id_{ji}> \in opm_p$ and $<\texttt{reinit\_time(i)} \Rightarrow id_{ji}> \in ipm_t$, we can deduce $\sigma(id_t)(''rt'')[i] = \sigma(id_{ji}) = \sigma(id_p)(''rtt'')[j]$.

Rewriting $\sigma(id_t)(''rt'')[i]$ as $\sigma(id_{ji})$ and $\sigma(id_{ji})$ as $\sigma(id_p)(''rtt'')[j]$, $\boxed{\sigma(id_p)(''rtt'')[j] = \texttt{false}.}$

By property of the $\mathcal{H}$-VHDL rising edge and stabilize relations, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the process defined in the place design architecture, we can deduce:

$$\begin{aligned}\sigma(id_p)(''rtt'')[j] =&((\sigma_0(id_p)(''oat'')[j] = \texttt{basic} + \sigma_0(id_p)(''oat'')[j] = \texttt{test})\\ &.(\sigma_0(id_p)(''sm'') - \sigma_0(id_p)(''sots'') < \sigma_0(id_p)(''oaw'')[j])\\ &.(\sigma_0(id_p)(''sots'') > 0))\\ &+ (\sigma_0(id_p)(''otf'')[j])\end{aligned} \tag{A.5}$$

Rewriting the goal with Equation (A.5),

$$\boxed{\begin{aligned}\texttt{false} =&((\sigma_0(id_p)(''oat'')[j] = \texttt{basic} + \sigma_0(id_p)(''oat'')[j] = \texttt{test})\\ &.(\sigma_0(id_p)(''sm'') - \sigma_0(id_p)(''sots'') < \sigma_0(id_p)(''oaw'')[j])\\ &.(\sigma_0(id_p)(''sots'') > 0))\\ &+ (\sigma_0(id_p)(''otf'')[j])\end{aligned}}$$

By construction, there exists an $id_{fj} \in Sigs(\Delta)$ s.t. $<\texttt{fired} \Rightarrow \texttt{id}_{\texttt{fj}}> \in opm_t$ and $<\texttt{output\_transitions\_fired(j)} \Rightarrow \texttt{id}_{\texttt{fj}}> \in ipm_p$.

By property of the initialization relation, $<\texttt{fired} \Rightarrow \texttt{id}_{\texttt{fj}}> \in opm_t$ and $<\texttt{output\_transitions\_fired(j)} \Rightarrow \texttt{id}_{\texttt{fj}}> \in ipm_p$, we can deduce $\sigma_0(id_p)(''otf'')[j] = \sigma_0(id_{fj}) = \sigma_0(id_t)(''fired'')$.

Appealing to Lemma 14, we can deduce $\sigma_0(id_t)(''fired'') = \texttt{false}$ and consequently $\sigma_0(id_p)(''otf'')[j] = \texttt{false}$.

Rewriting $\sigma_0(id_p)(''otf'')[j]$ as $\texttt{false}$ and simplifying the goal,

$$\boxed{\begin{aligned}false =&((\sigma_0(id_p)(''oat'')[j] = \texttt{BASIC} + \sigma_0(id_p)(''oat'')[j] = \texttt{TEST})\\ &.(\sigma_0(id_p)(''sm'') - \sigma_0(id_p)(''sots'') < \sigma_0(id_p)(''oaw'')[j])\\ &.(\sigma_0(id_p)(''sots'') > 0))\end{aligned}}$$

Appealing to Lemma 8, we can deduce $\sigma_0(id_p)(''sots'') = 0$.

Rewriting $\sigma_0(id_p)(''sots'')$ as 0 and simplifying the goal, tautology.

□

A.2.4 First rising edge and action executions

Lemma 19 (First rising edge equal action executions). *For all* $sitpn, d, \gamma, \Delta, \sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma, E_c, E_p, \tau$ *that verify the hypotheses of Definition 15, then*
$\forall a \in \mathcal{A}, id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a,\ s_0.ex(a) = \sigma(id_a)$.

Proof. Given an $a \in \mathcal{A}$ and an $id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a$, let us show that $\boxed{s_0.ex(a) = \sigma(id_a).}$
By construction, id_a is an output port identifier of Boolean type in the $\mathcal{H}$-VHDL design d. The generated `action` process assigns a value to the output port id_a only during the initialization phase or a falling edge phase.
By property of the $\texttt{Inject}_\uparrow$, $\mathcal{H}$-VHDL rising edge and stabilize relations, we can deduce $\sigma(id_a) = \sigma_0(id_a)$.
Rewriting $\sigma(id_a)$ as $\sigma_0(id_a)$, $\boxed{s_0.ex(a) = \sigma_0(id_a).}$ Appealing to Lemma 12, $s_0.ex(a) = \sigma_0(id_a)$. □

A.2.5 First rising edge and function executions

Lemma 20 (First rising edge equal function executions). *For all* $sitpn, d, \gamma, \Delta, \sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma, E_c, E_p, \tau$ *that verify the hypotheses of Definition 15, then*
$\forall f \in \mathcal{F}, id_f \in Outs(\Delta)\ s.t.\ \gamma(f) = id_f,\ s_0.ex(f) = \sigma(id_f)$.

Proof. Given an $f \in \mathcal{F}$ and an $id_f \in Outs(\Delta)$ s.t. $\gamma(f) = id_f$, let us show that $\boxed{s_0.ex(f) = \sigma(id_f).}$
Rewriting $s_0.ex(f)$ as `false`, by definition of s_0, $\boxed{\sigma(id_f) = \texttt{false}.}$
By construction, id_f is an output port identifier of Boolean type in the $\mathcal{H}$-VHDL design d. The generated `function` process assigns a value to the output port id_f only during the initialization phase or during a rising edge phase.
By construction, the `function` process is defined in the behavior of design d, i.e.
$\texttt{ps}(\text{"}function\text{"}, \emptyset, sl, ss) \in d.cs$.
Let $trs(f)$ be the set of transitions associated to function f, i.e $trs(f) = \{t \in T \mid \mathbb{F}(t, f) = true\}$.
Let us perform case analysis on $trs(f)$; there are two cases:

- **CASE** $trs(f) = \emptyset$:

  By construction, $\texttt{id}_\texttt{f} \Leftarrow \texttt{false} \in ss_\uparrow$ where $ss_\uparrow$ is the part of the ''`function`'' process body executed during a rising edge phase (i.e. a rising edge block statement).

  By property of the $\mathcal{H}$-VHDL rising edge and the stabilize relation, $\sigma(id_f) = \texttt{false}.$

- **CASE** $trs(f) \neq \emptyset$:

  By construction, $\texttt{id}_\texttt{f} \Leftarrow \texttt{id}_{\texttt{ft}_0} + \cdots + \texttt{id}_{\texttt{ft}_n} \in ss_\uparrow$ where $ss_\uparrow$ is the part of the ''`function`'' process body executed during the rising edge phase, and $n = |trs(f)| - 1$, and for all $i \in [0, n-1]$, id_{ft_i} is a internal signal of design d.

  By property of the $\texttt{Inject}_\uparrow$, the $\mathcal{H}$-VHDL rising edge and stabilize relations, we can deduce $\sigma(id_f) = \sigma_0(id_{ft_0}) + \cdots + \sigma_0(id_{ft_n})$.

Rewriting $\sigma(id_f)$ as $\sigma_0(id_{ft_0}) + \cdots + \sigma_0(id_{ft_n})$, $\boxed{\sigma_0(id_{ft_0}) + \cdots + \sigma_0(id_{ft_n}) = \texttt{false}.}$

By construction, for all id_{ft_i}, there exist a $t_i \in trs(f)$ and an id_{t_i} s.t. $\gamma(t_i) = id_{t_i}$.

By construction and by definition of id_{t_i}, there exist gm_{t_i}, ipm_{t_i} and opm_{t_i} s.t. $\texttt{comp}(id_{t_i}, "transition", gm_{t_i}, ipm_{t_i}, opm_{t_i}) \in d.cs$.

By construction, $<\texttt{fired} \Rightarrow \texttt{id}_{\texttt{ft}_\texttt{i}}> \in opm_{t_i}$, and by property of the initialization relation $\sigma_0(id_{ft_i}) = \sigma_0(id_{t_i})("fired")$.

Rewriting $\sigma_0(id_{ft_i})$ as $\sigma_0(id_{t_i})("fired")$, $\boxed{\sigma_0(id_{t_0})("fired") + \cdots + \sigma_0(id_{t_n})("fired") = \texttt{false}.}$

Appealing to Lemma 14, we can deduce $\sigma_0(id_{t_i})("fired") = \texttt{false}$.

Rewriting all $\sigma_0(id_{t_i})("fired")$ as $\texttt{false}$ and simplifying the goal, tautology.

□

A.2.6 First rising edge and sensitization

Lemma 21 (First rising edge equal sensitized). *For all $sitpn, d, \gamma, \Delta, \sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma, E_c, E_p, \tau$ that verify the hypotheses of Definition 15, then*
$\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t, t \in Sens(s_0.M) \Leftrightarrow \sigma(id_t)("s\_enabled") = \texttt{true}$.

Proof. See the proof of Lemma 30. □

Lemma 22 (First rising edge not equal sensitized). *For all $sitpn, d, \gamma, \Delta, \sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma, E_c, E_p, \tau$ that verify the hypotheses of Definition 15, then*
$\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t, t \notin Sens(s_0.M) \Leftrightarrow \sigma(id_t)("s\_enabled") = \texttt{false}$.

Proof. See the proof of Lemma 31. □

A.2.7 First rising edge and condition combination

Lemma 23 (First rising edge equal condition combination). *For all $sitpn, d, \gamma, \Delta, \sigma_e, \sigma_0, \sigma_i, \sigma_\uparrow, \sigma, E_c, E_p, \tau$ that verify the hypotheses of Definition 15, then*
$\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t$,

$$\sigma(id_t)("s\_condition\_combination") = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau, c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}$$

where $conds(t) = \{c \in \mathcal{C} \mid \mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1\}$.

Proof. See the proof of Lemma 25. □

A.3 Rising Edge

Definition 16 (Rising edge hypotheses). *Given an $sitpn \in SITPN, d \in design, \gamma \in WM(sitpn, d)$, $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $\Delta \in ElDesign(d, \mathcal{D}_\mathcal{H})$, $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value$, $\tau \in \mathbb{N}$, $s, s' \in S(sitpn)$, $\sigma_e, \sigma, \sigma_i, \sigma_\uparrow, \sigma' \in \Sigma(\Delta)$, assume that:*

- $\lfloor sitpn \rfloor_{\mathcal{H}} = (d, \gamma)$ *and* $\gamma \vdash E_p \overset{env}{=} E_c$ *and* $\mathcal{D}_{\mathcal{H}}, \emptyset \vdash \mathrm{d} \xrightarrow{elab} \Delta, \sigma_e$
- $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$
- $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$
- $\texttt{Inject}_{\uparrow}(\sigma, E_p, \tau, \sigma_i)$ *and* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma_i \vdash \mathrm{d.cs} \xrightarrow{\uparrow} \sigma_{\uparrow}$ *and* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma_{\uparrow} \vdash \mathrm{d.cs} \overset{\rightsquigarrow}{\longrightarrow} \sigma'$
- *State* σ *is a stable design state:* $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma \vdash \mathrm{d.cs} \xrightarrow{comb} \sigma$

A.3.1 Rising edge and Marking

Lemma 24 (Rising edge equal marking). *For all* $sitpn$, d, γ, E_c, E_p, τ, Δ, σ_e, s, s', σ, σ_i, $\sigma_{\uparrow}$, σ' *that verify the hypotheses of Def. 16, then* $\forall p, id_p$ *s.t.* $\gamma(p) = id_p$, $s'.M(p) = \sigma'(id_p)(\text{"s\_marking"})$.

Proof. Given a $p \in P$, let us show $\boxed{s'.M(p) = \sigma'(id_p)(\text{"s\_marking"}).}$
By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.
By definition of the SITPN state transition relation on rising edge:

$$s'.M(p) = s.M(p) - \sum_{t \in Fired(s)} pre(p,t) + \sum_{t \in Fired(s)} post(t,p) \tag{A.6}$$

By property of the $\texttt{Inject}_{\uparrow}$, the $\mathcal{H}$-VHDL rising edge and the stabilize relations, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `marking` process defined in the place design architecture, we can deduce:

$$\begin{aligned}\sigma'(id_p)(\text{"sm"}) = \sigma(id_p)(\text{"sm"}) - \sigma(id_p)(\text{"s\_output\_token\_sum"}) \\ + \sigma(id_p)(\text{"s\_input\_token\_sum"})\end{aligned} \tag{A.7}$$

Rewriting the goal with A.6 and A.7,

$$\boxed{s.M(p) - \sum_{t \in Fired(s)} pre(p,t) + \sum_{t \in Fired(s)} post(t,p) = \sigma(id_p)(\text{"sm"}) - \sigma(id_p)(\text{"sots"}) + \sigma(id_p)(\text{"sits"}).}$$

By definition of the Full post falling edge state similarity relation, we can deduce $s.M(p) = \sigma(id_p)(\text{"sm"})$, $\sum_{t \in Fired(s)} pre(p,t) = \sigma(id_p)(\text{"sots"})$ and $\sum_{t \in Fired(s)} post(t,p) = \sigma(id_p)(\text{"sits"})$, and thus,

$$s.M(p) - \sum_{t \in Fired(s)} pre(p,t) + \sum_{t \in Fired(s)} post(t,p) = \sigma(id_p)(\text{"sm"}) - \sigma(id_p)(\text{"sots"}) + \sigma(id_p)(\text{"sits"}).$$

□

A.3.2 Rising edge and condition combination

Lemma 25 (Rising edge equal condition combination). *For all sitpn, d, γ, E_c, E_p, τ, Δ, σ_e, s, s', σ, σ_i, $\sigma_\uparrow$, σ' that verify the hypotheses of Def. 16, then*
$\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t$,

$$\sigma'(id_t)(\text{"s\_condition\_combination"}) = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}$$

where $conds(t) = \{c \in \mathcal{C} \mid \mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1\}$.

Proof. Given a t and an id_t s.t. $\gamma(t) = id_t$, let us show

$$\boxed{\sigma'(id_t)(\text{"s\_condition\_combination"}) = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}.}$$

By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
By property of the $\mathcal{H}$-VHDL stabilize relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `condition_evaluation` process defined in the transition design architecture, we can deduce:

$$\sigma'(id_t)(\text{"scc"}) = \prod_{i=0}^{\Delta(id_t)(\text{"conditions\_number"})-1} \sigma'(id_t)(\text{"input\_conditions"})[i] \tag{A.8}$$

Rewriting the goal with A.8,

$$\boxed{\prod_{i=0}^{\Delta(id_t)(\text{"cn"})-1} \sigma'(id_t)(\text{"ic"})[i] = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}.}$$

Let us perform case analysis on $conds(t)$; there are two cases:

- **CASE** $conds(t) = \emptyset$: $\boxed{\prod_{i=0}^{\Delta(id_t)(\text{"cn"})-1} \sigma'(id_t)(\text{"ic"})[i] = \texttt{true}.}$

  By construction, $<\texttt{conditions\_number} \Rightarrow 1> \in gm_t$ and $<\texttt{input\_conditions}(0) \Rightarrow \texttt{true}> \in ipm_t$.

  By property of the stabilize relation, $<\texttt{conditions\_number} \Rightarrow 1> \in gm_t$ and $<\texttt{input\_conditions(0)} \Rightarrow \texttt{true}> \in ipm_t$, we can deduce $\Delta(id_t)(\text{"cn"}) = 1$ and $\sigma'(id_t)(\text{"ic"})[0] = \texttt{true}$.

  Rewriting the goal with $\Delta(id_t)(\text{"cn"}) = 1$ and $\sigma'(id_t)(\text{"ic"})[0] = \texttt{true}$, tautology.

- **CASE** $conds(t) \neq \emptyset$:
  By construction, $<\texttt{conditions\_number} \Rightarrow |\texttt{conds(t)}|> \in gm_t$, and by property of the stabilize relation, we can deduce $\Delta(id_t)(\text{"cn"}) = |conds(t)|$.

Rewriting the goal with $\Delta(id_t)("cn") = |conds(t)|$:

$$\boxed{\prod_{i=0}^{|conds(t)|-1} \sigma'(id_t)("ic")[i] = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}}$$

Let us reason by induction on the left product term:

– **BASE CASE**: $\prod_{i=0}^{|conds(t)|-1} \sigma'(id_t)("ic")[i] = 0$ and $|conds(t)| - 1 < 0$. Thus, we can deduce that $|conds(t)| = 0$ which contradicts $conds(t) \neq \emptyset$.

– **INDUCTION CASE**:

$$\forall conds' \subseteq \mathcal{C},\ \prod_{i=1}^{|conds(t)|-1} \sigma'(id_t)("ic")[i] = \prod_{c \in conds'} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}$$

$$\boxed{\sigma'(id_t)("ic")[0] \cdot \prod_{i=1}^{|conds(t)|-1} \sigma'(id_t)("ic")[i] = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}}$$

By construction, for all $i \in [0, |conds(t)| - 1]$, there exists $c \in conds(t)$ and an $id_c \in Ins(\Delta)$ such that

* $\gamma(c) = id_c$
* $\mathbb{C}(t,c) = 1$ implies $<\texttt{input\_conditions(i)} \Rightarrow \texttt{id}_\texttt{c}> \in ipm_t$
* $\mathbb{C}(t,c) = -1$ implies $<\texttt{input\_conditions(i)} \Rightarrow \texttt{not id}_\texttt{c}> \in ipm_t$

For $i = 0$, let us take such a $c \in conds(t)$ and an id_c with the above properties. By definition of $c \in conds(t)$, we can deduce $\mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1$. Let us perform case analysis on $\mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1$:

* **CASE** $\mathbb{C}(t,c) = 1$:
  Then, we have $<\texttt{input\_conditions}(0) \Rightarrow \texttt{id}_\texttt{c}> \in ipm_t$ and by property of the stabilize relation, we can deduce $\sigma(id_t)("ic")[0] = \sigma'(id_c)$.
  Rewriting the goal with $\sigma(id_t)("ic")[0] = \sigma'(id_c)$:

$$\boxed{\sigma'(id_c) \cdot \prod_{i=1}^{|conds(t)|-1} \sigma'(id_t)("ic")[i] = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}}$$

  By property of the $\texttt{Inject}_\uparrow$ relation and $id_c \in Ins(\Delta)$, we can deduce $\sigma'(id_c) = E_p(\tau, \uparrow)(id_c)$.
  By property of $\gamma \vdash E_p \overset{env}{=} E_c$, we can deduce $E_p(\tau, \uparrow)(id_c) = E_c(\tau, c)$.
  Rewriting the goal with $\sigma'(id_c) = E_p(\tau, \uparrow)(id_c)$ and $E_p(\tau, \uparrow)(id_c) = E_c(\tau, c)$:

$$\boxed{E_c(\tau, c) \cdot \prod_{i=1}^{|conds(t)|-1} \sigma'(id_t)("ic")[i] = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}}$$

  By definition of the $\prod$ operator, we can rewrite the right term of the goal as follows:

$$E_c(\tau,c)\,.\prod_{i=1}^{|conds(t)|-1}\sigma'(id_t)(''ic'')[i] = E_c(\tau,c)\,.\prod_{c'\in conds(t)\setminus\{c\}}\begin{cases}E_c(\tau,c') & if\ \mathbb{C}(t,c')=1\\ \texttt{not}(E_c(\tau,c')) & if\ \mathbb{C}(t,c')=-1\end{cases}$$

Appealing to the induction hypothesis, tautology.

* **CASE** $\mathbb{C}(t,c)=-1$:
Then, we have $<\texttt{input\_conditions(0)}\Rightarrow \texttt{not id}_c>\in ipm_t$ and by property of the stabilize relation, we can deduce $\sigma(id_t)(''ic'')[0]=\texttt{not}\ \sigma'(id_c)$.
Rewriting the goal with $\sigma(id_t)(''ic'')[0]=\texttt{not}\ \sigma'(id_c)$:

$$\texttt{not}\ \sigma'(id_c)\,.\prod_{i=1}^{|conds(t)|-1}\sigma'(id_t)(''ic'')[i] = \prod_{c\in conds(t)}\begin{cases}E_c(\tau,c) & if\ \mathbb{C}(t,c)=1\\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c)=-1\end{cases}$$

By property of the $\texttt{Inject}_{\uparrow}$ relation and $id_c\in Ins(\Delta)$, we can deduce $\sigma'(id_c)=E_p(\tau,\uparrow)(id_c)$.
By property of $\gamma\vdash E_p\overset{env}{=}E_c$, we can deduce $E_p(\tau,\uparrow)(id_c)=E_c(\tau,c)$.
Rewriting the goal with $\sigma'(id_c)=E_p(\tau,\uparrow)(id_c)$ and $E_p(\tau,\uparrow)(id_c)=E_c(\tau,c)$:

$$\texttt{not}\ E_c(\tau,c)\,.\prod_{i=1}^{|conds(t)|-1}\sigma'(id_t)(''ic'')[i] = \prod_{c\in conds(t)}\begin{cases}E_c(\tau,c) & if\ \mathbb{C}(t,c)=1\\ \texttt{not}(E_c(\tau,c)) & if\ \mathbb{C}(t,c)=-1\end{cases}$$

By definition of the $\prod$ operator, we can rewrite the right term of the goal as follows:

$$\texttt{not}\ E_c(\tau,c)\,.\prod_{i=1}^{|conds(t)|-1}\sigma'(id_t)(''ic'')[i] = \texttt{not}\ E_c(\tau,c)\,.\prod_{c'\in conds(t)\setminus\{c\}}\begin{cases}E_c(\tau,c') & if\ \mathbb{C}(t,c')=\\ \texttt{not}(E_c(\tau,c')) & if\ \mathbb{C}(t,c')=\end{cases}$$

Appealing to the induction hypothesis, tautology.

□

A.3.3 Rising edge and time counters

Lemma 26 (Rising edge equal time counters). *For all sitpn, d, γ, E_c, E_p, τ, Δ, σ_e, s, s', σ, σ_i, $\sigma_{\uparrow}$, σ' that verify the hypotheses of Def. 16, then*
$\forall t\in T_i, id_t\in Comps(\Delta)$ *s.t.* $\gamma(t)=id_t$,
$\big(upper(I_s(t))=\infty\wedge s'.I(t)\leq lower(I_s(t))\Rightarrow s'.I(t)=\sigma'(id_t)(''s\_time\_counter'')\big)$
$\wedge\big(upper(I_s(t))=\infty\wedge s'.I(t)>lower(I_s(t))\Rightarrow \sigma'(id_t)(''s\_time\_counter'')=lower(I_s(t))\big)$
$\wedge\big(upper(I_s(t))\neq\infty\wedge s'.I(t)>upper(I_s(t))\Rightarrow \sigma'(id_t)(''s\_time\_counter'')=upper(I_s(t))\big)$
$\wedge\big(upper(I_s(t))\neq\infty\wedge s'.I(t)\leq upper(I_s(t))\Rightarrow s'.I(t)=\sigma'(id_t)(''s\_time\_counter'')\big)$.

Proof. Given a $t\in T_i$ and an $id_t\in Comps(\Delta)$ s.t. $\gamma(t)=id_t$, let us show

$$\begin{array}{l}\big(upper(I_s(t))=\infty\wedge s'.I(t)\leq lower(I_s(t))\Rightarrow s'.I(t)=\sigma'(id_t)(''s\_time\_counter'')\big)\\ \wedge\big(upper(I_s(t))=\infty\wedge s'.I(t)>lower(I_s(t))\Rightarrow \sigma'(id_t)(''s\_time\_counter'')=lower(I_s(t))\big)\\ \wedge\big(upper(I_s(t))\neq\infty\wedge s'.I(t)>upper(I_s(t))\Rightarrow \sigma'(id_t)(''s\_time\_counter'')=upper(I_s(t))\big)\\ \wedge\big(upper(I_s(t))\neq\infty\wedge s'.I(t)\leq upper(I_s(t))\Rightarrow s'.I(t)=\sigma'(id_t)(''s\_time\_counter'')\big)\end{array}$$

By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t)\in d.cs$.

Then, there are 4 points to show:

1. $\boxed{upper(I_s(t)) = \infty \wedge s'.I(t) \leq lower(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"})}$

   Assuming that $upper(I_s(t)) = \infty$ and $s'.I(t) \leq lower(I_s(t))$, let us show $\boxed{s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"}).}$

   By property of the $\texttt{Inject}_{\uparrow}$, $\mathcal{H}$-VHDL rising edge and stabilize relations, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `time_counter` process defined in the transition design architecture, we can deduce $\sigma'(id_t)(\text{"stc"}) = \sigma(id_t)(\text{"stc"})$.

   By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $s.I(t) = \sigma(id_t)(\text{"stc"})$.

   Rewriting the goal with $\sigma'(id_t)(\text{"stc"}) = \sigma(id_t)(\text{"stc"})$ and $s.I(t) = \sigma(id_t)(\text{"stc"})$, tautology.

2. $\boxed{upper(I_s(t)) = \infty \wedge s'.I(t) > lower(I_s(t)) \Rightarrow \sigma'(id_t)(\text{"s\_time\_counter"}) = lower(I_s(t).}$

   Proved in the same fashion as 1.

3. $\boxed{upper(I_s(t)) \neq \infty \wedge s'.I(t) > upper(I_s(t)) \Rightarrow \sigma'(id_t)(\text{"s\_time\_counter"}) = upper(I_s(t).}$

   Proved in the same fashion as 1.

4. $\boxed{upper(I_s(t)) \neq \infty \wedge s'.I(t) \leq upper(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"})}$

   Proved in the same fashion as 1.

□

A.3.4 Rising edge and reset orders

Lemma 27 (Rising edge equal reset orders). *For all $sitpn$, d, γ, E_c, E_p, τ, Δ, σ_e, s, s', σ, σ_i, $\sigma_{\uparrow}$, σ' that verify the hypotheses of Def. 16, then*
$\forall t \in T_i, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t$, $s'.reset_t(t) = \sigma'(id_t)(\text{"s\_reinit\_time\_counter"})$

Proof. Given a $t \in T_i$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, let us show $\boxed{s'.reset_t(t) = \sigma'(id_t)(\text{"s\_reinit\_time\_counter"}).}$
By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
By property of the $\mathcal{H}$-VHDL stabilize relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `reinit_time_counter_evaluation` process defined in the transition design architecture, we can deduce:

$$\sigma'(id_t)(\text{"srtc"}) = \sum_{i=0}^{\Delta(id_t)(\text{"input\_arcs\_number"})-1} \sigma'(id_t)(\text{"reinit\_time"})[i] \tag{A.9}$$

Rewriting the goal with (A.9), $\boxed{s'.reset_t(t) = \sum_{i=0}^{\Delta(id_t)("ian")-1} \sigma'(id_t)("rt")[i].}$

Let us perform case analysis on $input(t)$; there are two cases:

- **CASE** $input(t) = \emptyset$:

  By construction, $<$`input_arcs_number` $\Rightarrow$ `1`$> \in gm_t$, and by property of the elaboration relation, we can deduce $\Delta(id_t)("ian") = 1$.

  By construction, there exists an $id_{ft} \in Sigs(\Delta)$ s.t. $<$`reinit_time(0)` $\Rightarrow$ `id`$_{\texttt{ft}}> \in ipm_t$ and $<$`fired` $\Rightarrow$ `id`$_{\texttt{ft}}> \in opm_t$, and by property of the stabilize relation and `comp`$(id_t,$ `transition`$, g_t, i_t, o_t) \in d.cs$, we can deduce $\sigma'(id_t)("rt")[0] = \sigma'(id_{ft}) = \sigma'(id_t)("fired")$.

  Rewriting the goal with $\Delta(id_t)("ian") = 1$ and $\sigma'(id_t)("rt")[0] = \sigma'(id_{ft}) = \sigma'(id_t)("fired")$:
  $\boxed{s'.reset_t(t) = \sigma'(id_t)("fired").}$

  By property of the stabilize relation, `comp`$(id_t,$ `transition`$, g_t, i_t, o_t) \in d.cs$, and through the examination of the `fired_evaluation` process, we can deduce:

  $$\sigma'(id_t)("fired") = \sigma'(id_t)("s\_firable") \,.\, \sigma'(id_t)("s\_priority\_combination") \tag{A.10}$$

  Rewriting the goal with (A.10):
  $\boxed{s'.reset_t(t) = \sigma'(id_t)("s\_firable") \,.\, \sigma'(id_t)("s\_priority\_combination").}$

  By property of the stabilize relation, `comp`$(id_t,$ `transition`$, g_t, i_t, o_t) \in d.cs$, and through the examination of the `priority_authorization_evaluation` process defined in the transition design architecture, we can deduce:

  $$\sigma'(id_t)("spc") = \prod_{i=0}^{\Delta(id_t)("ian")-1} \sigma'(id_t)("priority\_authorizations")[i] \tag{A.11}$$

  As $\Delta(id_t)("ian") = 1$, we can deduce $\prod_{i=0}^{\Delta(id_t)("ian")-1} \sigma'(id_t)("pauths")[i] = \sigma'(id_t)("pauths")[0]$.

  Rewriting the goal with (A.11) and $\prod_{i=0}^{\Delta(id_t)("ian")-1} \sigma'(id_t)("pauths")[i] = \sigma'(id_t)("pauths")[0]$:
  $\boxed{s'.reset_t(t) = \sigma'(id_t)("s\_firable") \,.\, \sigma'(id_t)("pauths")[0].}$

  By construction, $<$`priority_authorizations(0)` $\Rightarrow$ `true`$> \in ipm_t$, and by property of the stabilize relation and `comp`$(id_t,$ `transition`$, g_t, i_t, o_t) \in d.cs$, we can deduce $\sigma'(id_t)("pauths")[0] =$ `true`.

  Rewriting the goal with $\sigma'(id_t)("pauths")[0] =$ `true` , and simplifying the equation:
  $\boxed{s'.reset_t(t) = \sigma'(id_t)("s\_firable").}$

  Let us perform case analysis on $t \in Fired(s)$ or $t \notin Fired(s)$:

– **CASE** $t \in Fired(s)$:

By property of $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$ (Rule **??**), we can deduce $s'.reset_t(t) = \texttt{true}$.
Rewriting the goal with $s'.reset_t(t) = \texttt{true}$: $\boxed{\sigma'(id_t)(''s\_firable'') = \texttt{true}.}$
By property of the stabilize, the $\mathcal{H}$-VHDL rising edge and the $\texttt{Inject}_{\uparrow}$ relations, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `firable` process defined in the transition design architecture, we can deduce $\sigma(id_t)(''s\_firable'') = \sigma'(id_t)(''s\_firable'')$.
Rewriting the goal with $\sigma(id_t)(''s\_firable'') = \sigma'(id_t)(''s\_firable'')$, $\boxed{\sigma(id_t)(''s\_firable'') = \texttt{true}.}$
By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $t \in Firable(s) \Leftrightarrow \sigma(id_t)(''sfa'') = \texttt{true}$.
Rewriting the goal with $t \in Firable(s) \Leftrightarrow \sigma(id_t)(''sfa'') = \texttt{true}$, $\boxed{t \in Firable(s).}$
By property of $t \in Fired(s)$, $t \in Firable(s)$.

– **CASE** $t \notin Fired(s)$:
By property of $input(t) = \emptyset$, there does not exist any input place connected to t by a `basic` or `test` arc. Thus, by property of $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$ (Rule **??**), we can deduce $s'.reset_t(t) = \texttt{false}$.
Rewriting the goal with $s'.reset_t(t) = \texttt{false}$: $\boxed{\sigma'(id_t)(''s\_firable'') = \texttt{false}.}$
By property of the stabilize, the $\mathcal{H}$-VHDL rising edge and the $\texttt{Inject}_{\uparrow}$ relations, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `firable` process defined in the transition design architecture, we can deduce $\sigma(id_t)(''sfa'') = \sigma'(id_t)(''sfa'')$.
Rewriting the goal with $\sigma(id_t)(''sfa'') = \sigma'(id_t)(''sfa'')$, $\boxed{\sigma(id_t)(''sfa'') = \texttt{false}.}$
By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $t \notin Firable(s) \Leftrightarrow \sigma(id_t)(''sfa'') = \texttt{false}$.
By property of $t \notin Fired(s)$ and $input(t) = \emptyset$, $t \notin Firable(s)$.

- **CASE** $input(t) \neq \emptyset$:
By construction, $<\texttt{input\_arcs\_number} \Rightarrow |input(t)|> \in gm_t$, and by property of the elaboration relation, we can deduce $\Delta(id_t)(''ian'') = |input(t)|$.
Rewriting the goal with $\Delta(id_t)(''ian'') = |input(t)|$, $\boxed{s'.reset_t(t) = \sum_{i=0}^{|input(t)|-1} \sigma'(id_t)(''rt'')[i].}$
Let us perform case analysis on $t \in Fired(s)$ or $t \notin Fired(s)$:

– **CASE** $t \in Fired(s)$:

By property of $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$ (Rule **??**), we can deduce $s'.reset_t(t) = \texttt{true}$.
Rewriting the goal with $s'.reset_t(t) = \texttt{true}$, $\boxed{\sum_{i=0}^{|input(t)|-1} \sigma'(id_t)(''rt'')[i] = \texttt{true}.}$
To prove the goal, let us show $\boxed{\exists i \in [0, |input(t)| - 1] \text{ s.t. } \sigma'(id_t)(''rt'')[i] = \texttt{true}.}$

By construction, and $input(t) \neq \emptyset$, there exist $p \in input(t)$ and $id_p \in Comps(\Delta)$ s.t. $\gamma(p) = id_p$.
By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.
By construction, there exist an $i \in [0, |input(t)| - 1]$, a $j \in [0, |output(p)| - 1]$ and $id_{ji} \in Sigs(\Delta)$ s.t. $<\texttt{reinit\_transition\_time(j)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in opm_p$ and $<\texttt{reinit\_time(i)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in ipm_t$. Let us take such an i, j and id_{ji}, and let us use i to prove the goal: $\boxed{\sigma'(id_t)(''rt'')[i] = \texttt{true}.}$
By property of the stabilize relation, $<\texttt{reinit\_transition\_time(j)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in opm_p$ and $<\texttt{reinit\_time(i)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in ipm_t$, we can deduce $\sigma'(id_t)(''rt'')[i] = \sigma'(id_{ji}) = \sigma'(id_p)(''rtt'')[j]$.
Rewriting the goal with $\sigma'(id_t)(''rt'')[i] = \sigma'(id_{ji}) = \sigma'(id_p)(''rtt'')[j]$, $\boxed{\sigma'(id_p)(''rtt'')[j] = \texttt{true}.}$
By property of the $\texttt{Inject}_{\uparrow}$, the $\mathcal{H}$-VHDL rising edge and the stabilize relations, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `reinit_transitions_time_evaluation` process defined in the place design architecture, we can deduce:

$$\begin{aligned}\sigma'(id_p)(''rtt'')[j] =& \big((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test}) \\ & .(\sigma(id_p)(''sm'') - \sigma(id_p)(''sots'') < \sigma(id_p)(''oaw'')[j]) \\ & .(\sigma(id_p)(''sots'') > 0)\big) \\ & + \sigma(id_p)(''otf'')[j]\end{aligned} \tag{A.12}$$

Rewriting the goal with (A.12),

$$\boxed{\begin{aligned}\texttt{true} =& ((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test}) \\ & .(\sigma(id_p)(''sm'') - \sigma(id_p)(''sots'') < \sigma(id_p)(''oaw'')[j]) \\ & .(\sigma(id_p)(''sots'') > 0)) \\ & + (\sigma(id_p)(''otf'')[j])\end{aligned}}$$

By construction, there exists $id_{ft} \in Sigs(\Delta)$ s.t. $<\texttt{output\_transitions\_fired(j)} \Rightarrow \texttt{id}_{\texttt{ft}}> \in ipm_p$ and $<\texttt{fired} \Rightarrow \texttt{id}_{\texttt{ft}}> \in opm_t$. By property of state σ as being a stable state, we can deduce $\sigma(id_t)(''fired'') = \sigma(id_{ft}) = \sigma(id_p)(''otf'')[j]$.
Rewriting the goal with $\sigma(id_t)(''fired'') = \sigma(id_{ft}) = \sigma(id_p)(''otf'')[j]$,

$$\boxed{\begin{aligned}\texttt{true} =& ((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test}) \\ & .(\sigma(id_p)(''sm'') - \sigma(id_p)(''sots'') < \sigma(id_p)(''oaw'')[j]) \\ & .(\sigma(id_p)(''sots'') > 0)) \\ & + \sigma(id_t)(''fired'')\end{aligned}}$$

By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $t \in Fired(s) \Leftrightarrow \sigma(id_t)(''fired'') = \texttt{true}$.

Rewriting the goal with $t \in Fired(s) \Leftrightarrow \sigma(id_t)(''fired'') = \texttt{true}$ and simplify the goal, then tautology .

– **CASE** $t \notin Fired(s)$: Then, there are two cases that will determine the value of $s'.reset_t(t)$. Either there exists a place p with an output token sum greater than zero, that is connected to t by an `basic` or `test` arc, and such that the transient marking of p disables t; or such a place does not exist (the predicate is decidable).

∗ **CASE** there exists such a place p as described above:
Then, let us take such a place p and $\omega \in \mathbb{N}^*$ s.t.:

1. $\sum_{t_i \in Fired(s)} pre(p, t_i) > 0$
2. $pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})$
3. $s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) < \omega$

We will only consider the case where $pre(p,t) = (\omega, \texttt{basic})$; the proof is the similar when $pre(p,t) = (\omega, \texttt{test})$.

Assuming that p exists, and by property of $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$ (Rule **??**), we can deduce $s'.reset_t(t) = \texttt{true}$.

Rewriting the goal with $s'.reset_t(t) = \texttt{true}$, $\boxed{\sum_{i=0}^{|input(t)|-1} \sigma'(id_t)(''rt'')[i] = \texttt{true}.}$

To prove the goal, let us show $\boxed{\exists i \in [0, |input(t)| - 1] \text{ s.t. } \sigma'(id_t)(''rt'')[i] = \texttt{true}.}$
By construction, there exists $id_p \in Comps(\Delta)$ s.t. $\gamma(p) = id_p$.
By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.
By construction, there exist an $i \in [0, |input(t)| - 1]$, a $j \in [0, |output(p)| - 1]$ and $id_{ji} \in Sigs(\Delta)$ s.t. $<\texttt{reinit\_transition\_time(j)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in opm_p$ and $<\texttt{reinit\_time(i)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in ipm_t$. Let us take such an i, j and id_{ji}, and let us use i to prove the goal: $\boxed{\sigma'(id_t)(''rt'')[i] = \texttt{true}.}$
By property of the stabilize relation, $<\texttt{reinit\_transition\_time(j)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in opm_p$ and $<\texttt{reinit\_time(i)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in ipm_t$, we can deduce $\sigma'(id_t)(''rt'')[i] = \sigma'(id_{ji}) = \sigma'(id_p)(''rtt'')[j]$.
Rewriting the goal with $\sigma'(id_t)(''rt'')[i] = \sigma'(id_{ji}) = \sigma'(id_p)(''rtt'')[j]$, $\boxed{\sigma'(id_p)(''rtt'')[j] = \texttt{true}.}$
By property of the $\texttt{Inject}_{\uparrow}$, the $\mathcal{H}$-VHDL rising edge and the stabilize relation, and through the examination of the `reinit_transitions_time_evaluation` process defined in the place design architecture, we can deduce:

$$\begin{aligned}\sigma'(id_p)(''rtt'')[j] = &\big((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test})\\ &.(\sigma(id_p)(''sm'') - \sigma(id_p)(''sots'') < \sigma(id_p)(''oaw'')[j])\\ &.(\sigma(id_p)(''sots'') > 0)\big)\\ &+ \sigma(id_p)(''otf'')[j]\end{aligned} \tag{A.13}$$

Rewriting the goal with (A.13),

$$\boxed{\begin{array}{l}\texttt{true} = ((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test}) \\ \quad .(\sigma(id_p)(''sm'') - \sigma(id_p)(''sots'') < \sigma(id_p)(''oaw'')[j]) \\ \quad .(\sigma(id_p)(''sots'') > 0)) \\ \quad + \sigma(id_p)(''otf'')[j]\end{array}}$$

By construction, $<\texttt{output\_arcs\_types(j)} \Rightarrow \texttt{basic}> \in ipm_p$ and $<\texttt{output\_arcs\_weights(j)} \Rightarrow \omega> \in ipm_p$.
By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma'(id_p)(''oat'')[j] = \texttt{basic}$ and $\sigma'(id_p)(''oaw'')[j] = \omega$.
By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $\sigma(id_p)(''sm'') = s.M(p)$ and $\sigma(id_p)(''sots'') = \sum_{t_i \in Fired(s)} pre(p, t_i)$.
Rewriting the goal with $\sigma'(id_p)(''oat'')[j] = \texttt{basic}, \sigma'(id_p)(''oaw'')[j] = \omega, \sigma(id_p)(''sm'') = s.M(p)$ and $\sigma(id_p)(''sots'') = \sum_{t_i \in Fired(s)} pre(p, t_i)$, and simplifying the goal:

$$\boxed{\left((s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) < \omega) \,.\, (\sum_{t_i \in Fired(s)} pre(p, t_i) > 0)\right) + \sigma(id_t)(''fired'') = \texttt{true}}$$

We assumed that $s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) < \omega$ and $\sum_{t_i \in Fired(s)} pre(p, t_i) > 0$. Thus, by assumption:

$$\left((s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) < \omega) \,.\, (\sum_{t_i \in Fired(s)} pre(p, t_i) > 0)\right) + \sigma(id_t)(''fired'') = \texttt{true}$$

∗ **CASE** such a place does not exist:
Then, let us assume that, for all place $p \in P$

1. $\sum_{t_i \in Fired(s)} pre(p, t_i) = 0$
2. or $\forall \omega \in \mathbb{N}^*,\ pre(p, t) = (\omega, \texttt{basic}) \vee pre(p, t) = (\omega, \texttt{test}) \Rightarrow s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) \geq \omega$.

In that case, by property of $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$ (Rule **??**), we can deduce $s'.reset_t(t) = \texttt{false}$.
Rewriting the goal with $s'.reset_t(t) = \texttt{false}$: $\boxed{\sum_{i=0}^{|input(t)|-1} \sigma'(id_t)(''rt'')[i] = \texttt{false}.}$
To prove the goal, let us show $\boxed{\forall i \in [0, |input(t)| - 1],\ \sigma'(id_t)(''rt'')[i] = \texttt{false}.}$
Given an $i \in [0, |input(t)| - 1]$, let us show $\boxed{\sigma'(id_t)(''rt'')[i] = \texttt{false}.}$
By construction, there exist a $p \in input(t)$, an $id_p \in Comps(\Delta)$, gm_p, ipm_p, opm_p, a $j \in [0, |output(p)| - 1]$, an $id_{ji} \in Sigs(\Delta)$ s.t. $\gamma(p) = id_p$ and $\texttt{comp}(id_p, \texttt{place}, g_p,$

i_p, o_p) $\in$ *d.cs* and <`reinit_transition_time(j)` $\Rightarrow$ $\texttt{id}_{\texttt{ji}}$> $\in$ opm_p and <`reinit_time(i)` $\Rightarrow$ $\texttt{id}_{\texttt{ji}}$> $\in$ ipm_t. Let us take such a p, id_p, gm_p, ipm_p, opm_p, j and id_{ji}.
By property of the stabilize relation, <`reinit_transition_time(j)` $\Rightarrow$ $\texttt{id}_{\texttt{ji}}$> $\in$ opm_p and <`reinit_time(i)` $\Rightarrow$ $\texttt{id}_{\texttt{ji}}$> $\in$ ipm_t, we can deduce $\sigma'(id_t)(''rt'')[i] = \sigma'(id_{ji}) = \sigma'(id_p)(''rtt'')[j]$.
Rewriting the goal with $\sigma'(id_t)(''rt'')[i] = \sigma'(id_{ji}) = \sigma'(id_p)(''rtt'')[j]$: $\boxed{\sigma'(id_p)(''rtt'')[j] = \texttt{false}.}$
By property of the $\texttt{Inject}_{\uparrow}$, the $\mathcal{H}$-VHDL rising edge and the stabilize relations, `comp`(id_p, `place`, g_p, i_p, o_p) $\in$ *d.cs*, and through the examination of the `reinit_transitions_time_evaluation` process defined in the place design architecture, we can deduce:

$$\begin{aligned}\sigma'(id_p)(''rtt'')[j] = &\big((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test})\\ &.(\sigma(id_p)(''sm'') - \sigma(id_p)(''sots'') < \sigma(id_p)(''oaw'')[j])\\ &.(\sigma(id_p)(''sots'') > 0)\big)\\ &+ \sigma(id_p)(''otf'')[j]\end{aligned} \tag{A.14}$$

Rewriting the goal with (A.14),

$$\boxed{\begin{aligned}\texttt{false} = &\big((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test})\\ &.(\sigma(id_p)(''sm'') - \sigma(id_p)(''sots'') < \sigma(id_p)(''oaw'')[j])\\ &.(\sigma(id_p)(''sots'') > 0)\big)\\ &+ \sigma(id_p)(''otf'')[j])\end{aligned}}$$

By construction, there exists $id_{ft} \in Sigs(\Delta)$ s.t. <`output_transitions_fired(j)` $\Rightarrow$ $\texttt{id}_{\texttt{ft}}$> $\in$ ipm_p and <`fired` $\Rightarrow$ $\texttt{id}_{\texttt{ft}}$> $\in$ opm_t. By property of state σ as being a stable state, we can deduce $\sigma(id_t)(''fired'') = \sigma(id_{ft}) = \sigma(id_p)(''otf'')[j]$.
Rewriting the goal with $\sigma(id_t)(''fired'') = \sigma(id_{ft}) = \sigma(id_p)(''otf'')[j]$:

$$\boxed{\begin{aligned}\texttt{false} = &\big((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test})\\ &.(\sigma(id_p)(''sm'') - \sigma(id_p)(''sots'') < \sigma(id_p)(''oaw'')[j])\\ &.(\sigma(id_p)(''sots'') > 0)\big)\\ &+ \sigma(id_t)(''fired'')\end{aligned}}$$

By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $t \notin Fired(s) \Leftrightarrow \sigma(id_t)(''fired'') = \texttt{false}$
Rewriting the goal with $t \notin Fired(s) \Leftrightarrow \sigma(id_t)(''fired'') = \texttt{false}$ and simplifying the goal:

$$\boxed{\begin{aligned}\texttt{false} = &\big((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test})\\ &.(\sigma(id_p)(''sm'') - \sigma(id_p)(''sots'') < \sigma(id_p)(''oaw'')[j])\\ &.(\sigma(id_p)(''sots'') > 0)\big)\end{aligned}}$$

Then, based on the assumptions made at the beginning of case, there are two cases:

1. **CASE** $\sum_{t_i \in Fired(s)} pre(p, t_i) = 0$:

   By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $\sum_{t_i \in Fired(s)} pre(p, t_i) = \sigma(id_p)(''sots'')$.

   Rewriting the goal with $\sum_{t_i \in Fired(s)} pre(p, t_i) = \sigma(id_p)(''sots'')$ and $\sum_{t_i \in Fired(s)} pre(p, t_i) = 0$, and simplifying the goal: tautology.

2. **CASE** $\forall \omega \in \mathbb{N}^*,\ pre(p, t) = (\omega, \texttt{basic}) \vee pre(p, t) = (\omega, \texttt{test}) \Rightarrow s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) \geq \omega$:

   Let us perform case analysis on $pre(p, t)$; there are two cases:

   (a) **CASE** $pre(p, t) = (\omega, \texttt{basic})$ or $pre(p, t) = (\omega, \texttt{basic})$:

   By construction, $<\texttt{output\_arcs\_weights(j)} \Rightarrow \omega> \in ipm_p$.

   By property of stable state σ and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma(id_p)(''oaw'')[j] = \omega$.

   By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $\sigma(id_p)(''sm'') = s.M(p)$ and $\sigma(id_p)(''sots'') = \sum_{t_i \in Fired(s)} pre(p, t_i)$.

   Rewriting the goal with $\sigma(id_p)(''oaw'')[j] = \omega$, $\sigma(id_p)(''sm'') = s.M(p)$ and $\sigma(id_p)(''sots'') = \sum_{t_i \in Fired(s)} pre(p, t_i)$:

   $$\begin{aligned}\texttt{false} = &((\sigma(id_p)(''oat'')[j] = \texttt{basic} + \sigma(id_p)(''oat'')[j] = \texttt{test}) \\ &.(s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) < \omega) \\ &.(\sum_{t_i \in Fired(s)} pre(p, t_i) > 0))\end{aligned}$$

   We assumed that $s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) \geq \omega$, and then we can deduce $s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) < \omega = \texttt{false}$.

   Rewriting the goal with $s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) < \omega = \texttt{false}$, and simplifying the goal, tautology.

   (b) **CASE** $pre(p, t) = (\omega, \texttt{inhib})$:

   By construction, $<\texttt{output\_arcs\_types(j)} \Rightarrow \texttt{inhib}> \in ipm_p$.

   By property of stable state σ and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma(id_p)(''oat'')[j] = \texttt{inhib}$.

   Rewriting the goal with $\sigma(id_p)(''oat'')[j] = \texttt{inhib}$, and simplifying the goal, tautology.

□

A.3.5 Rising edge and action executions

Lemma 28 (Rising edge equal action executions). *For all sitpn, d, γ, E_c, E_p, τ, Δ, σ_e, s, s', σ, σ_i, $\sigma_\uparrow$, σ' that verify the hypotheses of Def. 16, then*
$\forall a \in \mathcal{A}, id_a \in Outs(\Delta)\ s.t.\ \gamma(a) = id_a,\ s'.ex(a) = \sigma'(id_a)$.

Proof. Given an $a \in \mathcal{A}$ and an $id_a \in Outs(\Delta)$ *s.t.* $\gamma(a) = id_a$, let us show $\boxed{s'.ex(a) = \sigma'(id_a).}$

By property of $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$, we can deduce $s.ex(a) = s'.ex(a)$.
By construction, id_a is an output port identifier of Boolean type in the $\mathcal{H}$-VHDL design d. The generated ''`action`'' process is responsible for the assignment of the *id a* only during the initialization phase or during a falling edge phase.
By property of the $\mathcal{H}$-VHDL $\texttt{Inject}_\uparrow$, rising edge, stabilize relations, and the ''`action`'' process, we can deduce $\sigma(id_a) = \sigma'(id_a)$.
Rewriting the goal with $s.ex(a) = s'.ex(a)$ and $\sigma(id_a) = \sigma'(id_a)$, $\boxed{s.ex(a) = \sigma(id_a).}$
By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, $s.ex(a) = \sigma(id_a)$. □

A.3.6 Rising edge and function executions

Lemma 29 (Rising edge equal function executions). *For all sitpn, d, γ, E_c, E_p, τ, Δ, σ_e, s, s', σ, σ_i, $\sigma_\uparrow$, σ' that verify the hypotheses of Def. 16, then*
$\forall f \in \mathcal{F}, id_f \in Outs(\Delta)\ s.t.\ \gamma(f) = id_f,\ s'.ex(f) = \sigma'(id_f)$.

Proof. Given an $f \in \mathcal{F}$ and an $id_f \in Outs(\Delta)$ s.t. $\gamma(f) = id_f$, let us show $\boxed{s'.ex(f) = \sigma'(id_f).}$

By property of $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$ (Rule **??**):

$$s'.ex(f) = \sum_{t \in Fired(s)} \mathbb{F}(t, f) \tag{A.15}$$

By construction, id_f is an output port identifier of Boolean type in the $\mathcal{H}$-VHDL design d. The generated `function` process assigns a value to the output port id_f only during the initialization phase or during a rising edge phase.
By construction, the `function` process is defined in the behavior of design d, i.e.
$\texttt{ps}(''function'', \emptyset, sl, ss) \in d.cs$.
Let $trs(f)$ be the set of transitions associated to function f, i.e $trs(f) = \{t \in T \mid \mathbb{F}(t, f) = \texttt{true}\}$.
Let us perform case analysis on $trs(f)$; there are two cases:

- **CASE** $trs(f) = \emptyset$:

  By construction, $\texttt{id}_\texttt{f} \Leftarrow \texttt{false} \in ss_\uparrow$ where $ss_\uparrow$ is the part of the `function` process body executed during a rising edge phase.

  By property of the $\mathcal{H}$-VHDL rising edge, the stabilize relations and $\texttt{ps}(''function'', \emptyset, sl, ss) \in d.cs$, we can deduce $\sigma'(id_f) = \texttt{false}$.

By property of $\sum_{t \in Fired(s)} \mathbb{F}(t,f)$ and $trs(f) = \emptyset$, we can deduce $\sum_{t \in Fired(s)} \mathbb{F}(t,f) = \texttt{false}$.

Rewriting the goal with (A.15), $\sigma'(id_f) = \texttt{false}$ and $\sum_{t \in Fired(s)} \mathbb{F}(t,f) = \texttt{false}$: tautology.

- **CASE** $trs(f) \neq \emptyset$:

  By construction, $\texttt{id}_\texttt{f} \Leftarrow \texttt{id}_{\texttt{ft}_0} + \cdots + \texttt{id}_{\texttt{ft}_n} \in ss_\uparrow$, where $id_{ft_i} \in Sigs(\Delta)$, $ss_\uparrow$ is the part of the `function` process body executed during a rising edge phase, and $n = |trs(f)| - 1$.

  By property of the $\texttt{Inject}_\uparrow$, the $\mathcal{H}$-VHDL rising edge, the stabilize relations, and $\texttt{ps}(\text{"}function\text{"}, \emptyset, sl, ss) \in d.cs$, we can deduce:

  $$\sigma'(id_f) = \sigma(id_{ft_0}) + \cdots + \sigma(id_{ft_n}) \tag{A.16}$$

  Rewriting the goal with (A.15) and (A.16), $\boxed{\sum_{t \in Fired(s)} \mathbb{F}(t,f) = \sigma(id_{ft_0}) + \cdots + \sigma(id_{ft_n}).}$

  Let us reason on the value of $\sigma(id_{ft_0}) + \cdots + \sigma(id_{ft_n})$; there are two cases:

  – **CASE** $\sigma(id_{ft_0}) + \cdots + \sigma(id_{ft_n}) = \texttt{true}$:

    Then, we can rewrite the goal as follows: $\boxed{\sum_{t \in Fired(s)} \mathbb{F}(t,f) = \texttt{true}.}$

    To prove the above goal, let us show $\boxed{\exists t \in Fired(s)\ s.t.\ \mathbb{F}(t,f) = \texttt{true}.}$

    From $\sigma(id_{ft_0}) + \cdots + \sigma(id_{ft_n}) = \texttt{true}$, we can deduce $\exists id_{ft_i}\ s.t.\ \sigma(id_{ft_i}) = \texttt{true}$. Let us take such an id_{ft_i}.

    By construction, there exist a $t \in trs(f)$, an $id_t \in Comps(\Delta)$, gm_t, ipm_t, opm_t such that:

    * $\gamma(t) = id_t$
    * $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$
    * $<\texttt{fired} \Rightarrow \texttt{id}_{\texttt{ft}_\texttt{i}}> \in opm_t$

    By property of σ as being a stable design state, and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, we can deduce $\sigma(id_t)(\text{"}fired\text{"}) = \sigma(id_{ft_i})$, and thus that $\sigma(id_t)(\text{"}fired\text{"}) = \texttt{true}$.

    By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $t \in Fired(s)$.

    Let us use t to prove the goal: $\boxed{\mathbb{F}(t,f) = \texttt{true}.}$

    By definition of $t \in trs(f)$, $\mathbb{F}(t,f) = \texttt{true}$.

  – **CASE** $\sigma(id_{ft_0}) + \cdots + \sigma(id_{ft_n}) = \texttt{false}$:

    Then, we can rewrite the goal as follows: $\boxed{\sum_{t \in Fired(s)} \mathbb{F}(t,f) = \texttt{false}.}$

    To prove the above goal, let us show $\boxed{\forall t \in Fired(s)\ s.t.\ \mathbb{F}(t,f) = \texttt{false}.}$

    Given a $t \in Fired(s)$, let us show $\boxed{\mathbb{F}(t,f) = \texttt{false}.}$

    Let us perform case analysis on $\mathbb{F}(t,f)$; there are 2 cases:

* **CASE** $\mathbb{F}(t,f) = \texttt{false}$.
* **CASE** $\mathbb{F}(t,f) = \texttt{true}$:

  By construction, there exist an $id_t \in Comps(\Delta)$, gm_t, ipm_t, opm_t and $id_{ft_i} \in Sigs(\Delta)$ such that:
  - $\gamma(t) = id_t$
  - $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$
  - $<\texttt{fired} \Rightarrow \texttt{id}_{\texttt{ft}_\texttt{i}}> \in opm_t$

  By property of stable design state σ and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, we can deduce $\sigma(id_t)(''fired'') = \sigma(id_{ft_i})$.

  By property of $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$, we can deduce $t \in Fired(s) \Leftrightarrow \sigma(id_t)(''fired'') = \texttt{true}$.
  Since $t \in Fired(s)$, we can deduce $\sigma(id_t)(''fired'') = \texttt{true}$, and from $\sigma(id_t)(''fired'') = \sigma(id_{ft_i})$, we can deduce $\sigma(id_{ft_i}) = \texttt{true}$.
  Then, $\sigma(id_{ft_i}) = \texttt{true}$ contradicts $\sigma(id_{ft_0}) + \cdots + \sigma(id_{ft_n}) = \texttt{false}$.

□

A.3.7 Rising edge and sensitization

Lemma 30 (Rising edge equal sensitized)**.** *For all sitpn, d, γ, E_c, E_p, τ, Δ, σ_e, s, s', σ, σ_i, $\sigma_\uparrow$, σ' that verify the hypotheses of Def. 16, then*
$\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t$, $t \in Sens(s'.M) \Leftrightarrow \sigma'(id_t)(''s\_enabled'') = \texttt{true}$.

Proof. Given a $t \in T$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, let us show
$\boxed{t \in Sens(s'.M) \Leftrightarrow \sigma'(id_t)(''s\_enabled'') = \texttt{true}.}$

By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.. Then, the proof is in two parts:

1. Assuming that $t \in Sens(s'.M)$, let us show $\boxed{\sigma'(id_t)(''s\_enabled'') = \texttt{true}.}$

   By property of the stabilize relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `enable_evaluation` process defined in the transition design architecture:

   $$\sigma'(id_t)(''se'') = \prod_{i=0}^{\Delta(id_t)(''ian'')-1} \sigma'(id_t)(''input\_arcs\_valid'')[i] \tag{A.17}$$

   Rewriting the goal with (A.17), $\boxed{\prod_{i=0}^{\Delta(id_t)(''ian'')-1} \sigma'(id_t)(''iav'')[i] = \texttt{true}.}$

   To prove the goal, let us show that $\boxed{\forall i \in [0, \Delta(id_t)(''ian'') - 1],\ \sigma'(id_t)(''iav'')[i] = \texttt{true}.}$

   Given an $i \in [0, \Delta(id_t)(''ian'') - 1]$, let us show $\boxed{\sigma'(id_t)(''iav'')[i] = \texttt{true}.}$

Let us perform case analysis on $input(t)$.

- **CASE** $input(t) = \emptyset$:
  By construction, $<$`input_arcs_number` $\Rightarrow$ `1`$>$ $\in gm_t$ and
  $<$`input_arcs_valid(0)` $\Rightarrow$ `true`$>$ $\in ipm_t$.
  By property of the elaboration and stabilize relations and `comp`$(id_t,$ `transition`$, g_t, i_t, o_t) \in d.cs$, we can deduce $\Delta(id_t)(''ian'') = 1$ and $\sigma'(id_t)(''iav''))[0] =$ `true`.
  Thanks to $\Delta(id_t)(''ian'') = 1$, we can deduce that $i = 0$.
  Rewriting the goal with $\sigma'(id_t)(''iav''))[0] =$ `true`, tautology.
- **CASE** $input(t) \neq \emptyset$:
  By construction, $<$`input_arcs_number` $\Rightarrow |input(t)|>$ $\in gm_t$.
  By property of the elaboration relation and `comp`$(id_t,$ `transition`$, g_t, i_t, o_t) \in d.cs$, we can deduce $\Delta(id_t)(''ian'') = |input(t)|$.
  Thanks to $\Delta(id_t)(''ian'') = |input(t)|$, we know that $i \in [0, |input(t)| - 1]$.
  By construction, there exist a $p \in input(t)$, $id_p \in Comps(\Delta)$, gm_p, ipm_p, opm_p, $j \in [0, |output(p)| - 1]$ and $id_{ji} \in Sigs(\Delta)$ s.t. $\gamma(p) = id_p$ and
  `comp`$(id_p,$ `place`$, g_p, i_p, o_p) \in d.cs$ and $<$`output_arcs_valid(j)` $\Rightarrow$ `id`$_{\texttt{ji}}>$ $\in opm_p$ and $<$`input_arcs_valid(i)` $\Rightarrow$ `id`$_{\texttt{ji}}>$ $\in ipm_t$.
  By property of the stabilize relation, `comp`$(id_t,$ `transition`$, g_t, i_t, o_t) \in d.cs$ and `comp`$(id_p,$ `place`$, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma'(id_t)(''iav''))[i] = \sigma'(id_{ji}) = \sigma'(id_p)(''oav''))[j]$.
  Rewriting the goal with $\sigma'(id_t)(''iav''))[i] = \sigma'(id_{ji}) = \sigma'(id_p)(''oav''))[j]$:
  $\boxed{\sigma'(id_p)(''oav''))[j] = \texttt{true}.}$
  By property of the stabilize relation, `comp`$(id_p,$ `place`$, g_p, i_p, o_p) \in d.cs$, and through the examination of the `marking_validation_evaluation` process defined in the place design architecture, we can deduce:

$$\begin{aligned}\sigma'(id_p)(''oav''))[j] = &\big((\sigma'(id_p)(''oat''))[j] = \texttt{basic} + \sigma'(id_p)(''oat''))[j] = \texttt{test}\big)\\ &\quad . \sigma'(id_p)(''sm'') \geq \sigma'(id_p)(''oaw''))[j]\big)\\ &+ \big(\sigma'(id_p)(''oat''))[j] = \texttt{inhib} \,.\, \sigma'(id_p)(''sm'') < \sigma'(id_p)(''oaw''))[j]\big)\end{aligned} \tag{A.18}$$

  Rewriting the goal with (A.18),

$$\boxed{\begin{aligned}\texttt{true} = &\big((\sigma'(id_p)(''oat''))[j] = \texttt{basic} + \sigma'(id_p)(''oat''))[j] = \texttt{test}\big)\\ &\quad . \sigma'(id_p)(''sm'') \geq \sigma'(id_p)(''oaw''))[j]\big)\\ &+ \big(\sigma'(id_p)(''oat''))[j] = \texttt{inhib} \,.\, \sigma'(id_p)(''sm'') < \sigma'(id_p)(''oaw''))[j]\big)\end{aligned}}$$

  Let us perform case analysis on $pre(p,t)$; there are 3 cases:
  – **CASE** $pre(p,t) = (\omega,$ `basic`$)$:

By construction, $<\texttt{output\_arcs\_types(j)} \Rightarrow \texttt{basic}> \in ipm_p$ and $<\texttt{output\_arcs\_weights(j)} \Rightarrow \omega> \in ipm_p$.
By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma'(id_p)(\text{"}oat\text{"})[j] = \texttt{basic}$ and $\sigma'(id_p)(\text{"}oaw\text{"})[j] = \omega$.
Rewriting the goal with $\sigma'(id_p)(\text{"}oat\text{"})[j] = \texttt{basic}$ and $\sigma'(id_p)(\text{"}oaw\text{"})[j] = \omega$, and simplifying the goal:
$\boxed{\sigma'(id_p)(\text{"}sm\text{"}) \geq \omega = \texttt{true}.}$
Appealing to Lemma 24, we can deduce $s'.M(p) = \sigma'(id_p)(\text{"}sm\text{"})$.
Rewriting the goal with $s'.M(p) = \sigma'(id_p)(\text{"}sm\text{"})$: $\boxed{s'.M(p) \geq \omega = \texttt{true}.}$
By definition of $t \in Sens(s'.M)$, $s'.M(p) \geq \omega = \texttt{true}.$

- **CASE** $pre(p,t) = (\omega, \texttt{test})$: same as above.
- **CASE** $pre(p,t) = (\omega, \texttt{inhib})$:
  By construction, $<\texttt{output\_arcs\_types(j)} \Rightarrow \texttt{inhib}> \in ipm_p$ and $<\texttt{output\_arcs\_weights(j)} \Rightarrow \omega> \in ipm_p$.
  By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma'(id_p)(\text{"}oat\text{"})[j] = \texttt{inhib}$ and $\sigma'(id_p)(\text{"}oaw\text{"})[j] = \omega$.
  Rewriting the goal with $\sigma'(id_p)(\text{"}oat\text{"})[j] = \texttt{inhib}$ and $\sigma'(id_p)(\text{"}oaw\text{"})[j] = \omega$, and simplifying the goal: $\boxed{\sigma'(id_p)(\text{"}sm\text{"}) < \omega = \texttt{true}.}$
  Appealing to Lemma 24, we can deduce $s'.M(p) = \sigma'(id_p)(\text{"}sm\text{"})$.
  Rewriting the goal with $s'.M(p) = \sigma'(id_p)(\text{"}sm\text{"})$: $\boxed{s'.M(p) < \omega = \texttt{true}.}$
  By definition of $t \in Sens(s'.M)$, $s'.M(p) < \omega = \texttt{true}.$

2. Assuming that $\sigma'(id_t)(\text{"}s\_enabled\text{"}) = \texttt{true}$, let us show $\boxed{t \in Sens(s'.M).}$

   By definition of $t \in Sens(s'.M)$, let us show

   $$\boxed{\forall p \in P, \omega \in \mathbb{N}^*,\ \big(pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test}) \Rightarrow s'.M(p) \geq \omega\big) \wedge \big(pre(p,t) = (\omega, \texttt{inhib}) \Rightarrow s'.M(p) < \omega\big)}$$

   Given a $p \in P$ and an $\omega \in \mathbb{N}^*$, let us show
   $\boxed{pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test}) \Rightarrow s'.M(p) \geq \omega}$ and
   $\boxed{pre(p,t) = (\omega, \texttt{inhib}) \Rightarrow s'.M(p) < \omega.}$

   (a) Assuming $pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})$, let us show $\boxed{s'.M(p) \geq \omega.}$

   The proceeding is the same for $pre(p,t) = (\omega, \texttt{basic})$ and $pre(p,t) = (\omega, \texttt{test})$. Therefore, we will only cover the case where $pre(p,t) = (\omega, \texttt{basic})$.

   By property of the stabilize relation and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, equation (A.17) holds.

Rewriting $\sigma'(id_t)("se") = \texttt{true}$ with (A.17), we can deduce:
$\prod_{i=0}^{\Delta(id_t)("ian")-1} \sigma'(id_t)("iav")[i] = \texttt{true}$.
Then, we can deduce that $\forall i \in [0, \Delta(id_t)("ian") - 1]$, $\sigma'(id_t)("iav")[i] = \texttt{true}$.
By construction, there exist an $id_p \in Comps(\Delta)$, gm_p, ipm_p, opm_p, $i \in [0, |input(t)| - 1]$, $j \in [0, |output(p)| - 1]$ and $id_{ji} \in Sigs(\Delta)$ s.t. $\gamma(p) = id_p$ and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$ and $<\texttt{output\_arcs\_valid(j)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in opm_p$ and $<\texttt{input\_arcs\_valid(i)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in ipm_t$. Let us take such an $id_p \in Comps(\Delta)$, gm_p, ipm_p, opm_p, $i \in [0, |input(t)| - 1]$, $j \in [0, |output(p)| - 1]$ and $id_{ji} \in Sigs(\Delta)$.
By construction, $<\texttt{input\_arcs\_number} \Rightarrow |input(t)|> \in gm_t$.
By property of the elaboration relation and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, we can deduce $\Delta(id_t)("ian") = |input(t)|$.
Thanks to $\Delta(id_t)("ian") = |input(t)|$, we can deduce that $\forall i \in [0, |input(t)| - 1]$, $\sigma'(id_t)("iav")[i] = \texttt{true}$.
Having such an $i \in [0, |input(t)| - 1]$, we can deduce that $\sigma'(id_t)("iav")[i] = \texttt{true}$.
By property of the stabilize relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$ and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma'(id_t)("iav")[i] = \sigma'(id_{ji}) = \sigma'(id_p)("oav")[j]$.
Thanks to $\sigma'(id_t)("iav")[i] = \sigma'(id_{ji}) = \sigma'(id_p)("oav")[j]$, we can deduce that $\sigma'(id_p)("oav")[j] = \texttt{true}$.
By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, equation (A.18) holds. Thanks to (A.18), we can deduce that:

$$\begin{aligned}\texttt{true} = &\big((\sigma'(id_p)("oat")[j] = \texttt{basic} + \sigma'(id_p)("oat")[j] = \texttt{test})\\ &\quad . \sigma'(id_p)("sm") \geq \sigma'(id_p)("oaw")[j]\big)\\ &+ \big(\sigma'(id_p)("oat")[j] = \texttt{inhib} \,.\, \sigma'(id_p)("sm") < \sigma'(id_p)("oaw")[j]\big)\end{aligned} \tag{A.19}$$

By construction, $<\texttt{output\_arcs\_types(j)} \Rightarrow \texttt{basic}> \in ipm_p$ and $<\texttt{output\_arcs\_weights(j)} \Rightarrow \omega> \in ipm_p$.
By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma'(id_p)("oat")[j] = \texttt{basic}$ and $\sigma'(id_p)("oaw")[j] = \omega$.
Thanks to $\sigma'(id_p)("oat")[j] = \texttt{basic}$, $\sigma'(id_p)("oaw")[j] = \omega$, and simplifying Equation (A.19), we can deduce $\sigma'(id_p)("sm") \geq \omega = \texttt{true}$.

Appealing to Lemma 24, $s'.M(p) \geq \omega$.

(b) Assuming $pre(p, t) = (\omega, \texttt{inhib})$, let us show $\boxed{s'.M(p) < \omega.}$

The proceeding is the same as in the preceding case. Here, we will start the proof where the two cases are diverging, i.e:
By construction, $<\texttt{output\_arcs\_types(j)} \Rightarrow \texttt{inhib}> \in ipm_p$ and $<\texttt{output\_arcs\_weights(j)} \Rightarrow \omega> \in ipm_p$.
By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma'(id_p)("oat")[j] = \texttt{inhib}$ and $\sigma'(id_p)("oaw")[j] = \omega$.

Thanks to $\sigma'(id_p)(\text{"}oat\text{"})[j] = \texttt{inhib}$ and $\sigma'(id_p)(\text{"}oaw\text{"})[j] = \omega$, and simplifying Equation (A.19), we can deduce $\sigma'(id_p)(\text{"}sm\text{"}) < \omega = \texttt{true}$.

Appealing to Lemma 24, $s'.M(p) < \omega$.

□

Lemma 31 (Rising edge equal not sensitized). *For all sitpn, d, γ, E_c, E_p, τ, Δ, σ_e, s, s', σ, σ_i, $\sigma_\uparrow$, σ' that verify the hypotheses of Def. 16, then*
$\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t, t \notin Sens(s'.M) \Leftrightarrow \sigma'(id_t)(\text{"}s\_enabled\text{"}) = \texttt{false}$.

Proof. Proving the above lemma is trivial by appealing to Lemma 30 and by reasoning on contrapositives. □

A.4 Falling Edge

A.4.1 Falling Edge and marking

Lemma 32 (Falling edge equal marking). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then* $\forall p \in P, id_p \in Comps(\Delta)$ *s.t.* $\gamma(p) = id_p$, $s'.M(p) = \sigma'(id_p)(\text{"}s\_marking\text{"})$.

Proof. Given a $p \in P$ and an $id \in Comps(\Delta)$ s.t. $\gamma(p) = id_p$, let us show
$s'.M(p) = \sigma'(id_p)(\text{"}s\_marking\text{"})$.
By definition of $E_c, \tau \vdash sitpn, s \xrightarrow{\downarrow} s'$, we can deduce $s.M(p) = s'.M(p)$.
By property of the $\texttt{Inject}_\downarrow$ relation, the $\mathcal{H}$-VHDL falling edge relation, the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `marking` process defined in the place design architecture, we can deduce $\sigma'(id_p)(\text{"}s\_marking\text{"}) = \sigma(id_p)(\text{"}s\_marking\text{"})$.
Rewriting the goal with $s.M(p) = s'.M(p)$ and $\sigma'(id_p)(\text{"}sm\text{"}) = \sigma(id_p)(\text{"}sm\text{"})$: $s.M(p) = \sigma(id_p)(\text{"}s\_ma$
By definition of $\gamma, E_c, \tau \vdash s \overset{\downarrow}{\approx} \sigma$: $s.M(p) = \sigma(id_p)(\text{"}s\_marking\text{"})$.

□

Lemma 33 (Falling Edge Equal Output Token Sum). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then* $\forall p, id_p$ *s.t.* $\gamma(p) = id_p$, $\sum_{t \in Fired(s')} pre(p,t) = \sigma'(id_p)(\text{"}s\_output\_token\_sum\text{"})$.

Proof. Given a $p \in P$ and an $id_p \in Comps(\Delta)$, let us show
$\sum_{t \in Fired(s')} pre(p,t) = \sigma'(id_p)(\text{"}s\_output\_token\_sum\text{"})$.
By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.

By property of the stabilize relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `output_tokens_sum` process defined in the place design architecture:

$$\sigma'(id_p)(''sots'') = \sum_{i=0}^{\Delta(id_p)(''oan'')-1} \begin{cases} \sigma'(id_p)(''oaw'')[i] \texttt{ if } (\sigma'(id_p)(''otf'')[i] \\ \qquad\qquad\qquad\qquad . \sigma'(id_p)(''oat'')[i] = \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases} \tag{A.20}$$

Rewriting the goal with (A.20):

$$\sum_{t \in Fired(s')} pre(p,t) = \sum_{i=0}^{\Delta(id_p)(''oan'')-1} \begin{cases} \sigma'(id_p)(''oaw'')[i] \texttt{ if } (\sigma'(id_p)(''otf'')[i] \\ \qquad\qquad\qquad\qquad . \sigma'(id_p)(''oat'')[i] = \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases}$$

Let us unfold the definition of the left sum term:

$$\sum_{t \in Fired(s')} \begin{cases} \omega \texttt{ if } pre(p,t) = (\omega, \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases} = \sum_{i=0}^{\Delta(id_p)(''oan'')-1} \begin{cases} \sigma'(id_p)(''oaw'')[i] \texttt{ if } (\sigma'(id_p)(''otf'')[i] \\ \qquad\qquad\qquad\qquad . \sigma'(id_p)(''oat'')[i] = \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases}$$

To ease the reading, let us define functions $f \in Fired(s') \to \mathbb{N}$ and $g \in [0, |output(p)| - 1] \to \mathbb{N}$

s.t. $f(t) = \begin{cases} \omega \texttt{ if } pre(p,t) = (\omega, \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases}$ and $g(i) = \begin{cases} \sigma'(id_p)(''oaw'')[i] \texttt{ if } (\sigma'(id_p)(''otf'')[i] \\ \qquad\qquad\qquad\qquad . \sigma'(id_p)(''oat'')[i] = \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases}$

Then, the goal is: $\sum_{t \in Fired(s')} f(t) = \sum_{i=0}^{\Delta(id_p)(''oan'')-1} g(i)$

Let us perform case analysis on $output(p)$; there are two cases:

- **CASE** $output(p) = \emptyset$:

  By construction, <`output_arcs_number`⇒ `1`> $\in gm_p$, <`output_arcs_types(0)`⇒ `basic`> $\in ipm_p$, <`output_transitions_fired(0)`⇒ `true`> $\in ipm_p$, and <`output_arcs_weights(0)`⇒ `0`> $\in ipm_p$.

  By property of the elaboration relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\Delta(id_p)(''oan'') = 1$.

  By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma'(id_p)(''oat'')[0] = \texttt{basic}$, $\sigma'(id_p)(''otf'')[0] = \texttt{true}$ and $\sigma'(id_p)(''oaw'')[0] = 0$.

  By property of $output(p) = \emptyset$, we can deduce

  $$\sum_{t \in Fired(s')} \begin{cases} \omega \texttt{ if } pre(p,t) = (\omega, \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases} = 0$$

Rewriting the goal with $\Delta(id_p)(''oan'') = 1$, $\sigma'(id_p)(''oat'')[0] = \texttt{basic}$, $\sigma'(id_p)(''otf'')[0] = \texttt{true}$, $\sigma'(id_p)(''oaw'')[0] = 0$ and $\sum_{t \in Fired(s')} \begin{cases} \omega \texttt{ if } pre(p,t) = (\omega, \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases} = 0$, tautology.

- **CASE** $output(p) \neq \emptyset$:

  By construction, $<\texttt{output\_arcs\_number} \Rightarrow |output(p)|> \in gm_p$, and by property of the elaboration relation, we can deduce $\Delta(id_p)(''oan'') = |output(p)|$.

  Rewriting the goal with $\Delta(id_p)(''oan'') = |output(p)|$: $\boxed{\sum_{t \in Fired(s')} f(t) = \sum_{i=0}^{|output(p)|-1} g(i).}$

  Let us reason by induction on the right sum term of the goal.

  – **BASE CASE**:

  In that case, $0 > |output| - 1$ and $\sum_{i=0}^{|output(p)|-1} g(i) = 0$.

  As $0 > |output| - 1$, then $|output(p)| = 0$, thus contradicting $output(p) \neq \emptyset$.

  – **INDUCTION CASE**:

  In that case, $0 \leq |output(p)| - 1$.

  $$\forall F \subseteq Fired(s'), g(0) + \sum_{t \in F} f(t) = g(0) + \sum_{i=1}^{|output(p)|-1} g(i)$$

  $$\boxed{\sum_{t \in Fired(s')} f(t) = g(0) + \sum_{i=1}^{|output(p)|-1} g(i)}$$

  By definition of g:

  $$g(0) = \begin{cases} \sigma'(id_p)(''oaw'')[0] \texttt{ if } (\sigma'(id_p)(''otf'')[0] \\ \qquad\qquad\qquad\qquad . \sigma'(id_p)(''oat'')[0] = \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases} \tag{A.21}$$

  Let us perform case analysis on the value of $\sigma'(id_p)(''otf'')[0] \,.\, \sigma'(id_p)(''oat'')[0] = \texttt{basic}$; there are two cases:

  1. $(\sigma'(id_p)(''otf'')[0] \,.\, \sigma'(id_p)(''oat'')[0] = \texttt{basic}) = \texttt{false}$:
     In that case, $g(0) = 0$, and then we can apply the induction hypothesis with $F = Fired(s')$ to solve the goal: $\sum_{t \in Fired(s')} f(t) = \sum_{i=1}^{|output(p)|-1} g(i).$

2. $(\sigma'(id_p)("otf")[0] \,.\, \sigma'(id_p)("oat")[0] = \texttt{basic}) = \texttt{true}$:
In that case, $g(0) = \sigma'(id_p)("oaw")[0], \sigma'(id_p)("otf")[0] = \texttt{true}$ and $\sigma'(id_p)("oat")[0] = \texttt{basic}$.
By construction, there exist a $t \in output(t)$, $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, and there exist gm_t, ipm_t, opm_t s.t. $\texttt{comp}(id_t, "transition", gm_t, ipm_t, opm_t) \in d.cs$, and there exist an $\omega \in \mathbb{N}^*$, an $a \in \{\texttt{basic}, \texttt{test}, \texttt{inhib}\}$ and an $id_{ft} \in Sigs(\Delta)$ such that:
   * $pre(p,t) = (\omega, a)$
   * $<\texttt{output\_arcs\_types}(0) \Rightarrow a> \in ipm_p$
   * $<\texttt{output\_arcs\_weights}(0) \Rightarrow \omega> \in ipm_p$
   * $<\texttt{fired} \Rightarrow id_{ft}> \in opm_t$
   * $<\texttt{output\_transitions\_fired}(0) \Rightarrow \texttt{id}_{\texttt{ft}}> \in ipm_p$

   By property of the stabilize relation, $\sigma'(id_p)("oat")[0] = \texttt{basic}$ and $<\texttt{output\_arcs\_types}(0) \Rightarrow \texttt{a}> \in ipm_p$, we can deduce $pre(p,t) = (\omega, \texttt{basic})$.
   By property of the stabilize relation, $<\texttt{fired} \Rightarrow \texttt{id}_{\texttt{ft}}> \in opm_t$, $<\texttt{output\_transitions\_fired}(0) \Rightarrow \texttt{id}_{\texttt{ft}}> \in ipm_p$ and $\sigma'(id_p)("otf")[0] = \texttt{true}$, we can deduce $\sigma'(id_t)("fired") = \texttt{true}$.
   Appealing to Lemma 4, and thanks to $\sigma'(id_t)("fired") = \texttt{true}$, we can deduce $t \in Fired(s')$.
   With $t \in Fired(s')$, we can rewrite the left sum term of the goal as follows:

$$\boxed{f(t) + \sum_{t' \in Fired(s') \setminus \{t\}} f(t') = g(0) + \sum_{i=1}^{|output(p)|-1} g(i)}$$

   We know that $g(0) = \sigma'(id_p)("oaw")[0]$, and by property of the stabilize relation and $<\texttt{output\_arcs\_weights}(0) \Rightarrow \omega> \in ipm_p$, we can deduce $\sigma'(id_p)("oaw")[0] = \omega$.
   Rewriting the goal with $\sigma'(id_p)("oaw")[0] = \omega$:

$$\boxed{f(t) + \sum_{t' \in Fired(s') \setminus \{t\}} f(t') = \omega + \sum_{i=1}^{|output(p)|-1} g(i)}$$

   By definition of f, and as $pre(p,t) = (\omega, \texttt{basic})$, then $f(t) = \omega$; thus, rewriting the goal:

$$\boxed{\omega + \sum_{t' \in Fired(s') \setminus \{t\}} f(t') = \omega + \sum_{i=1}^{|output(p)|-1} g(i)}$$

   Then, knowing that $g(0) = \omega$, we can apply the induction hypothesis with $F = Fired(s') \setminus \{t\}$: $g(0) + \sum_{t' \in Fired(s') \setminus \{t\}} f(t') = g(0) + \sum_{i=1}^{|output(p)|-1} g(i)$.

□

Lemma 34 (Falling Edge Equal Input Token Sum). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then $\forall p, id_p$ s.t. $\gamma(p) = id_p$, $\sum_{t \in Fired(s')} post(t,p) = \sigma'_p("s\_input\_token\_sum")$.*

Proof. Given a $p \in P$ and an $id_p \in Comps(\Delta)$, let us show

$$\boxed{\sum_{t \in Fired(s')} post(t,p) = \sigma'(id_p)(\text{"s\_input\_token\_sum"}).}$$

By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.
By property of the stabilize relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `input_tokens_sum` process defined in the place design architecture:

$$\sigma'(id_p)(\text{"sits"}) = \sum_{i=0}^{\Delta(id_p)(\text{"ian"})-1} \begin{cases} \sigma'(id_p)(\text{"iaw"})[i] \texttt{ if } \sigma'(id_p)(\text{"itf"})[i] \\ 0 \textit{ otherwise} \end{cases} \tag{A.22}$$

Rewriting the goal with (A.22):

$$\boxed{\sum_{t \in Fired(s')} post(t,p) = \sum_{i=0}^{\Delta(id_p)(\text{"ian"})-1} \begin{cases} \sigma'(id_p)(\text{"iaw"})[i] \texttt{ if } \sigma'(id_p)(\text{"otf"})[i] \\ 0 \textit{ otherwise} \end{cases}}$$

Let us unfold the definition of the left sum term:

$$\boxed{\begin{gathered} \sum_{t \in Fired(s')} \begin{cases} \omega \texttt{ if } post(t,p) = \omega \\ 0 \textit{ otherwise} \end{cases} \\ = \\ \sum_{i=0}^{\Delta(id_p)(\text{"ian"})-1} \begin{cases} \sigma'(id_p)(\text{"iaw"})[i] \texttt{ if } \sigma'(id_p)(\text{"itf"})[i] \\ 0 \textit{ otherwise} \end{cases} \end{gathered}}$$

Let us perform case analysis on $input(p)$; there are two cases:

- **CASE** $input(p) = \emptyset$:

  By construction, $<\texttt{input\_arcs\_number} \Rightarrow 1> \in gm_p$, $<\texttt{input\_transitions\_fired(0)} \Rightarrow \texttt{true}> \in ipm_p$, and $<\texttt{input\_arcs\_weights(0)} \Rightarrow 0> \in ipm_p$.

  By property of the elaboration relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\Delta(id_p)(\text{"ian"}) = 1$.

  By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma'(id_p)(\text{"itf"})[0] = \texttt{true}$ and $\sigma'(id_p)(\text{"iaw"})[0] = 0$.

  By property of $input(p) = \emptyset$, we can deduce $\sum_{t \in Fired(s')} \begin{cases} \omega \texttt{ if } post(t,p) = \omega \\ 0 \textit{ otherwise} \end{cases} = 0$.

  Rewriting the goal with $\Delta(id_p)(\text{"ian"}) = 1$, $\sigma'(id_p)(\text{"itf"})[0] = \texttt{true}$, $\sigma'(id_p)(\text{"iaw"})[0] = 0$, and $\sum_{t \in Fired(s')} \begin{cases} \omega \texttt{ if } post(t,p) = \omega \\ 0 \textit{ otherwise} \end{cases} = 0$, and simplifying the goal: tautology.

- **CASE** $input(p) \neq \emptyset$:

  By construction, $<$`input_arcs_number` $\Rightarrow |input(p)|> \in gm_p$, and by property of the elaboration relation, we can deduce $\Delta(id_p)("ian") = |input(p)|$.

  To ease the reading, let us define functions $f \in Fired(s') \rightarrow \mathbb{N}$ and $g \in [0, |input(p)| - 1] \rightarrow \mathbb{N}$ s.t. $f(t) = \begin{cases} \omega \texttt{ if } post(t,p) = \omega \\ 0 \textit{ otherwise} \end{cases}$ and $g(i) = \begin{cases} \sigma'(id_p)("iaw")[i] \texttt{ if } \sigma'(id_p)("itf")[i] \\ 0 \textit{ otherwise} \end{cases}$

  Then, the goal is: $\boxed{\sum_{t \in Fired(s')} f(t) = \sum_{i=0}^{\Delta(id_p)("ian")-1} g(i)}$

  Rewriting the goal with $\Delta(id_p)("ian") = |input(p)|$: $\boxed{\sum_{t \in Fired(s')} f(t) = \sum_{i=0}^{|input(p)|-1} g(i).}$

  Let us reason by induction on the right sum term of the goal.

  - **BASE CASE**: In that case, $0 > |input(p)| - 1$ and $\sum_{i=0}^{|input(p)|-1} g(i) = 0$.

    As $0 > |input(p)| - 1$, then $|input(p)| = 0$, thus contradicting $input(p) \neq \emptyset$.

  - **INDUCTION CASE**: In that case, $0 \leq |input(p)| - 1$.

    $$\forall F \subseteq Fired(s'),\ g(0) + \sum_{t \in F} f(t) = g(0) + \sum_{i=1}^{|input(p)|-1} g(i)$$

    $$\boxed{\sum_{t \in Fired(s')} f(t) = g(0) + \sum_{i=1}^{|input(p)|-1} g(i)}$$

    By definition of g, we can deduce $g(0) = \begin{cases} \sigma'(id_p)("iaw")[0] \texttt{ if } \sigma'(id_p)("itf")[0] \\ 0 \textit{ otherwise} \end{cases}$

    Let us perform case analysis on the value of $\sigma'(id_p)("itf")[0]$; there are two cases:

    1. $\sigma'(id_p)("itf")[0] =$ `false`:
       In that case, $g(0) = 0$, and then we can apply the induction hypothesis with $F = Fired(s')$ to solve the goal: $\sum_{t \in Fired(s')} f(t) = \sum_{i=1}^{|input(p)|-1} g(i).$
    2. $\sigma'(id_p)("itf")[0] =$ `true`:
       In that case, $g(0) = \sigma'(id_p)("iaw")[0]$ and $\sigma'(id_p)("itf")[0] =$ `true` .
       By construction, there exist a $t \in input(t)$, an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t, gm_t, ipm_t, opm_t$ s.t. $\texttt{comp}(id_t, "transition", gm_t, ipm_t, opm) \in d.cs$, an $\omega \in \mathbb{N}^*$ and an $id_{ft} \in Sigs(\Delta)$ such that:
       * $post(t,p) = \omega$

* $<\texttt{input\_arcs\_weights(0)} \Rightarrow \omega> \in ipm_p$
* $<\texttt{fired} \Rightarrow id_{ft}> \in opm_t$
* $<\texttt{input\_transitions\_fired(0)} \Rightarrow \texttt{id}_{\texttt{ft}}> \in ipm_p$

By property of the stabilize relation, $<\texttt{fired} \Rightarrow \texttt{id}_{\texttt{ft}}> \in opm_t$, $<\texttt{input\_transitions\_fired(0)} \Rightarrow \texttt{id}_{\texttt{ft}}> \in ipm_p$ and $\sigma'(id_p)("itf")[0] = \texttt{true}$, we can deduce $\sigma'(id_t)("fired") = \texttt{true}$.
Appealing to Lemma 4 and $\sigma'(id_t)("fired") = \texttt{true}$, we can deduce $t \in Fired(s')$.
As $t \in Fired(s')$, we can rewrite the left sum term of the goal as follows:

$$f(t) + \sum_{t' \in Fired(s') \setminus \{t\}} f(t') = g(0) + \sum_{i=1}^{|input(p)|-1} g(i)$$

We know that $g(0) = \sigma'(id_p)("iaw")[0]$, and by property of the stabilize relation and $<\texttt{input\_arcs\_weights(0)} \Rightarrow \omega> \in ipm_p$, we can deduce $\sigma'(id_p)("iaw")[0] = \omega$.
Rewriting the goal with $\sigma'(id_p)("iaw")[0] = \omega$:

$$f(t) + \sum_{t' \in Fired(s') \setminus \{t\}} f(t') = \omega + \sum_{i=1}^{|input(p)|-1} g(i)$$

By definition of f, and as $post(t,p) = \omega$, then $f(t) = \omega$; thus, rewriting the goal:

$$\omega + \sum_{t' \in Fired(s') \setminus \{t\}} f(t') = \omega + \sum_{i=1}^{|input(p)|-1} g(i)$$

Then, knowing that $g(0) = \omega$, we can apply the induction hypothesis with $F = Fired(s') \setminus \{t\}$: $g(0) + \sum_{t' \in Fired(s') \setminus \{t\}} f(t') = g(0) + \sum_{i=1}^{|input(p)|-1} g(i)$.

□

A.4.2 Falling edge and time counters

Lemma 35 (Falling edge equal time counters). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then* $\forall t \in T_i, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t$,
$\big(upper(I_s(t)) = \infty \wedge s'.I(t) \leq lower(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)("s\_time\_counter")\big)$
$\wedge\big(upper(I_s(t)) = \infty \wedge s'.I(t) > lower(I_s(t)) \Rightarrow \sigma'(id_t)("s\_time\_counter") = lower(I_s(t))\big)$
$\wedge\big(upper(I_s(t)) \neq \infty \wedge s'.I(t) > upper(I_s(t)) \Rightarrow \sigma'(id_t)("s\_time\_counter") = upper(I_s(t))\big)$
$\wedge\big(upper(I_s(t)) \neq \infty \wedge s'.I(t) \leq upper(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)("s\_time\_counter")\big)$.

Proof. Given a $t \in T_i$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, let us show

$\big(upper(I_s(t)) = \infty \wedge s'.I(t) \leq lower(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)("s\_time\_counter")\big)$
$\wedge\big(upper(I_s(t)) = \infty \wedge s'.I(t) > lower(I_s(t)) \Rightarrow \sigma'(id_t)("s\_time\_counter") = lower(I_s(t))\big)$
$\wedge\big(upper(I_s(t)) \neq \infty \wedge s'.I(t) > upper(I_s(t)) \Rightarrow \sigma'(id_t)("s\_time\_counter") = upper(I_s(t))\big)$
$\wedge\big(upper(I_s(t)) \neq \infty \wedge s'.I(t) \leq upper(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)("s\_time\_counter")\big)$

By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
By property of the elaboration, $\texttt{Inject}_{\downarrow}$, $\mathcal{H}$-VHDL rising edge and stabilize relations, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `time_counter` process defined in the transition design architecture, we can deduce:

$$\begin{aligned}\sigma(id_t)(\text{"se"}) = \texttt{true} \wedge \Delta(id_t)(\text{"tt"}) \neq \texttt{NOT\_TEMPORAL} \wedge \sigma(id_t)(\text{"srtc"}) = \texttt{false} \\ \wedge \sigma(id_t)(\text{"stc"}) < \Delta(id_t)(\text{"mtc"}) \Rightarrow \sigma'(id_t)(\text{"stc"}) = \sigma(id_t)(\text{"stc"}) + 1\end{aligned} \tag{A.23}$$

$$\begin{aligned}\sigma(id_t)(\text{"se"}) = \texttt{true} \wedge \Delta(id_t)(\text{"tt"}) \neq \texttt{NOT\_TEMPORAL} \wedge \sigma(id_t)(\text{"srtc"}) = \texttt{false} \\ \wedge \sigma(id_t)(\text{"stc"}) \geq \Delta(id_t)(\text{"mtc"}) \Rightarrow \sigma'(id_t)(\text{"stc"}) = \sigma(id_t)(\text{"stc"})\end{aligned} \tag{A.24}$$

$$\begin{aligned}\sigma(id_t)(\text{"se"}) = \texttt{true} \wedge \Delta(id_t)(\text{"tt"}) \neq \texttt{NOT\_TEMPORAL} \\ \wedge \sigma(id_t)(\text{"srtc"}) = \texttt{true} \Rightarrow \sigma'(id_t)(\text{"stc"}) = 1\end{aligned} \tag{A.25}$$

$$\sigma(id_t)(\text{"se"}) = \texttt{false} \vee \Delta(id_t)(\text{"tt"}) = \texttt{NOT\_TEMPORAL} \Rightarrow \sigma'(id_t)(\text{"stc"}) = 0 \tag{A.26}$$

Then, there are 4 points to show:

1. $\boxed{upper(I_s(t)) = \infty \wedge s'.I(t) \leq lower(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"})}$

   Assuming $upper(I_s(t)) = \infty$ and $s'.I(t) \leq lower(I_s(t))$, let us show $\boxed{s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"}).}$

   Let us perform case analysis on $t \in Sens(s.M)$; there are two cases:

   (a) **CASE** $t \notin Sens(s.M)$:

   By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we can deduce $\sigma(id_t)(\text{"se"}) = \texttt{false}$.
   Appealing to (A.26) and $\sigma(id_t)(\text{"se"}) = \texttt{false}$, we can deduce $\sigma'(id_t)(\text{"stc"}) = 0$.
   By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we can deduce $s'.I(t) = 0$.
   Rewriting the goal with $\sigma'(id_t)(\text{"stc"}) = 0$ and $s'.I(t) = 0$: tautology.

   (b) **CASE** $t \in Sens(s.M)$:

   By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we can deduce $\sigma(id_t)(\text{"se"}) = \texttt{true}$.
   By construction, and as $upper(I_s(t)) = \infty$, $<\texttt{transition\_type} \Rightarrow \texttt{TEMP\_A\_INF}> \in gm_t$. By property of the elaboration relation, we have $\Delta(id_t)(\text{"tt"}) = \texttt{TEMP\_A\_INF}$.
   Let us perform case analysis on $s.reset_t(t)$; there are two cases:

i. **CASE** $s.reset_t(t) = \texttt{true}$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, $\sigma(id_t)(''srtc'') = \texttt{true}$.
Appealing to (A.25), $\sigma(id_t)(''se'') = \texttt{true}$, $\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_INF}$ and $\sigma(id_t)(''srtc'') = \texttt{true}$, we can deduce $\sigma'(id_t)(''stc'') = 1$.

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$(Rule **??**), we can deduce $s'.I(t) = 1$.
Rewriting the goal with $\sigma'(id_t)(''stc'') = 1$ and $s'.I(t) = 1$: tautology.

ii. **CASE** $s.reset_t(t) = \texttt{false}$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)(''srtc'') = \texttt{false}$.
As $upper(I_s(t)) = \infty$, there exists an $a \in \mathbb{N}^*$ s.t. $I_s(t) = [a, \infty]$. Let us take such an $a \in \mathbb{N}^*$. By construction, $<\texttt{maximal\_time\_counter} \Rightarrow a> \in gm_t$, and by property of the elaboration relation, we have $\Delta(id_t)(''mtc'') = a$.

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$(Rule **??**), and knowing that $t \in Sens(s.M)$, $s.reset_t(t) = \texttt{false}$ and $upper(I_s(t)) = \infty$, we can deduce $s'.I(t) = s.I(t) + 1$.
Rewriting the goal with $s'.I(t) = s.I(t) + 1$: $\boxed{s.I(t) + 1 = \sigma'(id_t)(''stc'').}$
We assumed that $s'.I(t) \leq lower(I_s(t))$, and as $s'.I(t) = s.I(t) + 1$, then $s.I(t) + 1 \leq lower(I_s(t))$, then $s.I(t) < lower(I_s(t))$, then $s.I(t) < a$ since $a = lower(I_s(t))$.

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, and knowing that $s.I(t) < lower(I_s(t))$ and $upper(I_s(t)) = \infty$, we can deduce $s.I(t) = \sigma(id_t)(''stc'')$.
Appealing to $\Delta(id_t)(''mtc'') = a$, $s.I(t) = \sigma(id_t)(''stc'')$ and $s.I(t) < a$, we can deduce $\sigma(id_t)(''stc'') < \Delta(id_t)(''mtc'')$.
Appealing to (A.23), $\sigma(id_t)(''stc'') < \Delta(id_t)(''mtc'')$, $\sigma(id_t)(''srtc'') = \texttt{false}$ and $\sigma(id_t)(''se'') = \texttt{true}$, we can deduce: $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'') + 1$.
Rewriting the goal with $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'') + 1$ and $s.I(t) = \sigma(id_t)(''stc'')$: tautology.

2. $\boxed{upper(I_s(t)) = \infty \wedge s'.I(t) > lower(I_s(t)) \Rightarrow \sigma'(id_t)(''s\_time\_counter'') = lower(I_s(t).}$

Assuming that $upper(I_s(t)) = \infty$ and $s'.I(t) > lower(I_s(t))$, let us show $\boxed{\sigma'(id_t)(''s\_time\_counter'') = lower(I_s(t)).}$

As $upper(I_s(t)) = \infty$, there exists an $a \in \mathbb{N}^*$ s.t. $I_s(t) = [a, \infty]$. Let us take such an $a \in \mathbb{N}^*$.

By construction, $<\texttt{maximal\_time\_counter} \Rightarrow a> \in gm_t$, and $<\texttt{transition\_type} \Rightarrow \texttt{TEMP\_A\_INF}> \in gm_t$ by property of the elaboration relation, we can deduce $\Delta(id_t)(''mtc'') = a$ and $\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_INF}$.

Let us perform case analysis on $t \in Sens(s.M)$:

(a) **CASE** $t \notin Sens(s.M)$:

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), and knowing that $t \in Sens(s.M)$, we can deduce $s'.I(t) = 0$. Since $lower(I_s(t)) \in \mathbb{N}^*$, then $lower(I_s(t)) > 0$.
Contradicts $s'.I(t) > lower(I_s(t))$.

(b) **CASE** $t \in Sens(s.M)$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\sim} \sigma$ and $t \in Sens(s.M)$, we can deduce $\sigma(id_t)(''se'') = \texttt{true}$.
Let us perform case analysis on $s.reset_t(t)$; there are two cases:

i. **CASE** $s.reset_t(t) = \texttt{true}$:

By definition of $E_c, \tau \vdash s \overset{\downarrow}{\rightarrow} s'$: $s'.I(t) = 1$.
We assumed that $s'.I(t) > lower(I_s(t))$, then $1 > lower(I_s(t))$.
Contradicts $lower(I_s(t)) > 0$.

ii. **CASE** $s.reset_t(t) = \texttt{false}$:

By property of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$ and $s.reset_t(t) = \texttt{false}$, we can deduce $\sigma(id_t)(''srtc'') = \texttt{false}$.
By definition of $E_c, \tau \vdash s \overset{\downarrow}{\rightarrow} s'$ (Rule **??**), and knowing that $s'.I(t) > lower(I_s(t))$, we can deduce

$$\begin{aligned} s'.I(t) = s.I(t) + 1 &\Rightarrow s.I(t) + 1 > lower(I_s(t)) \\ &\Rightarrow s.I(t) \geq lower(I_s(t)) \end{aligned}$$

Let us perform case analysis on $s.I(t) \geq lower(I_s(t))$:

A. **CASE** $s.I(t) > lower(I_s(t))$: $\boxed{\sigma'(id_t)(''stc'') = lower(I_s(t)).}$

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we can deduce $\sigma(id_t)(''stc'') = lower(I_s(t))$.
Appealing to (A.24), we can deduce $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'')$.
Rewriting the goal with $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'')$ and $\sigma(id_t)(''stc'') = lower(I_s(t))$:
tautology.

B. **CASE** $s.I(t) = lower(I_s(t))$: $\boxed{\sigma'(id_t)(''stc'') = lower(I_s(t)).}$

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we can deduce $s.I(t) = \sigma(id_t)(''stc'')$.
Appealing to (A.24), we can deduce $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'')$.
Rewriting the goal with $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'')$, $s.I(t) = \sigma(id_t)(''stc'')$ and $s.I(t) = lower(I_s(t))$: tautology.

3. $\boxed{upper(I_s(t)) \neq \infty \wedge s'.I(t) > upper(I_s(t)) \Rightarrow \sigma'(id_t)(''s\_time\_counter'') = upper(I_s(t)).}$

Assuming that $upper(I_s(t)) \neq \infty$ and $s'.I(t) > upper(I_s(t))$, let us show
$\boxed{\sigma'(id_t)(''s\_time\_counter'') = upper(I_s(t)).}$

As $upper(I_s(t)) \neq \infty$, there exists an $a \in \mathbb{N}^*$, and a $b \in \mathbb{N}^*$ s.t. $I_s(t) = [a, b]$. Let us take such an a and b.

By construction, $<\texttt{maximal\_time\_counter} \Rightarrow b> \in gm_t$ and there exists $tt \in \{\texttt{TEMP\_A\_A}, \texttt{TEMP\_A\_B}\}$ s.t. $<\texttt{transition\_type} \Rightarrow tt> \in gm_t$.

By property of the elaboration relation and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, we can deduce $\Delta(id_t)(''mtc'') = b = upper(I_s(t))$ and $\Delta(id_t)(''tt'') \neq \texttt{NOT\_TEMP}$.

Let us perform case analysis on $t \in Sens(s.M)$:

(a) **CASE** $t \notin Sens(s.M)$:

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), and knowing that $t \in Sens(s.M)$, then $s'.I(t) = 0$. Since $upper(I_s(t)) \in \mathbb{N}^*$, then $upper(I_s(t)) > 0$.

Contradicts $s'.I(t) > upper(I_s(t))$.

(b) **CASE** $t \in Sens(s.M)$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$ and $t \in Sens(s.M)$, we can deduce $\sigma(id_t)(''se'') = \texttt{true}$.
Let us perform case analysis on $s.reset_t(t)$; there are two cases:

i. **CASE** $s.reset_t(t) = \texttt{true}$:

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we can deduce $s'.I(t) = 1$.
We assumed that $s'.I(t) > upper(I_s(t))$, then we can deduce $1 > upper(I_s(t))$.
Contradicts $upper(I_s(t)) > 0$.

ii. **CASE** $s.reset_t(t) = \texttt{false}$:

By property of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$ and $s.reset_t(t) = \texttt{false}$, we can deduce $\sigma(id_t)(''srtc'') = \texttt{false}$.
Let us perform case analysis on $s.I(t) > upper(I_s(t))$ or $s.I(t) \leq upper(I_s(t))$:

A. **CASE** $s.I(t) > upper(I_s(t))$: $\boxed{\sigma'(id_t)(''stc'') = upper(I_s(t)).}$

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$(Rule **??**), we can deduce $s'.I(t) = s.I(t)$.

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we can deduce $\sigma(id_t)(''stc'') = upper(I_s(t))$.
Appealing to (A.24), we have $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'')$.
Rewriting the goal with $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'')$ and $\sigma(id_t)(''stc'') = upper(I_s(t))$:
tautology.

B. **CASE** $s.I(t) \leq upper(I_s(t))$: $\boxed{\sigma'(id_t)(''stc'') = upper(I_s(t)).}$

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we can deduce $s.I(t) = \sigma(id_t)(''stc'')$.
Let us perform case analysis on $s.I(t) \leq upper(I_s(t))$; there are two cases:

- **CASE** $s.I(t) = upper(I_s(t))$:

  Appealing to $\Delta(id_t)(''mtc'') = b = upper(I_s(t))$, $s.I(t) = \sigma(id_t)(''stc'')$ and $s.I(t) = upper(I_s(t))$, we can deduce $\Delta(id_t)(''mtc'') \leq \sigma(id_t)(''stc'')$.
  Appealing to $\Delta(id_t)(''mtc'') \leq \sigma(id_t)(''stc'')$ and (A.24), we can deduce $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'')$.
  Rewriting the goal with $\sigma'(id_t)(''stc'') = \sigma(id_t)(''stc'')$, $s.I(t) = \sigma(id_t)(''stc'')$ and $s.I(t) = upper(I_s(t))$: tautology.

- **CASE** $s.I(t) < upper(I_s(t))$:

  By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$(Rule **??**), we can deduce $s'.I(t) = s.I(t) + 1$.

From $s'.I(t) = s.I(t) + 1$ and $s.I(t) < upper(I_s(t))$, we can deduce $s'.I(t) \leq upper(I_s(t))$; contradicts $s'.I(t) > upper(I_s(t))$.

4. $upper(I_s(t)) \neq \infty \wedge s'.I(t) \leq upper(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"})$.

Assuming that $upper(I_s(t)) \neq \infty$ and $s'.I(t) \leq upper(I_s(t))$, let us show $s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"})$.

As $upper(I_s(t)) \neq \infty$, there exists an $a \in \mathbb{N}^*$, and a $b \in \mathbb{N}^*$ s.t. $I_s(t) = [a, b]$. Let us take such an a and b.

By construction, $<$`maximal_time_counter` $\Rightarrow b>$ $\in gm_t$ and there exists $tt \in \{$`TEMP_A_A`, `TEMP_A_B`$\}$ s.t. $<$`transition_type` $\Rightarrow tt> \in gm_t$; by property of the elaboration relation, we can deduce $\Delta(id_t)(\text{"mtc"}) = b = upper(I_s(t))$ and $\Delta(id_t)(\text{"tt"}) \neq$ `NOT_TEMP`.

Let us perform case analysis on $t \in Sens(s.M)$:

(a) **CASE** $t \notin Sens(s.M)$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)(\text{"se"}) =$ `false`.
Appealing (A.26) and $\sigma(id_t)(\text{"se"}) =$ `false`, we have $\sigma'(id_t)(\text{"stc"}) = 0$.
By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$(Rule **??**), we have $s'.I(t) = 0$.
Rewriting the goal with $\sigma'(id_t)(\text{"stc"}) = 0$ and $s'.I(t) = 0$: tautology.

(b) **CASE** $t \in Sens(s.M)$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)(\text{"se"}) =$ `true`.
Let us perform case analysis on $s.reset_t(t)$:

i. **CASE** $s.reset_t(t) =$ `true`:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)(\text{"srtc"}) =$ `true`.
Appealing to (A.25), $\Delta(id_t)(\text{"tt"}) \neq$ `NOT_TEMP`, $\sigma(id_t)(\text{"se"}) =$ `true` and $\sigma(id_t)(\text{"srtc"}) =$ `true`, we have $\sigma'(id_t)(\text{"stc"}) = 1$.
By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$(Rule **??**), we have $s'.I(t) = 1$.
Rewriting the goal with $\sigma'(id_t)(\text{"stc"}) = 1$ and $s'.I(t) = 1$, tautology.

ii. **CASE** $s.reset_t(t) =$ `false`:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)(\text{"srtc"}) =$ `false`.
Let us perform case analysis on $s.I(t) > upper(I_s(t))$ or $s.I(t) \leq upper(I_s(t))$:

A. **CASE** $s.I(t) > upper(I_s(t))$:

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$, we have $s.I(t) = s'.I(t)$, and thus, $s'.I(t) > upper(I_s(t))$. Contradicts $s'.I(t) \leq upper(I_s(t))$.

B. **CASE** $s.I(t) \leq upper(I_s(t))$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.I(t) = \sigma(id_t)(\text{"}stc\text{"})$.

- **CASE** $s.I(t) < upper(I_s(t))$:
  From $s.I(t) < upper(I_s(t))$, $s.I(t) = \sigma(id_t)(\text{"}stc\text{"})$ and $\Delta(id_t)(\text{"}mtc\text{"}) = b = upper(I_s(t))$, we can deduce $\sigma(id_t)(\text{"}stc\text{"}) < \Delta(id_t)(\text{"}mtc\text{"})$.
  From (A.23), $\sigma(id_t)(\text{"}se\text{"}) = \texttt{true}$, $\Delta(id_t)(\text{"}tt\text{"}) \neq \texttt{NOT\_TEMP}$, $\sigma(id_t)(\text{"}srtc\text{"}) = \texttt{false}$ and $\sigma(id_t)(\text{"}stc\text{"}) < \Delta(id_t)(\text{"}mtc\text{"})$, we can deduce $\sigma'(id_t)(\text{"}stc\text{"}) = \sigma(id_t)(\text{"}stc\text{"}) + 1$.
  By definition of $E_c, \tau \vdash s \overset{\downarrow}{\to} s'$(Rule **??**), we can deduce $s'.I(t) = s.I(t) + 1$.
  Rewriting the goal with $\sigma'(id_t)(\text{"}stc\text{"}) = \sigma(id_t)(\text{"}stc\text{"}) + 1$ and $s'.I(t) = s.I(t) + 1$, tautology.
- **CASE** $s.I(t) = upper(I_s(t))$:
  By definition of $E_c, \tau \vdash s \overset{\downarrow}{\to} s'$(Rule **??**), we know that $s'.I(t) = s.I(t) + 1$. We assumed that $s'.I(t) \leq upper(I_s(t))$; thus, $s.I(t) + 1 \leq upper(I_s(t))$.
  Contradicts $s.I(t) = upper(I_s(t))$.

□

A.4.3 Falling edge and condition values

Lemma 36 (Falling edge equal condition values)**.** *For all sitpn, $d, \gamma, \Delta, \sigma_e, E_c, E_p, \tau, s, s', \sigma, \sigma_i, \sigma_\downarrow, \sigma'$ that verify the hypotheses of Definition 10, then $\forall c \in \mathcal{C}, id_c \in Ins(\Delta)$ s.t. $\gamma(c) = id_c$, $s'.cond(c) = \sigma'(id_c)$.*

Proof. Given a $c \in \mathcal{C}$ and an $id_c \in Ins(\Delta)$ s.t. $\gamma(c) = id_c$, let us show $\boxed{s'.cond(c) = \sigma'(id_c).}$

By definition of $E_c, \tau \vdash s \overset{\downarrow}{\to} s'$(Rule **??**), we have $s'.cond(c) = E_c(\tau, c)$.
By property of the $\texttt{Inject}_\downarrow$, the $\mathcal{H}$-VHDL falling edge, the stabilize relations and $id_c \in Ins(\Delta)$, we have $\sigma'(id_c) = E_p(\tau, \downarrow)(id_c)$.
Rewriting the goal with $s'.cond(c) = E_c(\tau, c)$ and $\sigma'(id_c) = E_p(\tau, \downarrow)(id_c)$: $\boxed{E_c(\tau, c) = E_p(\tau, \downarrow)(id_c)}$
By definition of $\gamma \vdash E_p \overset{env}{=} E_c$: $E_c(\tau, c) = E_p(\tau, \downarrow)(id_c)$.

□

A.4.4 Falling and action executions

Lemma 37 (Falling edge equal action executions)**.** *For all sitpn, $d, \gamma, \Delta, \sigma_e, E_c, E_p, \tau, s, s', \sigma, \sigma_i, \sigma_\downarrow, \sigma'$ that verify the hypotheses of Definition 10, then $\forall a \in \mathcal{A}, id_a \in Outs(\Delta)$ s.t. $\gamma(a) = id_a$, $s'.ex(a) = \sigma'(id_a)$.*

Proof. Given an $a \in \mathcal{A}$ and an $id_a \in Outs(\Delta)$ s.t. $\gamma(a) = id_a$, let us show $\boxed{s'.ex(a) = \sigma'(id_a).}$

By property of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$(Rule **??**):

$$s'.ex(a) = \sum_{p \in marked(s.M)} \mathbb{A}(p,a) \tag{A.27}$$

By construction, the generated `action` process is a part of design d's behavior, i.e there exist an $sl \subseteq Sigs(\Delta)$ and an $ss_a \in ss$ s.t. $\texttt{ps}(\text{"}action\text{"}, \emptyset, sl, ss) \in d.cs$.
By construction id_a is only assigned in the body of the `action` process during the initialization or a falling edge phase.
Let $pls(a)$ be the set of actions associated to action a, i.e $pls(a) = \{p \in P \mid \mathbb{A}(p,a) = true\}$.
Then, depending on $pls(a)$, there are two cases of assignment of output port id_a:

- **CASE** $pls(a) = \emptyset$:
  By construction, $\texttt{id}_\texttt{a} \Leftarrow \texttt{false} \in ss_{a\downarrow}$ where $ss_{a\downarrow}$ is the part of the ''`action`'' process body executed during a falling edge phase.

  By property of the $\mathcal{H}$-VHDL falling edge relation, the stabilize relation and $\texttt{ps}(\text{"}action\text{"}, \emptyset, sl, ss_a) \in d.cs$, we can deduce $\sigma'(id_a) = false$.

  By property of $\sum_{p \in marked(s.M)} \mathbb{A}(p,a)$ and $pls(a) = \emptyset$, we can deduce $\sum_{p \in marked(s.M)} \mathbb{A}(p,a) =$ `false`.

  Rewriting the goal with (A.27), $\sigma'(id_a) = false$ and $\sum_{p \in marked(s.M)} \mathbb{A}(p,a) = \texttt{false}$, tautology.

- **CASE** $pls(a) \neq \emptyset$:
  By construction, $\texttt{id}_\texttt{a} \Leftarrow \texttt{id}_{\texttt{mp}_0} + \cdots + \texttt{id}_{\texttt{mp}_\texttt{n}} \in ss_{a\downarrow}$, where $id_{mp_i} \in Sigs(\Delta)$, $ss_{a\downarrow}$ is the part of the `action` process body executed during the falling edge phase, and $n = |pls(a)| - 1$.

  By property of the $\texttt{Inject}_\downarrow$, the $\mathcal{H}$-VHDL falling edge relation, the stabilize relation, and $\texttt{ps}(\text{"}action\text{"}, \emptyset, sl, ss) \in d.cs$:

  $$\sigma'(id_a) = \sigma(id_{mp_0}) + \cdots + \sigma(id_{mp_n}) \tag{A.28}$$

  Rewriting the goal with (A.27) and (A.28): $\boxed{\sum_{p \in marked(s.M)} \mathbb{A}(p,a) = \sigma(id_{mp_0}) + \cdots + \sigma(id_{mp_n}).}$

  Let us reason on the value of $\sigma(id_{mp_0}) + \cdots + \sigma(id_{mp_n})$; there are two cases:

  - **CASE** $\sigma(id_{mp_0}) + \cdots + \sigma(id_{mp_n}) = \texttt{true}$:
    Then, we can rewrite the goal as follows: $\boxed{\sum_{p \in marked(s.M)} \mathbb{A}(p,a) = \texttt{true}.}$

    To prove the above goal, let us show $\boxed{\exists p \in marked(s.M)\ s.t.\ \mathbb{A}(p,a) = \texttt{true}.}$

    From $\sigma(id_{mp_0}) + \cdots + \sigma(id_{mp_n}) = \texttt{true}$, we can deduce that $\exists id_{mp_i}\ s.t.\ \sigma(id_{mp_i}) = \texttt{true}$.
    Let us take an id_{mp_i} s.t. $\sigma(id_{mp_i}) = \texttt{true}$.
    By construction, there exist a $p \in pls(a)$, an $id_p \in Comps(\Delta)$, gm_p, ipm_p and opm_p such that:

* $\gamma(p) = id_p$
* $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$
* $<\texttt{marked} \Rightarrow \texttt{id}_{\texttt{mp}_i}> \in opm_p$

Let us take such a p, id_p, gm_p, ipm_p and opm_p.
By property of stable σ and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce $\sigma(id_{mp_i}) = \sigma(id_p)(''marked'')$.
By property of stable σ, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `determine_marked` process defined in the place design architecture, we can deduce:

$$\sigma(id_p)(''marked'') = \sigma(id_p)(''sm'') > 0 \tag{A.29}$$

From $\sigma(id_{mp_i}) = \sigma(id_p)(''marked'')$, (A.29) and $\sigma(id_{mp_i}) = \texttt{true}$, we can deduce that $\sigma(id_p)(''marked'') = \texttt{true}$ and $(\sigma(id_p)(''sm'') > 0) = \texttt{true}$.

By property of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.M(p) = \sigma(id_p)(''sm'')$.
From $s.M(p) = \sigma(id_p)(''sm'')$ and $(\sigma(id_p)(''sm'') > 0) = \texttt{true}$, we can deduce $p \in marked(s.M)$, i.e $s.M(p) > 0$.

Let us use p to prove the goal: $\boxed{\mathbb{A}(p,a) = \texttt{true}.}$

By definition of $p \in pls(a)$, $\mathbb{A}(p,a) = \texttt{true}.$

– **CASE** $\sigma(id_{mp_0}) + \cdots + \sigma(id_{mp_n}) = \texttt{false}$:

Then, we can rewrite the goal as follows: $\boxed{\sum_{p \in marked(s.M)} \mathbb{A}(p,a) = \texttt{false}.}$

To prove the above goal, let us show $\boxed{\forall p \in marked(s.M)\ s.t.\ \mathbb{A}(p,a) = \texttt{false}.}$

Given a $p \in marked(s.M)$, let us show $\boxed{\mathbb{A}(p,a) = \texttt{false}.}$

Let us perform case analysis on $\mathbb{A}(p,a)$; there are 2 cases:

* **CASE** $\mathbb{A}(p,a) = \texttt{false}.$
* **CASE** $\mathbb{A}(p,a) = \texttt{true}$:
  By construction, there exist an $id_p \in Comps(\Delta)$, gm_{tp}, ipm_p, opm_p and $id_{mp_i} \in Sigs(\Delta)$ such that:
  · $\gamma(p) = id_p$
  · $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$
  · $<\texttt{marked} \Rightarrow \texttt{id}_{\texttt{mp}_i}> \in opm_p$
  Let us take such a id_p, gm_p, ipm_p, opm_p and id_{mp_i}.
  By property of stable σ, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and $<\texttt{marked} \Rightarrow \texttt{id}_{\texttt{mp}_i}> \in opm_p$, we can deduce $\sigma(id_{mp_i}) = \sigma(id_p)(''marked'')$.
  By property of stable σ, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `determine_marked` process defined in the place design architecture, we can deduce:

$$\sigma(id_p)(''marked'') = (\sigma(id_p)(''sm'') > 0) \tag{A.30}$$

From $\sigma(id_{mp_0}) + \cdots + \sigma(id_{mp_n}) = \texttt{false}$, we can deduce $\sigma(id_{mp_i}) = \texttt{false}$.
From $\sigma(id_p)(\text{"}marked\text{"}) = \texttt{false}$, we can deduce $(\sigma(id_p)(\text{"}sm\text{"}) > 0) = \texttt{false}$.
By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.M(p) = \sigma(id_p)(\text{"}sm\text{"})$, and thus, we can deduce that $s.M(p) = 0$ (equivalent to $(s.M(p) > 0) = \texttt{false}$).
Contradicts $p \in marked(s.M)$ (i.e, $s.M(p) > 0$).

□

A.4.5 Falling edge and function executions

Lemma 38 (Falling edge equal function executions). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then $\forall f \in \mathcal{F}, id_f \in Outs(\Delta)$ s.t. $\gamma(f) = id_f$, $s'.ex(f) = \sigma'(id_f)$.*

Proof. Given an $f \in \mathcal{F}$ and an $id_f \in Outs(\Delta)$ s.t. $\gamma(f) = id_f$, let us show $\boxed{s'.ex(f) = \sigma'(id_f).}$

By property of $E_c, \tau \vdash s \overset{\downarrow}{\to} s'$, we can deduce $s.ex(f) = s'.ex(f)$.
By construction, id_f is an output port identifier of boolean type in the $\mathcal{H}$-VHDL design d assigned by the `function` process only during the initialization or during a rising edge phase.
By property of the $\mathcal{H}$-VHDL $\texttt{Inject}_\uparrow$, rising edge, stabilize relations, and the `function` process, we can deduce $\sigma(id_f) = \sigma'(id_f)$.
Rewriting the goal with $s.ex(f) = s'.ex(f)$ and $\sigma(id_f) = \sigma'(id_f)$, $\boxed{s.ex(f) = \sigma(id_f).}$
By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, $s.ex(f) = \sigma(id_f)$. □

A.4.6 Falling edge and firable transitions

Lemma 39 (Falling edge equal firable). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then $\forall t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $t \in Firable(s') \Leftrightarrow \sigma'(id_t)(\text{"}s\_firable\text{"}) = \texttt{true}$.*

Proof. Given a $t \in T$ and $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, let us show that
$\boxed{t \in Firable(s') \Leftrightarrow \sigma'(id_t)(\text{"}s\_firable\text{"}) = \texttt{true}.}$

The proof is in two parts:

1. Assuming that $t \in Firable(s')$, let us show $\boxed{\sigma'(id_t)(\text{"}s\_firable\text{"}) = \texttt{true}.}$

   Appealing to Lemma 40: $\sigma'(id_t)(\text{"}s\_firable\text{"}) = \texttt{true}.$

2. Assuming that $\sigma'(id_t)(\text{"}s\_firable\text{"}) = \texttt{true}$, let us show $\boxed{t \in Firable(s').}$

   Appealing to Lemma 41: $t \in Firable(s')$.

□

Lemma 40 (Falling edge equal firable 1)**.** *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then* $\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t, t \in Firable(s') \Rightarrow \sigma'(id_t)(\text{"}s\_firable\text{"}) = \texttt{true}$.

Proof. Given a $t \in T$ and $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, and assuming that $t \in Firable(s')$, let us show $\boxed{\sigma'(id_t)(\text{"}s\_firable\text{"}) = \texttt{true}.}$

By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
By property of the $\texttt{Inject}_\downarrow$ relation, the $\mathcal{H}$-VHDL falling edge relation, the stabilize relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `firable` process defined in the transition design architecture, we can deduce:

$$\sigma'(id_t)(\text{"}sfa\text{"}) = \sigma(id_t)(\text{"}se\text{"}) \,.\, \sigma(id_t)(\text{"}scc\text{"}) \,.\, \texttt{checktc}(\Delta(id_t), \sigma(id_t)) \tag{A.31}$$

Term $\texttt{checktc}(\Delta(id_t), \sigma(id_t))$ is defined as follows:

$$\begin{aligned}\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \Big(&\texttt{not}\ \sigma(id_t)(\text{"}srtc\text{"}) \,. \\ &\big[(\Delta(id_t)(\text{"}tt\text{"}) = \texttt{TEMP\_A\_B} \,.\, (\sigma(id_t)(\text{"}stc\text{"}) \geq \sigma(id_t)(\text{"}A\text{"}) - 1) \\ &\qquad .\, (\sigma(id_t)(\text{"}stc\text{"}) \leq \sigma(id_t)(\text{"}B\text{"}) - 1)) \\ &+ (\Delta(id_t)(\text{"}tt\text{"}) = \texttt{TEMP\_A\_A} \,.\, (\sigma(id_t)(\text{"}stc\text{"}) = \sigma(id_t)(\text{"}A\text{"}) - 1)) \\ &+ (\Delta(id_t)(\text{"}tt\text{"}) = \texttt{TEMP\_A\_INF} \,.\, (\sigma(id_t)(\text{"}stc\text{"}) \geq \sigma(id_t)(\text{"}A\text{"}) - 1))\big]\Big) \\ &+ \big(\sigma(id_t)(\text{"}srtc\text{"}) \,.\, \Delta(id_t)(\text{"}tt\text{"}) \neq \texttt{NOT\_TEMP} \,.\, \sigma(id_t)(\text{"}A\text{"}) = 1\big) \\ &+ \Delta(id_t)(\text{"}tt\text{"}) = \texttt{NOT\_TEMP}\end{aligned} \tag{A.32}$$

Rewriting the goal with (A.31): $\boxed{\sigma(id_t)(\text{"}se\text{"}) \,.\, \sigma(id_t)(\text{"}scc\text{"}) \,.\, \texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}.}$
Then, there are three points to prove:

1. $\boxed{\sigma(id_t)(\text{"}se\text{"}) = \texttt{true}}$:

   From $t \in Firable(s')$, we can deduce $t \in Sens(s'.M)$. By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$, we have $s.M = s'.M$, and thus, we can deduce $t \in Sens(s.M)$.

   By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we know that $t \in Sens(s.M)$ implies $\sigma(id_t)(\text{"}se\text{"}) = \texttt{true}$.

2. $\boxed{\sigma(id_t)(\text{"}scc\text{"}) = \texttt{true}}$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$:

$$\sigma(id_t)(\text{"scc"}) = \prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau, c)) & if\ \mathbb{C}(t,c) = -1 \end{cases} \tag{A.33}$$

where $conds(t) = \{c \in \mathcal{C} \mid \mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1\}$.

Rewriting the goal with (A.33): $\boxed{\prod_{c \in conds(t)} \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau, c)) & if\ \mathbb{C}(t,c) = -1 \end{cases} = \texttt{true}.}$

To ease the reading, let us define $f(c) = \begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau, c)) & if\ \mathbb{C}(t,c) = -1 \end{cases}$.

Let us reason by induction on the left term of the goal:

- **BASE CASE**: $\texttt{true} = \texttt{true}.$
- **INDUCTION CASE**:

  $$\prod_{c' \in conds(t) \setminus \{c\}} f(c') = \texttt{true}$$

  $$\boxed{f(c) \cdot \prod_{c' \in conds(t) \setminus \{c\}} f(c') = \texttt{true}.}$$

  Rewriting the goal with the induction hypothesis, simplifying the goal, and unfolding the definition of $f(c)$: $\boxed{\begin{cases} E_c(\tau, c) & if\ \mathbb{C}(t,c) = 1 \\ \texttt{not}(E_c(\tau, c)) & if\ \mathbb{C}(t,c) = -1 \end{cases} = \texttt{true}.}$

  As $c \in conds(t)$, let us perform case analysis on $\mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1$:

  (a) **CASE** $\mathbb{C}(t,c) = 1$: $\boxed{E_c(\tau, c) = \texttt{true}.}$

  By definition of $t \in Firable(s')$, we can deduce that $s'.cond(c) = \texttt{true}$. By definition of $E_c, \tau \vdash s \overset{\downarrow}{\to} s'$ (Rule **??**), we have $s'.cond(c) = E_c(\tau, c)$. Thus, $E_c(\tau, c) = \texttt{true}.$

  (b) $\mathbb{C}(t,c) = -1$: $\boxed{\texttt{not}\ E_c(\tau, c) = \texttt{true}.}$

  By definition of $t \in Firable(s')$, we can deduce that $s'.cond(c) = \texttt{false}$. By definition of $E_c, \tau \vdash s \overset{\downarrow}{\to} s'$ (Rule **??**), we have $s'.cond(c) = E_c(\tau, c)$. Thus, $\texttt{not}\ E_c(\tau, c) = \texttt{true}.$

3. $\boxed{\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}}$:

By definition of $t \in Firable(s')$, we have $t \notin T_i \vee s'.I(t) \in I_s(t)$. Let us perform case analysis on $t \notin T_i \vee s'.I(t) \in I_s(t)$:

(a) **CASE** $t \notin T_i$: $\boxed{\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}}$

By construction, $<\texttt{transition\_type} \Rightarrow \texttt{NOT\_TEMP}> \in gm_t$, and by property of the elaboration relation, we have $\Delta(id_t)(''tt'') = \texttt{NOT\_TEMP}$.
From $\Delta(id_t)(''tt'') = \texttt{NOT\_TEMP}$, and by definition of $\texttt{checktc}(\Delta(id_t), \sigma(id_t))$, we can deduce $\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}$.

(b) **CASE** $s'.I(t) \in I_s(t)$: $\boxed{\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}}$

From $s'.I(t) \in I_s(t)$, we can deduce that $t \in T_i$. Thus, by construction, there exists $tt \in \{\texttt{TEMP\_A\_B}, \texttt{TEMP\_A\_A}, \texttt{TEMP\_A\_INF}\}$ s.t. $<\texttt{transition\_type} \Rightarrow tt> \in gm_t$. By property of the elaboration relation, we have $\Delta(id_t)(''tt'') = tt$, and thus, we know $\Delta(id_t)(''tt'') \neq \texttt{NOT\_TEMP}$. Therefore, we can simplfy the term $\texttt{checktc}(\Delta(id_t), \sigma(id_t))$ as follows:

$$\begin{aligned}\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \Big(&\texttt{not}\ \sigma(id_t)(''srtc'')\ .\\ &\big[(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_B}\ .\ (\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1)\\ &\qquad .\ (\sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1))\\ &+(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_A}\ .\\ &\quad (\sigma(id_t)(''stc'') = \sigma(id_t)(''A'') - 1))\\ &+(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_INF}\ .\\ &\quad (\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1))\big]\Big)\\ &+ \big(\sigma(id_t)(''srtc'')\ .\ \sigma(id_t)(''A'') = 1\big)\end{aligned} \tag{A.34}$$

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.reset_t(t) = \sigma(id_t)(''srtc'')$.
Let us perform case analysis on the value $s.reset_t(t)$:

i. **CASE** $s.reset_t(t) = \texttt{true}$: $\boxed{\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}}$

From $s.reset_t(t) = \sigma(id_t)(''srtc'')$, we can deduce that $\sigma(id_t)(''srtc'') = \texttt{true}$.
From $\sigma(id_t)(''srtc'') = \texttt{true}$, we can simplify the term $\texttt{checktc}(\Delta(id_t), \sigma(id_t))$ as follows:

$$\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \big(\sigma(id_t)(''A'') = 1\big) \tag{A.35}$$

Rewriting the goal with (A.35), and simplifying the goal: $\boxed{\sigma(id_t)(''A'') = 1.}$

By definition of $E_c, \tau \vdash s \overset{\downarrow}{\rightarrow} s'$ (Rule **??**), from $t \in Sens(s.M)$ and $s.reset_t(t) = \texttt{true}$, we can deduce $s'.I(t) = 1$. We know that $s'.I(t) \in I_s(t)$, and thus, we have $1 \in I_s(t)$. By definition of $1 \in I_s(t)$, there exist an $a \in \mathbb{N}^*$ and a $ni \in \mathbb{N}^* \sqcup \{\infty\}$ s.t. $I_s(t) = [a, ni]$ and $1 \in [a, ni]$.

By definition of $1 \in [a, ni]$, we have $a \leq 1$, and since $a \in \mathbb{N}^*$, we can deduce $a = 1$.
By construction, $<$`time_A_value` $\Rightarrow a>$ $\in ipm_t$, and by property of stable σ, we have $\sigma(id_t)(''A'') = a = 1$.

ii. **CASE** $s.reset_t(t) =$ `false`: $\boxed{\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}}$

From $s.reset_t(t) = \sigma(id_t)(''srtc'')$, we can deduce $\sigma(id_t)(''srtc'') =$ `false`.
From $\sigma(id_t)(''srtc'') =$ `false`, we can simplify the term $\texttt{checktc}(\Delta(id_t), \sigma(id_t))$ as follows:

$$\begin{array}{c} \texttt{checktc}(\Delta(id_t), \sigma(id_t)) \\ = \\ \big(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_B} \;\; . (\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1) \\ \qquad\qquad\qquad\qquad\quad . (\sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1)\big) \\ +(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_A} \,.\, (\sigma(id_t)(''stc'') = \sigma(id_t)(''A'') - 1)) \\ +(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_INF} \,.\, (\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1)) \end{array} \tag{A.36}$$

Let us perform case analysis on $I_s(t)$; there are two cases:

- **CASE** $I_s(t) = [a, b]$ where $a, b \in \mathbb{N}^*$; then, either $a = b$ or $a \neq b$:
  - **CASE** $a = b$:
    Then, we have $I_s(t) = [a, a]$, and by construction $<$`transition_type`$\Rightarrow$ `TEMP_A_A`$> \in gm_t$. By property of the elaboration relation, we have $\Delta(id_t)(''tt'') =$`TEMP_A_A`; thus we can simplify the `checktc` term as follows:

$$\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = (\sigma(id_t)(''stc'') = \sigma(id_t)(''A'') - 1) \tag{A.37}$$

    Rewriting the goal with (A.37), and simplifying the goal:
    $\boxed{\sigma(id_t)(''stc'') = \sigma(id_t)(''A'') - 1.}$

    From $s'.I(t) \in [a, a]$, we can deduce that $s'.I(t) = a$. Let us perform case analysis on $s.I(t) < upper(I_s(t))$ or $s.I(t) \geq upper(I_s(t))$:
    - **CASE** $s.I(t) < upper(I_s(t))$:
      By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.I(t) = \sigma(id_t)(''stc'')$. By definition of $E_c, \tau \vdash s \overset{\downarrow}{\rightarrow} s'$ (Rule **??**), we have $s'.I(t) = s.I(t) + 1$. From $s'.I(t) = a$ and $s'.I(t) = s.I(t) + 1$, we can deduce $a - 1 = s.I(t)$.
      By construction, $<$`time_A_value` $\Rightarrow a>$ $\in ipm_t$, and by property of stable σ, we have $\sigma(id_t)(''A'') = a$.
      Rewriting the goal with $\sigma(id_t)(''A'') = a$, $s.I(t) = \sigma(id_t)(''stc'')$, and $a - 1 = s.I(t)$: tautology.
    - **CASE** $s.I(t) \geq upper(I_s(t))$:
      In the case where $s.I(t) > upper(I_s(t))$, then $s.I(t) > a$. By definition of $E_c, \tau \vdash s \overset{\downarrow}{\rightarrow} s'$ (Rule **??**), we have $s.I(t) = s'.I(t) = a$. Then, $a > a$ is a contradiction.

In the case where $s.I(t) = upper(I_s(t))$, then $s.I(t) = a$. By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s'.I(t) = s.I(t) + 1$. Then, we have $s'.I(t) = a$ and $s'.I(t) = a + 1$. Then, $a = a + 1$ is a contradiction.

– **CASE** $a \neq b$: $\boxed{\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}}$
Then, we have $I_s(t) = [a, b]$, and by construction <transition_type⇒ TEMP_A_B> $\in gm_t$. By property of the elaboration relation, we have $\Delta(id_t)(''tt'')$ =TEMP_A_B; thus we can simplify the term checktc as follows:

$$\begin{gathered}\texttt{checktc}(\Delta(id_t), \sigma(id_t)) \\ = \\ (\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1) \,.\, (\sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1)\end{gathered} \tag{A.38}$$

Rewriting the goal with (A.38), and simplifying the goal:
$\boxed{(\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1) \wedge (\sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1).}$

Let us perform case analysis on $s.I(t) < upper(I_s(t))$ or $s.I(t) \geq upper(I_s(t))$:
∗ **CASE** $s.I(t) < upper(I_s(t))$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.I(t) = \sigma(id_t)(''stc'')$. By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$(Rule **??**), we have $s'.I(t) = s.I(t) + 1$. By definition of $s'.I(t) \in [a, b]$:
$\Rightarrow a \leq s'.I(t) \leq b$.
$\Rightarrow a \leq s'.I(t) \wedge s'.I(t) \leq b$
$\Rightarrow a \leq s.I(t) + 1 \wedge s.I(t) + 1 \leq b$
$\Rightarrow a - 1 \leq s.I(t) \wedge s.I(t) \leq b - 1$
By construction, <time_A_value $\Rightarrow a>$ $\in ipm_t$ and <time_B_value $\Rightarrow b>$ $\in ipm_t$, and by property of stable σ, we have $\sigma(id_t)(''A'') = a$ and $\sigma(id_t)(''B'') = b$.
Rewriting the goal with $\sigma(id_t)(''A'') = a$, $\sigma(id_t)(''B'') = b$ and $s.I(t) = \sigma(id_t)(''stc'')$: $a - 1 \leq s.I(t) \wedge s.I(t) \leq b - 1$.

∗ **CASE** $s.I(t) \geq upper(I_s(t))$:

In the case where $s.I(t) > upper(I_s(t))$, then $s.I(t) > b$. By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s.I(t) = s'.I(t) = b$. Then, $b > b$ is a contradiction.

In the case where $s.I(t) = upper(I_s(t))$, then $s.I(t) = b$. By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s'.I(t) = s.I(t) + 1$.
By definition of $s'.I(t) \in [a, b]$, we have $s'.I(t) \leq b$:
$\Rightarrow s.I(t) + 1 \leq b$
$\Rightarrow$ $b + 1 \leq b$ is contradiction.

• **CASE** $I_s(t) = [a, \infty]$ where $a \in \mathbb{N}^*$: $\boxed{\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}}$

By construction <`transition_type` ⇒ `TEMP_A_INF`> ∈ gm_t. By property of the elaboration relation, we have $\Delta(id_t)(''tt'') =$ `TEMP_A_INF`; thus we can simplify the term `checktc` as follows:

$$\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = (\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1)) \tag{A.39}$$

Rewriting the goal with (A.39), and simplifying the goal:
$\boxed{\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1.}$

From $s'.I(t) \in [a, \infty]$, we can deduce $a \leq s'.I(t)$. Then, let us perform case analysis on $s.I(t) \leq lower(I_s(t))$ or $s.I(t) > lower(I_s(t))$:

– **CASE** $s.I(t) \leq lower(I_s(t))$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.I(t) = \sigma(id_t)(''stc'')$.
By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s'.I(t) = s.I(t) + 1$:
$\Rightarrow s'.I(t) \geq a$
$\Rightarrow s.I(t) + 1 \geq a$
$\Rightarrow s.I(t) \geq a - 1$
By construction, <`time_A_value` ⇒a> ∈ ipm_t, and by property of stable σ, we have $\sigma(id_t)(''A'') = a$.
Rewriting the goal with $\sigma(id_t)(''A'') = a$ and $s.I(t) = \sigma(id_t)(''stc'')$:
$s.I(t) \geq a - 1.$

– **CASE** $s.I(t) > lower(I_s(t))$:

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)(''stc'') = lower(I_s(t)) = a$.
By construction, <`time_A_value` ⇒a> ∈ ipm_t, and by property of stable σ, we have $\sigma(id_t)(''A'') = a$.
Rewriting the goal with $\sigma(id_t)(''stc'') = a$ and $\sigma(id_t)(''A'') = a$: $a \geq a - 1.$

□

Lemma 41 (Falling Edge Equal Firable 2). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then $\forall t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $\sigma'(id_t)(''s\_firable'') =$ `true` $\Rightarrow t \in Firable(s')$.*

Proof. Given a $t \in T$ and $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, and assuming that $\sigma'(id_t)(''s\_firable'') =$ `true`, let us show $\boxed{t \in Firable(s').}$

By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. `comp`(id_t, `transition`, g_t, i_t, o_t) ∈ $d.cs$.
By property of the `Inject`$_\downarrow$ relation, the $\mathcal{H}$-VHDL falling edge relation, the stabilize relation, `comp`(id_t, `transition`, g_t, i_t, o_t) ∈ $d.cs$, and through the examination of the `firable` process defined in the transition design architecture, we can deduce:

$$\sigma'(id_t)(''sfa'') = \sigma(id_t)(''se'') \,.\, \sigma(id_t)(''scc'') \,.\, \texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true} \tag{A.40}$$

From (A.40), we can deduce:

$$\sigma(id_t)(\text{"se"}) = \texttt{true} \tag{A.41}$$
$$\sigma(id_t)(\text{"scc"}) = \texttt{true} \tag{A.42}$$
$$\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true} \tag{A.43}$$

Term $\texttt{checktc}(\Delta(id_t), \sigma(id_t))$ as the same definition as in Lemma Falling edge equal firable 1. By definition of $t \in Firable(s')$, there are three points to prove:

1. $\boxed{t \in Sens(s'.M)}$
2. $\boxed{\forall c \in \mathcal{C},\ \mathbb{C}(t,c) = 1 \Rightarrow s'.cond(c) = \texttt{true} \text{ and } \mathbb{C}(t,c) = -1 \Rightarrow s'.cond(c) = \texttt{false}}$
3. $\boxed{t \notin T_i \vee s'.I(t) \in I_s(t)}$

Let us prove these three points:

1. $\boxed{t \in Sens(s'.M)}$:

   By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$, we have $s.M = s'.M$. Rewriting the goal with $s.M = s'.M$: $\boxed{t \in Sens(s.M).}$

   By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)(\text{"se"}) = \texttt{true} \Leftrightarrow t \in Sens(s.M)$.

   From $\sigma(id_t)(\text{"se"}) = \texttt{true}$, we can deduce: $t \in Sens(s.M)$.

2. $\boxed{\forall c \in \mathcal{C},\ \mathbb{C}(t,c) = 1 \Rightarrow s'.cond(c) = \texttt{true} \text{ and } \mathbb{C}(t,c) = -1 \Rightarrow s'.cond(c) = \texttt{false}}$

   Given a $c \in \mathcal{C}$, there are two points to prove:

   (a) $\boxed{\mathbb{C}(t,c) = 1 \Rightarrow s'.cond(c) = \texttt{true}.}$

   (b) $\boxed{\mathbb{C}(t,c) = -1 \Rightarrow s'.cond(c) = \texttt{false}.}$

   Let us prove these two points:

   (a) Assuming that $\mathbb{C}(t,c) = 1$, let us show $\boxed{s'.cond(c) = \texttt{true}.}$

   By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have:

$$\sigma(id_t)(\text{"scc"}) = \prod_{c' \in conds(t)} \begin{cases} E_c(\tau, c') & if\ \mathbb{C}(t,c') = 1 \\ \texttt{not}(E_c(\tau, c')) & if\ \mathbb{C}(t,c') = -1 \end{cases} = \texttt{true} \tag{A.44}$$

   where $conds(t) = \{c_i \in \mathcal{C} \mid \mathbb{C}(t,c_i) = 1 \vee \mathbb{C}(t,c_i) = -1\}$.

From $\mathbb{C}(t,c) = 1$, we can deduce $c \in conds(t)$. By definition of the product expression, we have:

$$E_c(\tau,c)\,.\prod_{c'\in conds(t)\setminus\{c\}}\begin{cases}E_c(\tau,c') & if\ \mathbb{C}(t,c')=1\\ \texttt{not}(E_c(\tau,c')) & if\ \mathbb{C}(t,c')=-1\end{cases} = \texttt{true} \tag{A.45}$$

From (A.45), we can deduce that $E_c(\tau,c) = \texttt{true}$.

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s'.cond(c) = E_c(\tau,c)$.
Rewriting the goal with $s'.cond(c) = E_c(\tau,c)$ and $E_c(\tau,c) = \texttt{true}$: tautology.

(b) Assuming that $\mathbb{C}(t,c) = -1$, let us show $\boxed{s'.cond(c) = \texttt{false}.}$

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have:

$$\sigma(id_t)(\text{"}scc\text{"}) = \prod_{c'\in conds(t)}\begin{cases}E_c(\tau,c') & if\ \mathbb{C}(t,c')=1\\ \texttt{not}(E_c(\tau,c')) & if\ \mathbb{C}(t,c')=-1\end{cases} = \texttt{true} \tag{A.46}$$

where $conds(t) = \{c' \in \mathcal{C} \mid \mathbb{C}(t,c') = 1 \vee \mathbb{C}(t,c') = -1\}$.
From $\mathbb{C}(t,c) = -1$, we can deduce $c \in conds(t)$. By definition of the product expression, we have:

$$\texttt{not}\ E_c(\tau,c)\,.\prod_{c'\in conds(t)\setminus\{c\}}\begin{cases}E_c(\tau,c') & if\ \mathbb{C}(t,c')=1\\ \texttt{not}(E_c(\tau,c')) & if\ \mathbb{C}(t,c')=-1\end{cases} = \texttt{true} \tag{A.47}$$

From (A.47), we can deduce that $E_c(\tau,c) = \texttt{false}$.

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s'.cond(c) = E_c(\tau,c)$.
Rewriting the goal with $s'.cond(c) = E_c(\tau,c)$ and $E_c(\tau,c) = \texttt{false}$: tautology.

3. $\boxed{t \notin T_i \vee s'.I(t) \in I_s(t)}$

Reasoning on $\texttt{checktc}(\Delta(id_t), \sigma(id_t)) = \texttt{true}$, there are 3 cases:

(a) $\left(\texttt{not}\ \sigma(id_t)(\text{"}srtc\text{"})\,.\,[\ldots]\right) = \texttt{true}$[1]

(b) $\left(\sigma(id_t)(\text{"}srtc\text{"})\,.\,\Delta(id_t)(\text{"}tt\text{"}) \neq \texttt{NOT\_TEMP}\,.\,\sigma(id_t)(\text{"}A\text{"}) = 1\right) = \texttt{true}$

(c) $\left(\Delta(id_t)(\text{"}tt\text{"}) = \texttt{NOT\_TEMP}\right) = \texttt{true}$

(a) **CASE** $\left(\texttt{not}\ \sigma(id_t)(\text{"}srtc\text{"})\,.\,[\ldots]\right) = \texttt{true}$:
Then, we can deduce $\texttt{not}\ \sigma(id_t)(\text{"}srtc\text{"}) = \texttt{true}$ and $[\ldots] = \texttt{true}$. From $\texttt{not}\ \sigma(id_t)(\text{"}srtc\text{"}) = \texttt{true}$, we can deduce $\sigma(id_t)(\text{"}srtc\text{"}) = \texttt{false}$, and from $[\ldots] = \texttt{true}$, we have three other cases:

[1] See equation (A.32) for the full definition.

i. **CASE** $\big(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_B} \,.\, (\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1) \,.\, (\sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1)\big) = \texttt{true}$

ii. **CASE** $(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_A} \,.\, (\sigma(id_t)(''stc'') = \sigma(id_t)(''A'') - 1)) = \texttt{true}$

iii. **CASE** $(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_INF} \,.\, (\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1)) = \texttt{true}$

Let us prove the goal is these three contexts:

i. **CASE** $\big(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_B} \,.\, (\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1) \,.\, (\sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1)\big) = \texttt{true}$:
Then, converting boolean equalities into intuitionistic predicates, we have:
- $\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_B}$
- $\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1$
- $\sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1$

By property of the elaboration relation, and $\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_B}$, there exist $a, b \in \mathbb{N}^*$ s.t. $I_s(t) = [a, b]$. Let us take such an a and b. Then, let us show $\boxed{s'.I(t) \in I_s(t).}$

Rewriting the goal with $I_s(t) = [a, b]$: $\boxed{s'.I(t) \in [a, b].}$
By construction, <`time_A_value` $\Rightarrow a>$ and <`time_B_value` $\Rightarrow b>$, and by property of stable σ, we have $\sigma(id_t)(''A'') = a$ and $\sigma(id_t)(''B'') = b$.
Rewriting the goal with $\sigma(id_t)(''A'') = a$ and $\sigma(id_t)(''B'') = b$, and by definition of $\in$: $\boxed{\sigma(id_t)(''A'') \leq s'.I(t) \leq \sigma(id_t)(''B'').}$
Now, let us perform case analysis on $s.I(t) \leq upper(I_s(t))$ or $s.I(t) > upper(I_s(t))$:
- **CASE** $s.I(t) \leq upper(I_s(t))$:

  By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.I(t) = \sigma(id_t)(''stc'')$.
  From $\sigma(id_t)(''se'') = \texttt{true}$, we can deduce $t \in Sens(s.M)$, and from $\sigma(id_t)(''srtc'') = \texttt{false}$, we can deduce $s.reset_t(t) = \texttt{false}$. Then, by definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s'.I(t) = s.I(t) + 1$.
  $\Rightarrow$ $\boxed{\sigma(id_t)(''A'') \leq s.I(t) + 1 \leq \sigma(id_t)(''B'')}$ (by $s'.I(t) = s.I(t) + 1$)
  $\Rightarrow$ $\boxed{\sigma(id_t)(''A'') \leq \sigma(id_t)(''stc'') + 1 \leq \sigma(id_t)(''B'')}$ (by $s.I(t) = \sigma(id_t)(''stc'')$)
  $\Rightarrow$ $\boxed{\sigma(id_t)(''A'') - 1 \leq \sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1}$
  We assumed $\sigma(id_t)(''stc'') \geq \sigma(id_t)(''A'') - 1$ and $\sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1$, and thus we can deduce: $\sigma(id_t)(''A'') - 1 \leq \sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1$
- **CASE** $s.I(t) > upper(I_s(t))$:

  By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)(''stc'') = upper(I_s(t)) = b$.
  Then, from $\sigma(id_t)(''stc'') \leq \sigma(id_t)(''B'') - 1$, $\sigma(id_t)(''stc'') = upper(I_s(t)) = b$ and $\sigma(id_t)(''B'') = b$, we can deduce the following contradiction:
  $\sigma(id_t)(''B'') \leq \sigma(id_t)(''B'') - 1.$

ii. $(\Delta(id_t)(''tt'') = \texttt{TEMP\_A\_A} \,.\, (\sigma(id_t)(''stc'') = \sigma(id_t)(''A'') - 1)) = \texttt{true}$:
Then, converting boolean equalities into intuitionistic predicates, we have:

- $\Delta(id_t)("tt") = \texttt{TEMP\_A\_A}$
- $\sigma(id_t)("stc") = \sigma(id_t)("A") - 1$

By property of the elaboration relation, and $\Delta(id_t)("tt") = \texttt{TEMP\_A\_A}$, there exist $a \in \mathbb{N}^*$ s.t. $I_s(t) = [a, a]$. Let us take such an a. Then, let us show $\boxed{s'.I(t) \in I_s(t).}$

Rewriting the goal with $I_s(t) = [a, a]$: $\boxed{s'.I(t) \in [a, a].}$

By construction, $<\texttt{time\_A\_value} \Rightarrow a>$, and by property of stable σ, we have $\sigma(id_t)("A") = a$.

Rewriting the goal with $\sigma(id_t)("A") = a$, unfolding the definition of $\in$, and simplifying the goal: $\boxed{s'.I(t) = \sigma(id_t)("A").}$

Now, let us perform case analysis on $s.I(t) \leq upper(I_s(t))$ or $s.I(t) > upper(I_s(t))$:

- **CASE** $s.I(t) \leq upper(I_s(t))$:

  By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.I(t) = \sigma(id_t)("stc")$.
  From $\sigma(id_t)("se") = \texttt{true}$, we can deduce $t \in Sens(s.M)$, and from $\sigma(id_t)("srtc") = \texttt{false}$, we can deduce $s.reset_t(t) = \texttt{false}$. Then, by definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s'.I(t) = s.I(t) + 1$.
  $\Rightarrow$ $\boxed{s.I(t) + 1 = \sigma(id_t)("A")}$ (by $s'.I(t) = s.I(t) + 1$)
  $\Rightarrow$ $\boxed{\sigma(id_t)("stc") + 1 = \sigma(id_t)("A")}$ (by $s.I(t) = \sigma(id_t)("stc")$)
  $\Rightarrow$ $\sigma(id_t)("stc") = \sigma(id_t)("A") - 1$ (assumption)

- **CASE** $s.I(t) > upper(I_s(t))$:

  By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)("stc") = upper(I_s(t)) = a$.
  Then, from $\sigma(id_t)("stc") = \sigma(id_t)("A") - 1$, $\sigma(id_t)("stc") = upper(I_s(t)) = a$, $\sigma(id_t)("A") = a$, and $a \in \mathbb{N}^*$, we can derive the following contradiction:
  $\sigma(id_t)("A") = \sigma(id_t)("A") - 1.$

iii. $(\Delta(id_t)("tt") = \texttt{TEMP\_A\_INF} \,.\, (\sigma(id_t)("stc") \geq \sigma(id_t)("A") - 1)) = \texttt{true}$:
Then, converting boolean equalities into intuitionistic predicates, we have:

- $\Delta(id_t)("tt") = \texttt{TEMP\_A\_INF}$
- $\sigma(id_t)("stc") \geq \sigma(id_t)("A") - 1$

By property of the elaboration relation, and $\Delta(id_t)("tt") = \texttt{TEMP\_A\_INF}$, there exist $a \in \mathbb{N}^*$ s.t. $I_s(t) = [a, \infty]$. Let us take such an a. Then, let us show $\boxed{s'.I(t) \in I_s(t).}$

Rewriting the goal with $I_s(t) = [a, \infty]$: $\boxed{s'.I(t) \in [a, \infty].}$

By construction, $<\texttt{time\_A\_value} \Rightarrow a>$, and by property of stable σ, we have $\sigma(id_t)("A") = a$.

Rewriting the goal with $\sigma(id_t)("A") = a$, unfolding the definition of $\in$, and simplifying the goal: $\boxed{\sigma(id_t)("A") \leq s'.I(t).}$

Now, let us perform case analysis on $s.I(t) \leq lower(I_s(t))$ or $s.I(t) > lower(I_s(t))$:

- **CASE** $s.I(t) \leq lower(I_s(t))$:

  By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $s.I(t) = \sigma(id_t)("stc")$.

From $\sigma(id_t)(\text{"}se\text{"}) = \texttt{true}$, we can deduce $t \in Sens(s.M)$, and from $\sigma(id_t)(\text{"}srtc\text{"}) = \texttt{false}$, we can deduce $s.reset_t(t) = \texttt{false}$. Then, by definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s'.I(t) = s.I(t) + 1$.

$\Rightarrow$ $\boxed{\sigma(id_t)(\text{"}A\text{"}) \leq s.I(t) + 1}$ (by $s'.I(t) = s.I(t) + 1$)

$\Rightarrow$ $\boxed{\sigma(id_t)(\text{"}A\text{"}) \leq \sigma(id_t)(\text{"}stc\text{"}) + 1}$ (by $s.I(t) = \sigma(id_t)(\text{"}stc\text{"})$)

$\Rightarrow$ $\sigma(id_t)(\text{"}A\text{"}) - 1 \leq \sigma(id_t)(\text{"}stc\text{"})$ (assumption)

- **CASE** $s.I(t) > lower(I_s(t))$:

  By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, we have $\sigma(id_t)(\text{"}stc\text{"}) = lower(I_s(t)) = a$.
  From $\sigma(id_t)(\text{"}se\text{"}) = \texttt{true}$, we can deduce $t \in Sens(s.M)$, and from $\sigma(id_t)(\text{"}srtc\text{"}) = \texttt{false}$, we can deduce $s.reset_t(t) = \texttt{false}$. Then, by definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), we have $s'.I(t) = s.I(t) + 1$.

  $\Rightarrow$ $\boxed{\sigma(id_t)(\text{"}A\text{"}) \leq s.I(t) + 1}$ (by $s'.I(t) = s.I(t) + 1$)

  $\Rightarrow$ $\boxed{a \leq s.I(t) + 1}$ (by $\sigma(id_t)(\text{"}A\text{"}) = a$)

  $\Rightarrow$ $\boxed{a < s.I(t)}$

  $\Rightarrow$ $lower(I_s(t)) < s.I(t)$ (assumption)

(b) $\left(\sigma(id_t)(\text{"}srtc\text{"}) \,.\, \Delta(id_t)(\text{"}tt\text{"}) \neq \texttt{NOT\_TEMP} \,.\, \sigma(id_t)(\text{"}A\text{"}) = 1\right) = \texttt{true}$

Then, converting boolean equalities into intuitionistic predicates, we have:

- $\sigma(id_t)(\text{"}srtc\text{"}) = \texttt{true}$
- $\Delta(id_t)(\text{"}tt\text{"}) \neq \texttt{NOT\_TEMP}$
- $\sigma(id_t)(\text{"}A\text{"}) = 1$

By property of the elaboration relation, and $\Delta(id_t)(\text{"}tt\text{"}) \neq \texttt{NOT\_TEMP}$, there exist an $a \in \mathbb{N}^*$ and a $ni \in \mathbb{N}^* \sqcup \{\infty\}$ s.t. $I_s(t) = [a, ni]$. Let us take such an a and ni.

By construction, $<\texttt{time\_A\_value} \Rightarrow a> \in ipm_t$, and by property of stable σ, we have $\sigma(id_t)(\text{"}A\text{"}) = a$. Thus, we can deduce $a = 1$ and $I_s(t) = [1, ni]$.

By definition of $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$, from $\sigma(id_t)(\text{"}se\text{"}) = \texttt{true}$, we can deduce $t \in Sens(s.M)$, and from $\sigma(id_t)(\text{"}srtc\text{"}) = \texttt{true}$, we can deduce $s.reset_t(t) = \texttt{true}$.

By definition of $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$ (Rule **??**), $t \in Sens(s.M)$ and $s.reset_t(t) = \texttt{true}$, we have $s'.I(t) = 1$.

Now, let us show $\boxed{s'.I(t) \in I_s(t)}$.

Rewriting the goal with $s'.I(t) = 1$ and $I_s(t) = [1, ni]$: $1 \in [1, ni]$.

(c) $\left(\Delta(id_t)(\text{"}tt\text{"}) = \texttt{NOT\_TEMP}\right) = \texttt{true}$

Let us show $\boxed{t \notin T_i}$.

By property of the elaboration relation and $\Delta(id_t)(\text{"}tt\text{"}) = \texttt{NOT\_TEMP}$, we have $t \notin T_i$.

□

Lemma 42 (Falling edge equal not firable). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then $\forall t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $t \notin Firable(s') \Leftrightarrow \sigma'(id_t)(\text{"s\_firable"}) = \texttt{false}$.*

Proof. Proving the above lemma is trivial by appealing to Lemma 39 and by reasoning on contrapositives. □

A.4.7 Falling edge and fired transitions

Lemma 43 (Falling edge equal fired set). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then $\forall t \in T$, $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $\forall Fset \subseteq T$, s.t. $IsFiredSet(s', Fset)$, $t \in Fset \Leftrightarrow \sigma'(id_t)(\mathit{fired}) = \texttt{true}$.*

Proof. Given a $t \in T$, and $id_t \in Comps(\Delta)$, and a $Fset \subseteq T$ s.t. $IsFiredSet(s', Fset)$, let us show
$\boxed{t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}.}$

By definition of $IsFiredSet(s', Fset)$, we have $IsFiredSetAux(s', T, \emptyset, Fset)$.
Then, we can appeal to Lemma 44 to solve the goal, but first we must prove the following *extra hypothesis* (i.e, one of the premise of Lemma 44):

$\boxed{\begin{array}{l}\forall t' \in T, id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in \emptyset \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in \emptyset \ \vee \ t' \in T).\end{array}}$

Given a $t' \in T$ and an $id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$, there are two points to prove:

1. $\boxed{t' \in \emptyset \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}}$
2. $\boxed{\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in \emptyset \ \vee \ t' \in T}$

Let us show these two points:

1. Assuming $t' \in \emptyset$, let us show $\boxed{\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}.}$

   $t' \in \emptyset$ is a contradiction.

2. Assuming $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$, let us show $\boxed{t' \in \emptyset \ \vee \ t' \in T.}$

   By definition, $t' \in T$.

□

Lemma 44 (Falling edge equal fired set aux). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then $\forall t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $\forall F \subseteq T$, $T_s \subseteq T$, $Fset \subseteq T$, assume that:*

- $IsFiredSetAux(s', T_s, F, Fset)$
- *EH (Extra. Hypothesis):*
  $\forall t' \in T$, $id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
  $(t' \in F \Rightarrow \sigma'(id_{t'})(\mathit{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\mathit{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s)$.

then $t \in Fset \Leftrightarrow \sigma'(id_t)(\mathit{fired}) = \texttt{true}$.

Proof. Given a $t \in T$, an $id_t \in Comps(\Delta)$, a $T_s, F, Fset \subseteq T$, and assuming $IsFiredSetAux(s', T_s, F, Fset)$, let us show

$$\begin{array}{l}(\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s)) \Rightarrow\\ t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}.\end{array}$$

Let us use rule induction on $IsFiredSetAux(s', T_s, F, Fset)$. Let us define the property P taken into account in the induction scheme as follows

$$\begin{array}{c}P(s', T_s, F, Fset)\\ \equiv\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s)) \Rightarrow\\ t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}\end{array}$$

- **CASE** FSETEMP: we must show $P(s', \emptyset, F, F)$, i.e.

  $$\begin{array}{l}(\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in \emptyset)) \Rightarrow\\ t \in F \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}.\end{array}$$

  Assuming

  $$\begin{array}{l}\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in \emptyset)\end{array}$$

  we can easily show $t \in F \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}$.

- **CASE** FSETFIRED:

  Assuming

  - $t \in Firable(s')$
  - $t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$
  - $IsFiredSetAux(s', T_s, F \cup \{t\}, Fset)$
  - $\nexists t' \in T_s \text{ s.t. } t' \succ t$
  - $Pr(t, F) = \{t' \mid t' \succ t \wedge t' \in F\}$

  and the induction hypothesis (i.e. $P(s', T_s, F \cup \{t\}, Fset)$)

$$\begin{array}{l}(\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \cup \{t\} \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true})\\ \wedge\ (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \cup \{t\}\ \vee\ t' \in T_s)) \Rightarrow\\ t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}\end{array}$$

we must show

$$\boxed{\begin{array}{l}(\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true})\\ \wedge\ (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F\ \vee\ t' \in T_s \cup \{t\})) \Rightarrow\\ t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}\end{array}}$$

Assuming the following hypothesis that we will call EH (for Extra Hypothesis)

$$\begin{array}{l}\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F\ \vee\ t' \in T_s \cup \{t\})\end{array}$$

we must show

$$\boxed{t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}}$$

Appealing to the induction hypothesis, to prove the current goal, it is sufficient to prove that

$$\boxed{\begin{array}{l}\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \cup \{t\} \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true})\\ \wedge\ (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \cup \{t\}\ \vee\ t' \in T_s)\end{array}}$$

Given a $t' \in T$, an $id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$, we must show that

$$\boxed{\begin{array}{l}(t' \in F \cup \{t\} \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true})\\ \wedge\ (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \cup \{t\}\ \vee\ t' \in T_s)\end{array}}$$

There are two points to prove

1. Assuming $t' \in F \cup \{t\}$, then $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$
2. Assuming $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$, then $t' \in F \cup \{t\}\ \vee\ t' \in T_s$

1. Assuming $t' \in F \cup \{t\}$, let us show $\boxed{\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}}$. Let us perform case analysis on $t' \in F \cup \{t\}$; there are 2 cases:
   - **CASE** $t' \in F$: Appealing to EH, the goal is trivially proved.

– **CASE** $t' = t$: Then, $id_t = id_{t'}$, and we must show $\boxed{\sigma'(id_t)(\texttt{fired}) = \texttt{true}}$.
By definition of id_t, there exist a g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
By property of the stabilize relation and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `fired_evaluation` process defined in the `transition` design architecture:

$$\sigma(id_t)(\texttt{fired}) = \sigma(id_t)(\texttt{sfa}) \,.\, \sigma(id_t)(\texttt{spc})$$

Rewriting the goal with the above equation: $\boxed{\sigma(id_t)(\texttt{sfa}) \,.\, \sigma(id_t)(\texttt{spc}) = \texttt{true}.}$
Then, there are two points to prove:

(a) $\boxed{\sigma(id_t)(\texttt{sfa}) = \texttt{true}.}$
Appealing to Lemma 39, and since $t \in Firable(s')$, we can deduce $\sigma(id_t)(\texttt{sfa}) = \texttt{true}$.

(b) $\boxed{\sigma(id_t)(\texttt{spc}) = \texttt{true}.}$
Appealing to Lemma 45, and since $t \in Sens(s'M - \sum_{t_i \in Pr(t,F)} pre(t_i))$, we can deduce $\sigma(id_t)(\texttt{spc}) = \texttt{true}$.

2. Assuming $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$, let us show $\boxed{t' \in F \cup \{t\} \ \vee \ t' \in T_s}$. Appealing to EH, we can deduce that $t' \in F \ \vee \ t' \in T_s \cup \{t\}$. Then, the goal is trivially shown.

- **CASE** FSetNotFirable: Assuming

– $t \notin Firable(s')$
– $IsFiredSetAux(s', T_s, F, Fset)$
– $\nexists t' \in T_s \ s.t. \ t' \succ t$

and the induction hypothesis (i.e. $P(s', T_s, F, Fset)$)

$$\begin{array}{l}(\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true})\\ \wedge\ (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s)) \Rightarrow\\ t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}\end{array}$$

we must show

$$\begin{array}{l}(\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true})\\ \wedge\ (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s \cup \{t\})) \Rightarrow\\ t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}\end{array}$$

Assuming the following hypothesis that we will call EH (for Extra Hypothesis)

$\forall t' \in T,\ id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
$(t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s \cup \{t\})$

we must show

$$t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}$$

Appealing to the induction hypothesis, to prove the current goal, it is sufficient to prove that

$\forall t' \in T,\ id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
$(t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s)$

Given a $t' \in T$, an $id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$, we must show that

$(t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s)$

There are two points to prove

1. Assuming $t' \in F$, then $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$
2. Assuming $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$, then $t' \in F \ \vee \ t' \in T_s$

1. Assuming $t' \in F$, let us show $\boxed{\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}}$.
   Appealing to EH, the goal is trivially shown.
2. Assuming $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$, let us show $\boxed{t' \in F \ \vee \ t' \in T_s}$.
   Appealing to EH, we can deduce $t' \in F \ \vee \ t' \in T_s \cup \{t\}$. Let us perform case analysis on $t' \in F \ \vee \ t' \in T_s \cup \{t\}$; there are 2 cases:
   - **CASE** $t' \in F$: trivially shown, as it is an assumption.
   - **CASE** $t' \in T_s \cup \{t\}$: In the case where $t' \in T_s$, the goal is trivially shown. In the case where $t' = t$, we can prove a contradiction based on $t \notin Firable(s')$ and $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$.
     Since $t = t'$, then $id_t = id_{t'}$, and we know that $\sigma'(id_t)(\texttt{fired}) = \texttt{true}$.
     By definition of id_t, there exist a g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
     By property of the stabilize relation and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `fired_evaluation` process defined in the `transition` design architecture, we can deduce

     $$\sigma(id_t)(\texttt{fired}) = \sigma(id_t)(\texttt{sfa}) \,.\, \sigma(id_t)(\texttt{spc}) = \texttt{true}$$

     Thus, we have

     $$\sigma(id_t)(\texttt{sfa}) = \texttt{true}$$

     and, appealing to Lemma 39, we can deduce $t \in Firable(s')$, which directly contradicts $t \notin Firable(s')$.

- **CASE** FSETNOTSENS: Assuming
  - $t \notin Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$
  - $IsFiredSetAux(s', T_s, F, Fset)$
  - $\nexists t' \in T_s$ *s.t.* $t' \succ t$
  - $Pr(t, F) = \{t' \mid t' \succ t \wedge t' \in F\}$

  and the induction hypothesis (i.e. $P(s', T_s, F, Fset)$)

  $$\begin{array}{l}(\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true})\\ \wedge\ (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F\ \vee\ t' \in T_s)) \Rightarrow\\ t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}\end{array}$$

  we must show

  $$\begin{array}{l}(\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true})\\ \wedge\ (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F\ \vee\ t' \in T_s \cup \{t\})) \Rightarrow\\ t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}\end{array}$$

  Assuming the following hypothesis, which we will call EH (for Extra Hypothesis)

  $$\begin{array}{l}\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F\ \vee\ t' \in T_s \cup \{t\})\end{array}$$

  we must show

  $$t \in Fset \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{true}$$

  Appealing to the induction hypothesis, to prove the current goal, it is sufficient to prove that

  $$\begin{array}{l}\forall t' \in T,\ id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F\ \vee\ t' \in T_s)\end{array}$$

  Given a $t' \in T$, an $id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$, we must show that

$$(t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s)$$

There are two points to prove

1. Assuming $t' \in F$, then $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$
2. Assuming $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$, then $t' \in F \ \vee \ t' \in T_s$

1. Assuming $t' \in F$, let us show $\boxed{\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}}$.
   Appealing to EH, the goal is trivially shown.
2. Assuming $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$, let us show $\boxed{t' \in F \ \vee \ t' \in T_s}$.
   Appealing to EH, we can deduce $t' \in F \ \vee \ t' \in T_s \cup \{t\}$. Let us perform case analysis on $t' \in F \ \vee \ t' \in T_s \cup \{t\}$; there are 2 cases:
   - **CASE** $t' \in F$: trivially shown, as it is an assumption.
   - **CASE** $t' \in T_s \cup \{t\}$: In the case where $t' \in T_s$, the goal is trivially shown. In the case where $t' = t$, we can prove a contradiction based on $t \notin Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$ and $\sigma'(id_{t'})(\texttt{fired}) = \texttt{true}$.
     Since $t = t'$, then $id_t = id_{t'}$, and we know that $\sigma'(id_t)(\texttt{fired}) = \texttt{true}$.
     By definition of id_t, there exist a g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
     By property of the stabilize relation and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `fired_evaluation` process defined in the `transition` design architecture, we can deduce

     $$\sigma(id_t)(\texttt{fired}) = \sigma(id_t)(\texttt{sfa}) \,.\, \sigma(id_t)(\texttt{spc}) = \texttt{true}$$

     Thus, we have

     $$\sigma(id_t)(\texttt{spc}) = \texttt{true}$$

     and, appealing to Lemma 45, we can deduce $t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$, which directly contradicts $t \notin Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$.

□

Lemma 45 (Stabilize compute priority combination after falling edge)**.** *For all $sitpn$, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then $\forall t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $\forall T_s, F$, $Fset \subseteq T$ assume that:*

- $t \in Firable(s')$
- $\nexists t' \in T_s$ *s.t.* $t' \succ t$
- *EH:* $\forall t' \in T, id_{t'} \in Comps(\Delta)$ *s.t.* $\gamma(t') = id_{t'}$,
  $(t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \ \vee \ t' \in T_s)$.

then $t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i)) \Leftrightarrow \sigma'(id_t)(spc) = \texttt{true}$

Proof. Given a $t \in T$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, a T_s, F, $Fset \subseteq T$ and assuming

- $t \in Firable(s')$
- $\nexists t' \in T_s \; s.t. \; t' \succ t$
- EH: $\forall t' \in T, id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
  $(t' \in F \Rightarrow \sigma'(id_{t'})(\texttt{fired}) = \texttt{true}) \wedge (\sigma'(id_{t'})(\texttt{fired}) = \texttt{true} \Rightarrow t' \in F \; \vee \; t' \in T_s)$.

let us show

$$t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i)) \Leftrightarrow \sigma'(id_t)(\texttt{spc}) = \texttt{true}.$$

By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
By property of the stabilize relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$, and through the examination of the `priority_authorization_evaluation` process defined in the `transition` design architecture, we can deduce:

$$\sigma'(id_t)(''spc'') = \prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i]$$

Rewriting the goal with the above equation:

$$t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i)) \Leftrightarrow \prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}.$$

Then, the proof is in two parts:

1. $t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i)) \Rightarrow \prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}$
2. $\prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i] = \texttt{true} \Rightarrow t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$

Let us prove both sides of the equivalence:

1. Assuming that $t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$, let us show

   $$\prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}.$$

   Let us perform case analysis on $input(t)$; there are 2 cases:

- **CASE** $input(t) = \emptyset$:
  By construction, $<$`input_arcs_number` $\Rightarrow 1>$ $\in g_t$ and $<$`priority_authorizations(0)` $\Rightarrow$ `true`$> \in i_t$.
  By property of the elaboration relation, we have $\Delta(id_t)(\texttt{ian}) = 1$, and by property of the stabilize relation, we have $\sigma'(id_t)(\texttt{pauths})[0] = \texttt{true}$.
  Rewriting the goal with $\Delta(id_t)(\texttt{ian}) = 1$ and $\sigma'(id_t)(\texttt{pauths})[0] = \texttt{true}$, and simplifying the goal: tautology.
- **CASE** $input(t) \neq \emptyset$:
  Then, let us show an equivalent goal:
  $\forall i \in [0, \Delta(id_t)(\texttt{ian}) - 1],\ \sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}.$
  Given an $i \in [0, \Delta(id_t)(\texttt{ian}) - 1]$, let us show $\sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}.$
  By construction, $<$`input_arcs_number` $\Rightarrow |input(t)|> \in g_t$.
  By property of the elaboration relation, we have $\Delta(id_t)(\texttt{ian}) = |input(t)|$. Then, we can deduce $i \in [0, |input(t)| - 1]$.
  By construction, for all $i \in [0, |input(t)| - 1]$, there exist a $p \in input(t)$ and an $id_p \in Comps(\Delta)$ s.t. $\gamma(p) = id_p$, there exist a g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and there exist a $j \in [0, |output(p)|]$ and an $id_{ji} \in Sigs(\Delta)$ s.t.
  $<$`input_arcs_valid(i)` $\Rightarrow$ $\texttt{id}_{\texttt{ji}}> \in i_t$ and $<$`output_arcs_valid(j)` $\Rightarrow$ $\texttt{id}_{\texttt{ji}}> \in o_t$.
  Let us take such a $p \in input(t)$, $id_p \in Comps(\Delta)$, g_p, i_p, o_p, $j \in [0, |output(p)|]$ and $id_{ji} \in Sigs(\Delta)$.

  Now, let us perform case analysis on the nature of the arc connecting p and t; there are 2 cases:

  – **CASE** $pre(p,t) = (\omega, \texttt{test})$ or $pre(p,t) = (\omega, \texttt{inhib})$:
    By construction, $<$`priority_authorizations(i)` $\Rightarrow$ `true`$> \in i_t$, and by property of the stabilize relation: $\sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}.$
  – **CASE** $pre(p,t) = (\omega, \texttt{basic})$:
    Let us define $output_c(p) = \{t \in T \mid \exists \omega,\ pre(p,t) = (\omega, \texttt{basic})\}$, the set of output transitions of p that are in conflict. Then, there are two cases, one for each way to solve the conflicts between the output transitions of p:
    * **CASE** For all pair of transitions in $output_c(p)$, all conflicts are solved by mutual exclusion:
      By construction, $<$`priority_authorizations(i)` $\Rightarrow$ `true`$> \in i_t$, and by property of the stabilize relation: $\sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}.$
    * **CASE** The priority relation is a strict total order over the set $output_c(p)$:

    By construction, there exists an $id'_{ji} \in Sigs(\Delta)$ s.t.
    $<$`priority_authorizations(i)` $\Rightarrow$ $\texttt{id}'_{\texttt{ji}}> \in i_t$ and
    $<$`priority_authorizations(j)` $\Rightarrow$ $\texttt{id}'_{\texttt{ji}}> \in o_p$.

By property of the stabilize relation, $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$ and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we can deduce:

$$\sigma'(id_t)(\texttt{pauths})[i] = \sigma'(id'_{ji}) = \sigma'(id_p)(\texttt{pauths})[j]$$

Rewriting the goal with the above equation: $\boxed{\sigma'(id_p)(\texttt{pauths})[j] = \texttt{true}.}$
By property of the stabilize relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, and through the examination of the `priority_evaluation` process defined in the `place` design behavior, we can deduce:

$$\sigma'(id_p)(\texttt{pauths})[j] = (\sigma'(id_p)(\texttt{sm}) \geq \texttt{vsots} + \sigma'(id_p)(\texttt{oaw})[j]) \tag{A.48}$$

Let us define the `vsots` term as follows:

$$\texttt{vsots} = \sum_{i=0}^{j-1} \begin{cases} \sigma'(id_p)(\texttt{oaw})[i] \texttt{ if } \sigma'(id_p)(\texttt{otf})[i]. \\ \qquad\qquad\qquad\qquad \sigma'(id_p)(\texttt{oat})[i] = \texttt{basic} \\ 0 \textit{ otherwise} \end{cases} \tag{A.49}$$

Rewriting the goal with (A.48): $\boxed{\sigma'(id_p)(\texttt{sm}) \geq \texttt{vsots} + \sigma'(id_p)(\texttt{oaw})[j]}$
By definition of $t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$, we can deduce:
$s'.M(p) \geq \sum_{t_i \in Pr(t,F)} pre(p, t_i) + \omega$.
Then, there are three points to prove:

(a) $\boxed{s'.M(p) = \sigma'(id_p)(\texttt{sm})}$

(b) $\boxed{\omega = \sigma'(id_p)(\texttt{oaw})[j]}$

(c) $\boxed{\sum_{t_i \in Pr(t,F)} pre(p, t_i) = \texttt{vsots}}$

Let us prove these three points:

(a) $\boxed{s'.M(p) = \sigma'(id_p)(\texttt{sm})}$
Appealing to Lemma 32, $s'.M(p) = \sigma'(id_p)(\texttt{sm})$.

(b) $\boxed{\omega = \sigma'(id_p)(\texttt{oaw})[j]}$
By construction, and as $pre(p,t) = (\omega, \texttt{basic})$, we know that $<$`output_arcs_weights(j)`$\Rightarrow \omega > \in i_p$.
By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$:
$\omega = \sigma'(id_p)(\texttt{oaw})[j]$.

(c) $\boxed{\sum_{t_i \in Pr(t,F)} pre(p, t_i) = \texttt{vsots}}$

Let us replace the left and right term of the equality by their full definition:

$$\sum_{t_i \in Pr(t,F)} \begin{cases} \omega \texttt{ if } pre(p,t_i) = (\omega, \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases} = \sum_{i=0}^{j-1} \begin{cases} \sigma'(id_p)(\texttt{oaw})[i] \texttt{ if } \sigma'(id_p)(\texttt{otf})[i]. \\ \qquad\qquad\qquad \sigma'(id_p)(\texttt{oat})[i] = \texttt{basic} \\ 0 \textit{ otherwise} \end{cases}$$

Now, we must reason on the priority status of transition t regarding the group of conflicting output transitions of p. There 2 cases:

* **CASE** t is the top-priority transition in the group of conflicting output transitions of p:
  In that case, the set $Pr(t,F)$ is empty and, by construction, $j = 0$. Thus, the goal is a tautology $0 = 0$.
* **CASE** t is not the top-priority transition in the group of conflicting output transitions of p:

In that case, we know that there is a least one element in $Pr(t,F)$ and the index $j > 0$. Let us replace the sum terms in the goal by equivalent terms:

$$\sum_{t_i \in Pr_p} \begin{cases} \omega \texttt{ if } pre(p,t_i) = (\omega, \texttt{basic}) \texttt{ and } t_i \in F \\ 0 \textit{ otherwise} \end{cases} = \sum_{i \in IPr_p} \begin{cases} \sigma'(id_p)(\texttt{oaw})[i] \texttt{ if } \sigma'(id_p)(\texttt{otf})[i] \\ 0 \textit{ otherwise} \end{cases}$$

Let us define the set Pr_p as

$$Pr_p = \{t_i \mid t_i \succ t \wedge \exists \omega \textit{ s.t. } pre(p,t_i) = (\omega, \texttt{basic})\}$$

and set IPr_p as

$$IPr_p = \{i \mid i \in [0, j-1] \wedge \sigma'(id_p)(\texttt{oat})[i] = \texttt{basic}\}$$

Let us define $f(t_i)$ as

$$f(t_i) = \begin{cases} \omega \texttt{ if } pre(p,t_i) = (\omega, \texttt{basic}) \texttt{ and } t_i \in F \\ 0 \textit{ otherwise} \end{cases}$$

and $g(i)$ as

$$g(i) = \begin{cases} \sigma'(id_p)(\texttt{oaw})[i] \texttt{ if } \sigma'(id_p)(\texttt{otf})[i] \\ 0 \textit{ otherwise} \end{cases}$$

then, we must prove $\boxed{\sum_{t_i \in Pr_p} f(t_i) = \sum_{i \in IPr_p} g(i).}$

To prove the above equality, it is sufficient to prove that there exists a bijection β from Pr_p to IPr_p such that for all $t_i \in Pr_p$, $f(t_i) = g(\beta(t_i))$. Let us use the function β that

takes a $t_i \in Pr_p$ and yields the index denoting the position of t_i in the priority-ordered version of set Pr_p. We assumed that a total order existed over the conflicting output transitions of place p, then there exists a total ordering of the transitions of set Pr_p, i.e. the conflicting output transitions of place p with a higher priority than t. By property of the HILECOP transformation function, we know that the index returned by the function β belongs to the interval $[0, j-1]$ and verifies $\sigma'(id_p)(\texttt{oat})[i] = \texttt{basic}$. Given a $t_i \in Pr_p$, we must show $\boxed{f(t_i) = g(\beta(t_i)).}$
Let us unfold terms $f(t_i)$ and $g(\beta(t_i))$ to their full definition:

$$\begin{cases} \omega \texttt{ if } pre(p,t_i) = (\omega, \texttt{basic}) \texttt{ and } t_i \in F \\ 0 \textit{ otherwise} \end{cases}$$
$$=$$
$$\begin{cases} \sigma'(id_p)(\texttt{oaw})[\beta(t_i)] \texttt{ if } \sigma'(id_p)(\texttt{otf})[\beta(t_i)] \\ 0 \textit{ otherwise} \end{cases}$$

By construction, there exists an $id_{t_i} \in Comps(\Delta)$ such that $\gamma(t_i) = id_{t_i}$, and there exist g_{t_i}, i_{t_i} and o_{t_i} such that $\texttt{comp}(id_{t_i}, \texttt{transition}, g_{t_i}, i_{t_i}, o_{t_i}) \in d.cs$.

By property of the function β and by construction, we can deduce that the element of index $\beta(t_i)$ of the `otf` input port of PCI id_p is connected the `fired` output port of TCI id_{t_i}. Thus, there exists an $id_{\beta i} \in Sigs(\Delta)$ s.t. $<\texttt{otf}(\beta(t_i)) \Rightarrow \texttt{id}_{\beta i}> \in i_p$ and $<\texttt{fired} \Rightarrow \texttt{id}_{\beta i}> \in o_{t_i}$.
By property of the stabilize relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$ and $\texttt{comp}(id_{t_i}, \texttt{transition}, g_{t_i}, i_{t_i}, o_{t_i}) \in d.cs$, we have

$$\sigma'(id_{t_i})(\texttt{fired}) = \sigma'(id_{\beta i}) = \sigma'(id_p)(\texttt{otf})[\beta(t_i)]$$

then, we can rewrite the goal with the above equation

$$\begin{cases} \omega \texttt{ if } pre(p,t_i) = (\omega, \texttt{basic}) \texttt{ and } t_i \in F \\ 0 \textit{ otherwise} \end{cases}$$
$$=$$
$$\begin{cases} \sigma'(id_p)(\texttt{oaw})[\beta(t_i)] \texttt{ if } \sigma'(id_{t_i})(\texttt{fired}) \\ 0 \textit{ otherwise} \end{cases}$$

By property of the function β and by construction, we can deduce that the element of index $\beta(t_i)$ of the `oaw` input port of PCI id_p is connected to a constant value denoting the weight of the arc between place p and transition t_i. Thus, we have

$$<\texttt{oaw}(\beta(t_i)) \Rightarrow \omega> \in i_p \text{ where } pre(p,t_i) = (\omega, \texttt{basic})$$

By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$, we have

$$\sigma'(id_p)(\texttt{oaw})[\beta(t_i)] = \omega$$

then, we can rewrite the goal with the above equation

$$\begin{cases} \omega \texttt{ if } pre(p,t_i) = (\omega, \texttt{basic}) \texttt{ and } t_i \in F \\ 0 \textit{ otherwise} \end{cases} = \begin{cases} \omega \texttt{ if } \sigma'(id_{t_i})(\texttt{fired}) \\ 0 \textit{ otherwise} \end{cases}$$

Finally, proving the goal comes down to proving

$$t_i \in F \Leftrightarrow \sigma'(id_{t_i})(\texttt{fired}) = \texttt{true}$$

Let us prove both sense of the equivalence:

(a) Assuming $t_i \in F$, let us show $\boxed{\sigma'(id_{t_i})(\texttt{fired}) = \texttt{true}}$.
Appealing to EH, proving the goal is trivial.

(b) Assuming $\sigma'(id_{t_i})(\texttt{fired}) = \texttt{true}$, let us show $\boxed{t_i \in F}$.
Appealing to EH, we have $t_i \in F \vee t_i \in T_s$. There are two cases: either $t_i \in F$ or $t_i \in T$. In the case where $t_i \in T$, we can show a contradiction with the fact that t is a top-priority transition in set T_s. By definition, transition t_i has a higher firing priority than t, and thus, if t_i belongs to set T_s, then t is no longer a top-priority transition of set T_s; whence the contradiction.

2. Assuming that $\prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}$, let us show

$$\boxed{t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i)).}$$

By definition of $t \in Sens(s'.M - \sum_{t_i \in Pr(t,F)} pre(t_i))$:

$$\begin{aligned} &\forall p \in P, \omega \in \mathbb{N}^*, \\ &\left((pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})) \Rightarrow s'.M(p) - \sum_{t_i \in Pr(t,F)} pre(p,t_i) \geq \omega\right) \\ &\wedge \left(pre(p,t) = (\omega, \texttt{inhib}) \Rightarrow s'.M(p) - \sum_{t_i \in Pr(t,F)} pre(p,t_i) < \omega\right) \end{aligned}$$

Given a $p \in P$ and an $\omega \in \mathbb{N}^*$, let us show

$$\begin{aligned} &\left((pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})) \Rightarrow s'.M(p) - \sum_{t_i \in Pr(t,F)} pre(p,t_i) \geq \omega\right) \\ &\wedge \left(pre(p,t) = (\omega, \texttt{inhib}) \Rightarrow s'.M(p) - \sum_{t_i \in Pr(t,F)} pre(p,t_i) < \omega\right) \end{aligned}$$

By construction, there exists an $id_p \in Comps(\Delta)$ s.t. $\gamma(p) = id_p$. By construction and by definition of id_p, there exist g_p, i_p, o_p s.t. $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$.

To prove the goal, there are different cases:

(a) Assuming that $pre(p,t) = (\omega, \texttt{test})$, let us show $\boxed{s'.M(p) - \sum_{t_i \in Pr(t,F)} pre(p,t_i) \geq \omega.}$

Then, assuming that the priority relation is well-defined, there exists no transition t_i connected by a `basic` arc to p that verifies $t_i \succ t$. This is because t is connected to p by a `test` arc; thus, t is not in conflict with the other output transitions of p; thus, there is no relation of priority between t and the other output transitions of p.
Then, we can deduce that $\sum_{t_i \in Pr(t,F)} pre(p,t_i) = 0$.
Then, the new goal is $s'.M(p) \geq \omega$.
Knowing that $t \in Firable(s')$, thus, $t \in Sens(s'.M)$, thus, we have $s'.M(p) \geq \omega$.

(b) Assuming that $pre(p,t) = (\omega, \texttt{inhib})$, let us show $\boxed{s'.M(p) - \sum_{t_i \in Pr(t,F)} pre(p,t_i) < \omega.}$

Use the same strategy as above.

(c) Assuming that $pre(p,t) = (\omega, \texttt{basic})$, let us show $\boxed{s'.M(p) - \sum_{t_i \in Pr(t,F)} pre(p,t_i) \geq \omega.}$

Then, there are two cases:

i. **CASE** For all pair of transitions in $output_c(p)$, all conflicts are solved by mutual exclusion.
Then, assuming that the priority relation is well-defined, it must not be defined over the set $output_c(t)$, and we know that $t \in output_c(p)$ since $pre(p,t) = (\omega, \texttt{basic})$.
Then, there exists no transition t_i connected to p by a `basic` arc that verifies $t_i \succ t$.
Then, we can deduce $\sum_{t_i \in Pr(t,F)} pre(p,t_i) = 0$.
Then, the new goal is $s'.M(p) \geq \omega$.
We know $t \in Firable(s')$, thus, $t \in Sens(s'.M)$, thus, $s'.M(p) \geq \omega$.

ii. **CASE** The priority relation is a strict total order over the set $output_c(p)$.
By construction, there exists $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$. By construction and by definition of id_t, there exist g_t, i_t, o_t s.t. $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$.
By construction, there exist $j \in [0, |input(t)| - 1]$, $k \in [0, |output(t)| - 1]$, and $id_{kj} \in Sigs(\Delta)$ s.t. $<\texttt{priority\_authorizations(j)} \Rightarrow \texttt{id}_{\texttt{kj}}> \in i_t$ and $<\texttt{priority\_authorizations(k)} \Rightarrow \texttt{id}_{\texttt{kj}}> \in o_p$. Let us take such an j, k and id_{kj}.
From $\prod_{i=0}^{\Delta(id_t)(\texttt{ian})-1} \sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}$, we can deduce that for all $i \in [0, \Delta(id_t)(\texttt{ian}) - 1]$, $\sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}$.
By construction, $<\texttt{input\_arcs\_number} \Rightarrow |input(t)|> \in g_t$, and by property of the elaboration relation, we have $\Delta(id_t)(\texttt{ian}) = |input(t)|$. Then, from $j \in [0, |input(t)| -$

1], we can deduce $j \in [0, \Delta(id_t)(\texttt{ian}) - 1]$. And, from $\forall i \in [0, \Delta(id_t)(\texttt{ian}) - 1]$, $\sigma'(id_t)(\texttt{pauths})[i] = \texttt{true}$, we can deduce $\sigma'(id_t)(\texttt{pauths})[j] = \texttt{true}$.
By property of the stabilize relation, $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$ and $\texttt{comp}(id_t, \texttt{transition}, g_t, i_t, o_t) \in d.cs$:

$$\sigma'(id_p)(\texttt{pauths})[k] = \sigma'(id_{kj}) = \sigma'(id_t)(\texttt{pauths})[j] = \texttt{true} \tag{A.50}$$

By property of the stabilize relation and $\texttt{comp}(id_p, \texttt{place}, g_p, i_p, o_p) \in d.cs$:

$$\sigma'(id_p)(\texttt{pauths})[k] = (\sigma'(id_p)(\texttt{sm}) \geq \texttt{vsots} + \sigma'(id_p)(\texttt{oaw})[k]) \tag{A.51}$$

Let us define the vsots term as follows:

$$\texttt{vsots} = \sum_{i=0}^{k-1} \begin{cases} \sigma'(id_p)(oaw)[i] \texttt{ if } \sigma'(id_p)(\texttt{otf})[i]. \\ \qquad\qquad\qquad\qquad \sigma'(id_p)(\texttt{oat})[i] = \texttt{basic} \\ 0 \textit{ otherwise} \end{cases} \tag{A.52}$$

From (A.50) and (A.51), we can deduce that $\sigma'(id_p)(\texttt{sm}) \geq \texttt{vsots} + \sigma'(id_p)(\texttt{oaw})[k]$.
Then, there are three points to prove:

A. $\boxed{s'.M(p) = \sigma'(id_p)(\texttt{sm})}$

B. $\boxed{\omega = \sigma'(id_p)(\texttt{oaw})[k]}$

C. $\boxed{\sum_{t_i \in Pr(t,F)} pre(p, t_i) = \texttt{vsots}}$

See 1 for the remainder of the proof.

□

Lemma 46 (Falling edge equal not fired). *For all sitpn, d, γ, Δ, σ_e, E_c, E_p, τ, s, s', σ, σ_i, $\sigma_\downarrow$, σ' that verify the hypotheses of Definition 10, then $\forall t, id_t$ s.t. $\gamma(t) = id_t$, $t \notin Fired(s') \Leftrightarrow \sigma'(id_t)(\texttt{fired}) = \texttt{false}$.*

Proof. Proving the above lemma is trivial by appealing to Lemma Falling edge equal fired and by reasoning on contrapositives. □

Bibliography

[1] Karima Berramla, El Abbassia Deba, and Mohammed Senouci. "Formal Validation of Model Transformation with Coq Proof Assistant". In: *2015 First International Conference on New Technologies of Information and Communication (NTIC)*. 2015 First International Conference on New Technologies of Information and Communication (NTIC). Nov. 2015, pp. 1–6. DOI: 10.1109/NTIC.2015.7368755.

[2] Yves Bertot and P. Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Berlin ; New York: Springer, 2004. 469 pp. ISBN: 978-3-540-20854-9.

[3] Sandrine Blazy, Zaynah Dargaye, and Xavier Leroy. "Formal Verification of a C Compiler Front-End". In: *FM 2006: Formal Methods*. International Symposium on Formal Methods. Springer, Berlin, Heidelberg, Aug. 21, 2006, pp. 460–475. DOI: 10.1007/11813040_31. URL: https://link.springer.com/chapter/10.1007/11813040_31 (visited on 05/25/2020).

[4] Thomas Bourgeat et al. "The Essence of Bluespec: A Core Language for Rule-Based Hardware Design". In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2020. New York, NY, USA: Association for Computing Machinery, June 11, 2020, pp. 243–257. ISBN: 978-1-4503-7613-6. DOI: 10.1145/3385412.3385965. URL: https://doi.org/10.1145/3385412.3385965 (visited on 05/05/2021).

[5] Timothy Bourke et al. "A Formally Verified Compiler for Lustre". In: (), p. 17.

[6] Thomas Braibant and Adam Chlipala. "Formal Verification of Hardware Synthesis". In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 213–228. ISBN: 978-3-642-39799-8. DOI: 10.1007/978-3-642-39799-8_14.

[7] Daniel Calegari et al. "A Type-Theoretic Framework for Certified Model Transformations". In: *Formal Methods: Foundations and Applications*. Ed. by Jim Davies, Leila Silva, and Adenilso Simao. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 112–127. ISBN: 978-3-642-19829-8. DOI: 10.1007/978-3-642-19829-8_8.

[8] Adam Chlipala. "A Verified Compiler for an Impure Functional Language". In: *ACM SIGPLAN Notices* 45.1 (Jan. 17, 2010), pp. 93–106. ISSN: 0362-1340. DOI: 10.1145/1707801.1706312. URL: https://doi.org/10.1145/1707801.1706312 (visited on 05/22/2020).

[9] Adam Chlipala. *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant*. MIT Press, 2013. ISBN: 978-0-262-02665-9.

[10] Benoît Combemale et al. "Essay on Semantics Definition in MDE. An Instrumented Approach for Model Verification". In: *Journal of Software* 4 (Nov. 1, 2009). DOI: 10.4304/jsw.4.9.943-958.

[11] Thierry Coquand and Christine Paulin. "Inductively defined types". In: *COLOG-88*. Ed. by Per Martin-Löf and Grigori Mints. Lecture Notes in Computer Science. Springer, 1990, 50–66. ISBN: 978-3-540-46963-6. DOI: 10.1007/3-540-52335-9_47.

[12] Lukasz Fronc and Franck Pommereau. "Towards a Certified Petri Net Model-Checker". In: *Programming Languages and Systems*. Ed. by Hongseok Yang. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 322–336. ISBN: 978-3-642-25318-8. DOI: 10.1007/978-3-642-25318-8_24.

[13] A. Habibi and S. Tahar. "Design and Verification of SystemC Transaction-Level Models". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.1 (Jan. 2006), pp. 57–68. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2005.863187.

[14] William A Howard. "The formulae-as-types notion of construction". In: *To HB Curry: essays on combinatory logic, lambda calculus and formalism* 44 (1980), pp. 479–490.

[15] Xavier Leroy. "A Formally Verified Compiler Back-End". In: *Journal of Automated Reasoning* 43.4 (Nov. 4, 2009), p. 363. ISSN: 1573-0670. DOI: 10.1007/s10817-009-9155-4. URL: https://doi.org/10.1007/s10817-009-9155-4 (visited on 01/21/2020).

[16] Andreas Lööw. "Lutsig: A Verified Verilog Compiler for Verified Circuit Development". In: *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2021. New York, NY, USA: Association for Computing Machinery, Jan. 17, 2021, pp. 46–60. ISBN: 978-1-4503-8299-1. DOI: 10.1145/3437992.3439916. URL: https://doi.org/10.1145/3437992.3439916 (visited on 05/04/2021).

[17] Said Meghzili et al. "On the Verification of UML State Machine Diagrams to Colored Petri Nets Transformation Using Isabelle/HOL". In: *2017 IEEE International Conference on Information Reuse and Integration (IRI)*. 2017 IEEE International Conference on Information Reuse and Integration (IRI). Aug. 2017, pp. 419–426. DOI: 10.1109/IRI.2017.63.

[18] Joan Moschovakis. "Intuitionistic Logic". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2018. Metaphysics Research Lab, Stanford University, 2018. URL: https://plato.stanford.edu/archives/win2018/entries/logic-intuitionistic/.

[19] Christine Paulin-Mohring. "Introduction to the Coq Proof-Assistant for Practical Software Verification". In: *Tools for Practical Software Verification*. Ed. by Bertrand Meyer and Martin Nordio. Vol. 7682. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 45–95. ISBN: 978-3-642-35745-9 978-3-642-35746-6. DOI: 10.1007/978-3-642-35746-6_3. URL: http://link.springer.com/10.1007/978-3-642-35746-6_3 (visited on 10/14/2019).

[20] Martin Strecker. "Formal Verification of a Java Compiler in Isabelle". In: *Automated Deduction—CADE-18*. International Conference on Automated Deduction. Springer, Berlin, Heidelberg, July 27, 2002, pp. 63–77. DOI: 10.1007/3-540-45620-1_5. URL: https://link.springer.com/chapter/10.1007/3-540-45620-1_5 (visited on 06/08/2020).

[21] Yong Kiam Tan et al. "A New Verified Compiler Backend for CakeML". In: (), p. 14.

[22] The Coq Development Team. *Coq, version 8.13.2*. Citation Key: Coq. 2021. URL: https://coq.inria.fr/.

[23] Freek Wiedijk. "The De Bruijn Factor". In: (Aug. 12, 2000).

[24] Glynn Winskel. *The formal semantics of programming languages: an introduction*. Citation Key: Winskel1993. MIT press, 1993.

[25] Zhibin Yang et al. "From AADL to Timed Abstract State Machines: A Verified Model Transformation". In: *Journal of Systems and Software* 93 (July 1, 2014), pp. 42–68. ISSN: 0164-1212. DOI: 10.1016/j.jss.2014.02.058. URL: http://www.sciencedirect.com/science/article/pii/S0164121214000727 (visited on 01/16/2020).

[26] Zhibin Yang et al. "Towards a Verified Compiler Prototype for the Synchronous Language SIGNAL". In: *Frontiers of Computer Science* 10.1 (Feb. 1, 2016), pp. 37–53. ISSN: 2095-2236. DOI: 10.1007/s11704-015-4364-y. URL: https://doi.org/10.1007/s11704-015-4364-y (visited on 01/21/2020).