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

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM

À la recherche de l'échafaudage parfait : efficace, de qualité et garanti

Présenté par Vincent IAMPIETRO

Le Date de la soutenance

Sous la direction de Annie Chateau
et Rodolphe Giroudeau

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

For/Dedicated to/To my…

Chapter 1

Proving semantic preservation in HILECOP

In this chapter, I want to talk about/draw the attention to:

- The differentiation of boolean operators and intuitionistic logic operators
- The correspondence between combinational signal value and there assignment expression deduced from the code. Explain that this is where the $\mathcal{H}$-VHDL semantics plays its part in the proof; although we are not detailling how assignment expressions are deduced from running the semantics of the $\mathcal{H}$-VHDL code. Give some examples of correspondence between combinational signal value and assignment expressions (in part "a detailled proof")
- the properties of comp. instances itfaces deduced from the transformation (in part "a detailled proof")
- The particularity of the similarity relation for time counters.
- In the part about the implementation, explain that the lemmas about the transformation function are expressed with universal binders rather than with existential ones to avoid the multiple instantiation of components in the proceeding of a proof.

In this chapter, we present our semantic preservation theorem along with its 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 used in the proof, and some hints regarding the proof strategy. The full proof is available to the reader in Appendix **??**. The theorems and lemmas presented in this chapter will be refering to the lemmas of Appendix **??**. The structure of this chapter is the following one: in Section 1.1, we present our review of the literature pertaining to the proof of semantic preservation theorems for transformation functions; in Section , we detail our state similarity relation, i.e, the semantic bound between an SITPN and its $\mathcal{H}$-VHDL translation; in Section, we draw out our semantic preservation theorem; in Section, we detail a particularly tricky point of the proof related to the computation of fired transitions, and we show how it has led to a bug detection in HILECOP's code; in Section, we present some points of the mechanization of the proof verification with the Coq proof assistant.

1.1 Semantic preserving transformations in the literature

In this section, we present the review of the literature pertaining to 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. 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 we used for the literature review is divided in 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

1.1.1 Transformations and proofs of semantic preservation

In the introduction of his article about CompCert [11], 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 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, or bisimulation, diagram. Figure shows the different kind of simulation diagrams possibly relating two programs. 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.

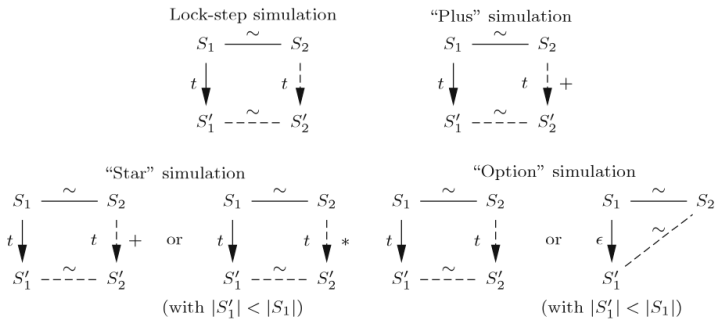

FIGURE 1.1: Simulation diagrams between source and target programs

Anyway, in the case where the source program execution succeeds, the theorem of semantic preservation takes this general form:

Consider a source program P_1 compiled into a target program P_2, a starting state S_1 for program P_1 and a starting state S_2 for program P_2 such that S_1 and S_2 are similar states w.r.t. the

exhibited state comparison relation. If the execution of P_1 leads from state S_1 to state S'_1, then there exists a 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 exhibited state comparison relation.

Compiler verification tasks 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 aim of the thesis is to perform compiler *verification* over the HILECOP methodology. Some of the works, cited afterwards, are more interested in compiler or transformation validation techniques than in verification. They are presented here for the sake of coverage.

Now that we have clarified the meaning of semantic preservation for GPL compilers, we state that this definition of semantic preservation holds also for 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?
- How are described the runtime state for the source and target representations?
- How is expressed the state comparison relation?
- How is stated the semantic preservation theorem?
- What is the employed proof strategy?

Compilers for generic programming languages

Taking the CompCert compiler as an example, the compilation pass from Clight programs to Cminor programs is described in [2, 11]. 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 is the same for all intermediate language involved in the CompCert compiler. The memory model consists in block references; each block has a lower and an upper bound. To access a 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 sof the source program to the memory states 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: the authors prove a simulation lemma for the

Clight expressions, then for the 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 [11, 14], or compilers for functional languages [7, 15], compiler verfication proceeds as follows:

1. establish a relation between the memory models of the source and target languages, and between the global execution environments
2. prove simulation 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 [11]). As a consequence, the complexity of the relation for memory model comparison increases.

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 in 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 time-related semantics of the considered languages.

In [3, 5], the source languages are a subset of the BlueSpec specification language for hardware synthesis , and the target is an RTL representation of the circuit. The execution states of the source and target are based on registers. In [3], the execution state also hold 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 states that the registers hold the same values after the execution of source program and the resulting RTL circuit after one clock cycle.

In [4], the source language is a subset of Lustre and the target language is 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 refer to instances of other nodes. In the compilation process, each Lustre node is translated into an Obc class. An Obc class hold 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. Also, as we are dealing with hardwares, the judgments treat synchronous statements and combinational ones. On the side of the Obc language, the semantics define a function *step* that computes the execution 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 output streams o from input streams i, then the iterative execution of the *step* function for the corresponding Obc class builds every step of output streams o given the values of input streams i. The proof is done by induction

over the clock step count, and by induction over the evaluation derivation of the nody instruction body.

In [12], 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 resembles the one of VHDL; the 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 by means of 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 [17], 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 (refering 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 [10], the authors verify a methodology to design hardware models with SystemC models. SystemC models describe hardware with modules; a module is a C++ class with ports, data members and methods. The methodology describes a transformation from SystemC into 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 modules 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 module 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.

Model transformations

Regarding model transformations, a lot of works consider semantic preservation as the preservation of structural properties in the transformed model [1, 6, 13].

Still, there are many cases where the source model and th 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 simulation relation (see Figure 1.1).

In [8] and [16], the authors are interested in giving a translational semantics to a given model having itself a reference execution semantics. In [8], the source models are called xSpem models; they describe a set of activities exchanging resources and an holding 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 1.2.

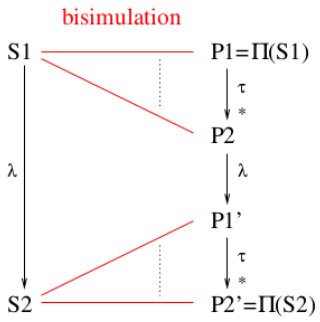

FIGURE 1.2: Bisimulation diagram relating an xSpem model execution and a Petri net execution

In Figure 1.2, 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 interest for the proof (i.e, action λ). Referring to the diagrams of Figure 1.1, this is a case of "star" simulation. The proof is performed by reasoning by induction on the structure of the xSpem model, and then by reasoning of the state transition semantics of xSpem models and PNs.

In [16], 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. 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 [9], 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 trully 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 straight-forward. The PN model has a 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 Petri net.

Discussions on transformations and proof strategies

In this thesis, we are interested in the verification of a semantic preservation property for a given transformation by proving a bisimulation theorem. To achieve this kind of proof task, the proceedings are quite similar, at least in the three cases of transformation presented above (i.e, GPLs compilation, HDLs compilation and model transformations). Even though the source and target languages or models are different from one case of transformation to the other, however, bisimulation theorems carry the same structure. The state comparison relation and the choice of the bisimulation diagram are the two angular stones of the process.

One can notice that when verifying the transformation of HDL programs, the bisimulation 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 this time-related computational step. This differs from the expression of bisimulation 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 bisimulation theorems are systematically done by induction over the semantics relation of the source languages. The semantics relation 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 enable compositional reasoning, i.e: to split the difficulty of proving the bisimulation theorem into simpler lemmas about the execution of simpler programs or simple model structures.

1.2 The state similarity relation

Before stating the 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 semantical matching means between an SITPN state and an $\mathcal{H}$-VHDL state?
2. when, in the course of the execution of an SITPN and an $\mathcal{H}$-VHDL design, does this semantical matching must hold?

We must relate the elements that constitute the execution state of an SITPN to the elements that constitute the execution state of an $\mathcal{H}$-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 Add ref.

to values, and of a component store mapping component instances to their own internal 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 structure to the component instances and signals on the $\mathcal{H}$-VHDL side. Thus, the state similarity relation expressing a semantical match between an SITPN state and an $\mathcal{H}$-VHDL design is defined as follows:

Definition 1 (General state 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 $\gamma \in WM(sitpn, d)$, an SITPN state $s \in S(sitpn)$ and a design state $\sigma \in \Sigma(\Delta)$ are similar, written $\gamma \vdash s \sim \sigma$ iff*

1. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s.M(p) = \sigma(id_p)(\text{"}s\_marking\text{"})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\big(upper(I_s(t)) = \infty \wedge s.I(t) \leq lower(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\text{"}s\_time\_counter\text{"})\big)$
   $\wedge\big(upper(I_s(t)) = \infty \wedge s.I(t) > lower(I_s(t)) \Rightarrow \sigma(id_t)(\text{"}s\_time\_counter\text{"}) = lower(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) > upper(I_s(t)) \Rightarrow \sigma(id_t)(\text{"}s\_time\_counter\text{"}) = upper(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) \leq upper(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\text{"}s\_time\_counter\text{"})\big)$.
3. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, s.reset_t(t) = \sigma(id_t)(\text{"}s\_reinit\_time\_counter\text{"})$.
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 to the value of the *s_marking* signal inside the internal state of the place component instance id_p. 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. 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.

Explain the time counter particular relation.

The second question that we asked above was: when does this 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 and correction (see Section 1.4), the state similarity relation of Definition 1.2 does not hold at the end of a clock cycle. The equality between reset orders (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 defined two state similarity relation; one that checks the semantic matching after the rising edge of the clock signal (i.e, in the middle of the clock cycle), and one that checks the semantic matching 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 2 (Post rising edge state similarity). *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, an elaborated design $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$, and a binder $\gamma \in WM(sitpn, d)$, an SITPN state $s \in S(sitpn)$ and a design state $\sigma \in \Sigma(\Delta)$ are similar after a rising edge happening, written $\gamma \vdash s \overset{\uparrow}{\sim} \sigma$ iff*

1. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s.M(p) = \sigma(id_p)(\text{"s\_marking"})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\big(upper(I_s(t)) = \infty \wedge s.I(t) \leq lower(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\text{"s\_time\_counter"})\big)$
   $\wedge\big(upper(I_s(t)) = \infty \wedge s.I(t) > lower(I_s(t)) \Rightarrow \sigma(id_t)(\text{"s\_time\_counter"}) = lower(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) > upper(I_s(t)) \Rightarrow \sigma(id_t)(\text{"s\_time\_counter"}) = upper(I_s(t))\big)$
   $\wedge\big(upper(I_s(t)) \neq \infty \wedge s.I(t) \leq upper(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\text{"s\_time\_counter"})\big)$.
3. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, s.reset_t(t) = \sigma(id_t)(\text{"s\_reinit\_time\_counter"})$.
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 2 is similar to Definition 1 in all points, except for the value of conditions. A condition of an SITPN is implemented by an primary input port in the resulting $\mathcal{H}$-VHDL design. In $\mathcal{H}$-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.

The state similarity relation draws out a correspondence between the values hold by an SITPN state and the values of the signals declared in an $\mathcal{H}$-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 transition component instance 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 similiraty 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 3 (Full post rising edge state similarity). *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, an elaborated design $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$, 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 $\sigma \in \Sigma(\Delta)$ are fully similar after a rising edge happening at clock cycle count τ, written $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$ iff $\gamma \vdash s \overset{\uparrow}{\sim} \sigma$ (Definition 2) and*

1. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Sens(s.M) \Leftrightarrow \sigma(id_t)(\text{"s\_enabled"}) = \texttt{true}$.
2. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Sens(s.M) \Leftrightarrow \sigma(id_t)(\text{"s\_enabled"}) = \texttt{false}$.

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

Definition 3 extends Definition 2 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*.

The state similarity relation after a falling edge is defined as follows:

Definition 4 (Post falling edge state similarity)**.** *For a given* $sitpn \in SITPN$*, a* $\mathcal{H}$*-VHDL design* $d \in design$*, an elaborated design* $\Delta \in ElDesign(d, \mathcal{D}_\mathcal{H})$*, and a binder* $\gamma \in WM(sitpn, d)$*, an SITPN state* $s \in S(sitpn)$ *and a design state* $\sigma \in \Sigma(\Delta)$ *are similar after a falling edge, written* $\gamma \vdash s \overset{\downarrow}{\sim} \sigma$ *iff*

1. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s.M(p) = \sigma(id_p)(\text{"s\_marking"}).$
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
$\big(upper(I_s(t)) = \infty \wedge s.I(t) \leq lower(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\text{"s\_time\_counter"})\big)$
$\wedge \big(upper(I_s(t)) = \infty \wedge s.I(t) > lower(I_s(t)) \Rightarrow \sigma(id_t)(\text{"s\_time\_counter"}) = lower(I_s(t))\big)$
$\wedge \big(upper(I_s(t)) \neq \infty \wedge s.I(t) > upper(I_s(t)) \Rightarrow \sigma(id_t)(\text{"s\_time\_counter"}) = upper(I_s(t))\big)$
$\wedge \big(upper(I_s(t)) \neq \infty \wedge s.I(t) \leq upper(I_s(t)) \Rightarrow s.I(t) = \sigma(id_t)(\text{"s\_time\_counter"})\big).$
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 4 is similar to Definition 1 except for the equality between reset orders and the value of signal *s_reinit_time_counter*.

The extended version of the post falling edge state similarity relation is as follows:

Definition 5 (Full post falling edge state similarity)**.** *For a given* $sitpn \in SITPN$*, a* $\mathcal{H}$*-VHDL design* $d \in design$*, an elaborated design* $\Delta \in ElDesign(d, \mathcal{D}_\mathcal{H})$*, and a binder* $\gamma \in WM(sitpn, d)$*, an SITPN state* $s \in S(sitpn)$ *and a design state* $\sigma \in \Sigma(\Delta)$ *are fully similar after a falling edge, written* $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$ *iff* $\gamma \vdash s \overset{\downarrow}{\sim} \sigma$ *(Definition 4) and*

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

Definition 5 extends Definition 4 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 $s\_input\_token\_sum$

1.3 Behavior Preservation Theorem

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

1.3.1 Proof notations

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

- At the point of reading, the most recent framed box 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): $\boxed{\texttt{true} = \texttt{true}}$
- 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 **??** gives the full correspondence between constants and signals, and their aliases.

1.3.2 Preliminary definitions

We define here some relations that are necessary to formalize the 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 mimick 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 6 states that the execution environment for a given SITPN and the simulation environment for a given $\mathcal{H}$-VHDL design are similar.

Definition 6 (Similar environments). *For a given $sitpn \in SITPN$, a $\mathcal{H}$-VHDL design $d \in design$, a design store $\mathcal{D} \in entity\text{-}id \nrightarrow design$, an elaborated version $\Delta \in ElDesign(d, \mathcal{D})$ 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, noted $\gamma \vdash E_p \overset{env}{=} E_c$, iff 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 6 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 6, 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. The relation presented in Definition 7 permits to compare such traces.

Definition 7 (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 $\theta_\sigma \in \texttt{list}(\Sigma(\Delta))$ 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 7, 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 and then by a falling edge. 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 a initial state, that is, a state neither reached after a rising nor after falling edge, we give a slightly different definition of the execution trace similarity relation in Definition 8.

Definition 8 (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 $\theta_\sigma \in \texttt{list}(\Sigma(\Delta))$ are fully similar, written $\gamma \vdash \theta_s \sim \theta_\sigma$, according to the following rules:*

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

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.

1.3.3 The behavior preservation theorem

Theorem 1 states that the HILECOP transformation is semantic preserving when the input model is a well-defined SITPN. As a complementary task, we could show that if the transformation function returns a couple design and binder, and not an error, then the input SITPN is well-defined.

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 $\tau \in \mathbb{N}$, a 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 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 exists an elaborated design $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$ s.t. for all simulation environment $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value$, verifying

- *Simulation/Execution environments are similar: $\gamma \vdash E_p \overset{env}{=} E_c$*

then there exists a simulation trace $\theta_\sigma \in \texttt{list}(\Sigma(\Delta))$ s.t.

- *Under the HILECOP design store $\mathcal{D}_{\mathcal{H}}$ and with an empty generic constant dimensioning function ($\emptyset$), design d 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 $\theta_s \in \texttt{list}(S(sitpn))$, let us show that

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

Appealing to Theorems Elaboration, Initialization and Simulation, let us take an elaborated design Δ, two design states $\sigma_e, \sigma_0 \in \Sigma(\Delta)$, and a simulation trace $\theta_\sigma \in$ such that:

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

By definition of the $\mathcal{H}$-VHDL full simulation relation, we have:

$$\begin{aligned} \mathcal{D}_{\mathcal{H}}, \Delta, \emptyset, E_p, \tau \vdash \mathrm{d} \xrightarrow{full} \theta_\sigma \equiv \exists \sigma_e, \sigma_0 \in \Sigma(\Delta),\ & \mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e) \\ & \wedge \mathcal{D}_{\mathcal{H}}, \Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0 \\ & \wedge \mathcal{D}_{\mathcal{H}}, E_p, \Delta, \tau, \sigma_0 \vdash \mathrm{d.cs} \to \theta_\sigma \end{aligned} \tag{1.1}$$

Rewriting the goal with (1.1):

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

Let us use Δ, σ_e, $\sigma_0 \in \Sigma(\Delta)$ and θ_σ to prove the goal:

$$\mathcal{D}_{\mathcal{H}}, \emptyset \vdash d \xrightarrow{elab} (\Delta, \sigma_e) \wedge \mathcal{D}_{\mathcal{H}}, \Delta, \sigma_e \vdash d.cs \xrightarrow{init} \sigma_0 \wedge \mathcal{D}_{\mathcal{H}}, E_p, \Delta, \tau, \sigma_0 \vdash \text{d.cs} \rightarrow \theta_\sigma \wedge \theta_s \sim \theta_\sigma$$

We assumed the three first points of the goal, and the last point, i.e $\theta_s \sim \theta_\sigma$, is proved by appealing to Theorem Full bisimulation.

□

To prove Theorem 1, we must first prove that for all $\mathcal{H}$-VHDL design returned by the transformation function, there exists an elaborated version of it (Theorem Elaboration); then, we must prove that we can always build a simulation trace respecting the $\mathcal{H}$-VHDL simulation relation over τ simulation cycles (Theorem Initialization and Simulation). Finally, we can establish that the behaviors are similar by comparing the respective SITPN execution and $\mathcal{H}$-VHDL simulation traces. In this thesis, we are focusing on the proof that the execution/simulation traces are similar. For now, we choose to disregard the proof of theorems Elaboration, Initialization and Simulation stating the existence of an elaborated design and of a simulation trace for all $\mathcal{H}$-VHDL design returned by the HILECOP transformation function.

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

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

then there exists an elaborated design $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$ and a design state $\sigma_e \in \Sigma(\Delta)$ 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 (Initialization). *For all $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$, $\sigma_e \in \Sigma(\Delta)$ 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 (Simulation). *For all $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $\Delta \in ElDesign(d, \mathcal{D}_{\mathcal{H}})$, $\sigma_e, \sigma_0 \in \Sigma(\Delta)$ 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 for all simulation environment $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \rightarrow Ins(\Delta) \rightarrow value$, and simulation cycle count $\tau \in \mathbb{N}$, there exists s simulation trace $\theta_\sigma \in \texttt{list}(\Sigma(\Delta))$ s.t.

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

1.3.4 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 8. In this thesis, we proved this particular theorem, and as said before, we left the proofs of Theorems Elaboration, Initialization and Simulation) to later. We chose to focus our work on the bisimulation theorem, because it directly adresses the semantic preservation property of HILECOP's transformation function.

Theorem 5 (Full bisimulation). *For all* $sitpn \in SITPN, d \in design, \gamma \in WM(sitpn, d), \tau \in \mathbb{N}, E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}, \theta_s \in \texttt{list}(S(sitpn)), \Delta \in ElDesign(d, \mathcal{D}_\mathcal{H}), E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value, \theta_\sigma \in \texttt{list}(\Sigma(\Delta))$ *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. Given all the above variables and 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 **??**, 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 $E_p, \Delta, \tau, \sigma_0 \vdash \mathrm{d.cs} \to \theta'$ 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, $E_p, \Delta, \tau, \sigma_0 \vdash \mathrm{d.cs} \to \theta'$), we have:

$E_p, \Delta, \tau, \sigma_0 \vdash d.cs \xrightarrow{\uparrow\downarrow} \sigma, \sigma'$ and $E_p, \Delta, \tau - 1, \sigma' \vdash \text{d.cs} \rightarrow \theta''$ 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 **??**, 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 **??**, 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 **??** and Lemma Falling edge, 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 **??** and Lemma Falling edge, we have $\gamma \vdash s \overset{\downarrow}{\approx} \sigma'$.

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

□

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 1); this is done by appealing to Lemma **??**. 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 **??**. The detailled proofs for Lemmas **??** and **??** are given in Sections **??** and **??**.

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:* $\gamma \vdash s \overset{\downarrow}{\approx} \sigma$
- $E_c, \tau \vdash sitpn, s \to \theta_s$
- $E_p, \Delta, \tau, \sigma \vdash d.cs \to \theta_\sigma$

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

Proof. Given all the above variables and assuming the above hypotheses, let us show $\boxed{\gamma \vdash \theta_s \overset{\uparrow}{\sim} \theta_\sigma}$. 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**: $\tau > 0$.

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

  By definition of the SITPN execution and the $\mathcal{H}$-VHDL simulation relations for $\tau > 0$, we have:

  – $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$ and $E_c, \tau \vdash s' \xrightarrow{\downarrow} s''$ and $E_c, \tau - 1 \vdash sitpn, s'' \to \theta$.
  – $\texttt{Inject}_\uparrow(\sigma, E_p, \tau, \sigma_i)$ and $\Delta, \sigma_i \vdash$ d.cs $\xrightarrow{\uparrow} \sigma_\uparrow$ and $\Delta, \sigma_\uparrow \vdash$ d.cs $\xrightarrow{\rightsquigarrow} \sigma'$
  – $\texttt{Inject}_\downarrow(\sigma', E_p, \tau, \sigma'_i)$ and $\Delta, \sigma'_i \vdash$ d.cs $\xrightarrow{\downarrow} \sigma_\downarrow$ and $\Delta, \sigma_\downarrow \vdash$ d.cs $\xrightarrow{\rightsquigarrow} \sigma''$
  – $E_p, \Delta, \tau - 1, \sigma'' \vdash d.cs \to \theta'$.

  and $\theta_s = s' :: s'' :: \theta$ 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 Falling edge, 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 Falling edge and Rising edge, 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. $\boxed{\gamma \vdash \theta \overset{\updownarrow}{\sim} \theta'}$

   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 Falling edge and Rising edge, 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 1.3. 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 Rising edge. 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 1.3). Then, Lemma Rising edge 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 Falling edge. 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 1.3). Then, Lemma Rising edge 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.

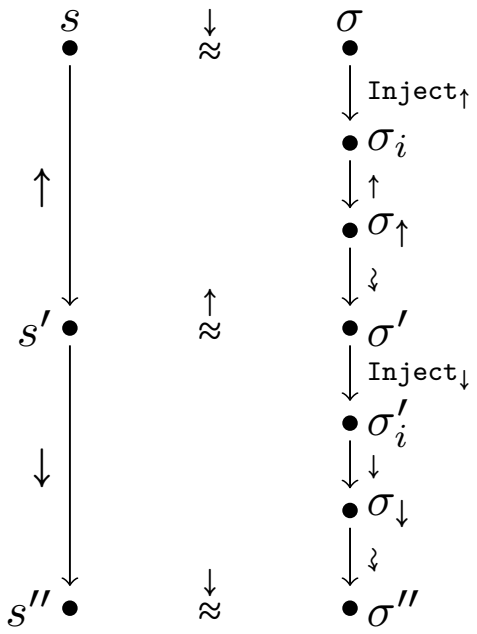

FIGURE 1.3: Bisimulation diagram over one clock cycle for a source SITPN and a target $\mathcal{H}$-VHDL design.

Here, we present Lemma Rising edge and Lemma Falling edge, along with their proofs. In the two lemmas, we added an extra hypothesis about the starting state of the $\mathcal{H}$-VHDL design: $\mathcal{D}_{\mathcal{H}}, \Delta, \sigma \vdash \text{d.cs} \xrightarrow{comb} \sigma$. The 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 1.4 for more details about combinational equations).

To prove Lemmas Rising edge and Falling edge, 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 **??**. 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 detailled in Section 1.4.

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

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

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. $\forall p \in P, id_p \in Comps(\Delta)\ s'.t.\ \gamma(p) = id_p,\ s'.M(p) = \sigma'(id_p)(\text{"s\_marking"})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $\big(upper(I_s(t)) = \infty \wedge s'.I(t) \leq lower(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"})\big)$
   $\wedge \big(upper(I_s(t)) = \infty \wedge s'.I(t) > lower(I_s(t)) \Rightarrow \sigma'(id_t)(\text{"s\_time\_counter"}) = lower(I_s(t))\big)$
   $\wedge \big(upper(I_s(t)) \neq \infty \wedge s'.I(t) > upper(I_s(t)) \Rightarrow \sigma'(id_t)(\text{"s\_time\_counter"}) = upper(I_s(t))\big)$
   $\wedge \big(upper(I_s(t)) \neq \infty \wedge s'.I(t) \leq upper(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"})\big)$.

3. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, s'.reset_t(t) = \sigma'(id_t)(\text{"s\_reinit\_time\_counter"})$.
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. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Sens(s'.M) \Leftrightarrow \sigma'(id_t)(\text{"s\_enabled"}) = \texttt{true}$.
7. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Sens(s'.M) \Leftrightarrow \sigma'(id_t)(\text{"s\_enabled"}) = \texttt{false}$.
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 $conds(t) = \{c \in \mathcal{C} \mid \mathbb{C}(t,c) = 1 \vee \mathbb{C}(t,c) = -1\}$.

Each point is proved by a separate lemma:

– Apply Lemma **??** to solve 1.

– Apply Lemma **??** lemma to solve 2.

– Apply Lemma **??** to solve 3.

– Apply Lemma **??** to solve 4.

– Apply Lemma **??** to solve 5.

– Apply Lemma **??** to solve 6.

– Apply Lemma **??** to solve 7.

– Apply Lemma **??** to solve 8.

□

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

- $\lfloor sitpn \rfloor_\mathcal{H} = (d, \gamma)$ *and* $\gamma \vdash E_p \overset{env}{=} E_c$ *and* $\mathcal{D}_\mathcal{H}, \emptyset \vdash \mathrm{d} \xrightarrow{elab} \Delta, \sigma_e$
- $\gamma, E_c, \tau \vdash s \overset{\uparrow}{\approx} \sigma$
- $E_c, \tau \vdash s \overset{\downarrow}{\to} s'$
- $\texttt{Inject}_\downarrow(\sigma, E_p, \tau, \sigma_i)$ *and* $\Delta, \sigma_i \vdash \mathrm{d.cs} \overset{\downarrow}{\to} \sigma_\downarrow$ *and* $\Delta, \sigma_\downarrow \vdash \mathrm{d.cs} \overset{\rightsquigarrow}{\to} \sigma'$
- *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. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p,\ s'.M(p) = \sigma'(id_p)(\text{"s\_marking"})$.
2. $\forall t \in T_i, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t,$
   $(upper(I_s(t)) = \infty \wedge s'.I(t) \leq lower(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"}))$
   $\wedge(upper(I_s(t)) = \infty \wedge s'.I(t) > lower(I_s(t)) \Rightarrow \sigma'(id_t)(\text{"s\_time\_counter"}) = lower(I_s(t)))$
   $\wedge(upper(I_s(t)) \neq \infty \wedge s'.I(t) > upper(I_s(t)) \Rightarrow \sigma'(id_t)(\text{"s\_time\_counter"}) = upper(I_s(t)))$
   $\wedge(upper(I_s(t)) \neq \infty \wedge s'.I(t) \leq upper(I_s(t)) \Rightarrow s'.I(t) = \sigma'(id_t)(\text{"s\_time\_counter"}))$.
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. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Firable(s') \Leftrightarrow \sigma'(id_t)(\text{"s\_firable"}) = \texttt{true}$.
7. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Firable(s') \Leftrightarrow \sigma'(id_t)(\text{"s\_firable"}) = \texttt{false}$.
8. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \in Fired(s') \Leftrightarrow \sigma'(id_t)(\text{"fired"}) = \texttt{true}$.
9. $\forall t \in T, id_t \in Comps(\Delta)\ s.t.\ \gamma(t) = id_t, t \notin Fired(s') \Leftrightarrow \sigma'(id_t)(\text{"fired"}) = \texttt{false}$.
10. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p, \sum_{t \in Fired(s')} pre(p,t) = \sigma'(id_p)(\text{"s\_output\_token\_sum"})$.
11. $\forall p \in P, id_p \in Comps(\Delta)\ s.t.\ \gamma(p) = id_p, \sum_{t \in Fired(s')} post(t,p) = \sigma'(id_p)(\text{"s\_input\_token\_sum"})$.

Each point is proved by a separate lemma:

– Apply Lemma **??** to solve 1.

– Apply Lemma **??** to solve 2.

– Apply Lemma **??** to solve 3.

– Apply Lemma **??** to solve 4.

– Apply Lemma **??** to solve 5.

– Apply Lemma **??** to solve 6.

– Apply Lemma **??** to solve 7.

– Apply Lemma Falling edge equal fired to solve 8.

– Apply Lemma Falling Edge Equal Not Fired to solve 9.

– Apply Lemma **??** to solve 10.

– Apply Lemma **??** to solve 11.

□

1.4 A detailled proof: equivalence of fired transitions

The goal of this section is to present the overall proof strategy to establish the semantic preservation property. We use the proof of Lemma Falling edge equal fired, involved in the proof of Lemma Falling edge, to illustrate our demonstration technics. The proof of Lemma Falling edge equal fired has been one tricky part of the proof, and therefore, 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.

1.4.1 Informal presentation of the proof

The proof we will detail here pertains to the set of fired transitions. In an SITPN, the firing process is involved in the computation of the new marking, the reset orders, and the execution of functions during the rising edge phase. Therefore, 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. To prove the equivalence, we must first look at the definition of the set of fired transitions on the SITPN and 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 (which will also be the state at the beginning of the next rising edge phase). A transition belongs to the set of fired transitions if it is *firable* (see Definition **??**) and sensitized by the *residual* marking at the considered SITPN state. Figure 1.4 gives the set of fired transitions, i.e $Fired(s)$, for 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.

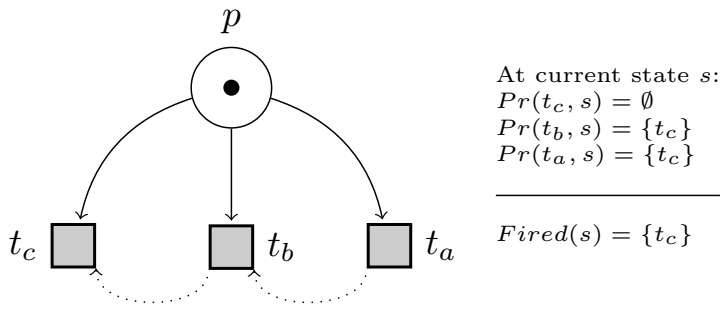

FIGURE 1.4: The set of fired transitions for an example SITPN at a given SITPN state s; on the right side, the dotted arrows indicates the priority relation between the three transitions (t_c is the top-priority transition); on the left side, each transition is associated to its Pr set which are necessary 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 1.4). 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 1.4, it is impossible to know the content of $Pr(t_a)$ 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 $Pr(t_b)$. 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 transition component instances. The transition design declares an output port of 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 transition component instance 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 on the described hardware circuit. Figure 1.5 shows a part of the transition design architecture describing how the `fired` port is connected with to the other internal signals.

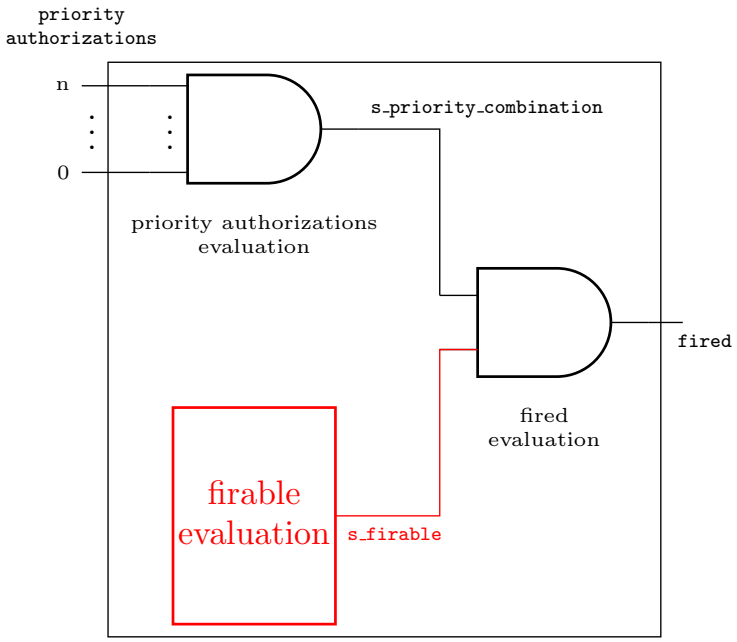

FIGURE 1.5: Wiring of the `fired` output port in the transition design architecture; on the left side is the input interface of the transition design; on the right side is the output interface of the transition design, with the `fired` port; in red are the parts of the architecture that depend on synchronous logic and in black are the parts that are purely combinational.

In Figure 1.5, the labels underneath the and 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 1.5 is a 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. Considering a transition component instance id_t at a given stable state σ, the `fired` port equation is:

$$\sigma(id_t)(''fired'') = \sigma(id_t)(''s\_firable'') \,.\, \sigma(id_t)(''s\_priority\_combination'') \tag{1.2}$$

In Equation (1.2), $\sigma(id_t)$ denotes the internal state of the transition component instance id_t. From $\sigma(id_t)$, we have access to the values of the input ports, the outport ports and the internal signals of the component. Equation (1.2) states that the value of the `fired` port is a simple and expression between the value of the internal signal `s_firable` and `s_priority_combination`. 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.

1.4.2 Formal presentation of the proof

Definition 9 (Falling edge hypotheses)**.** *Given an* $sitpn \in SITPN$, $d \in design$, $\gamma \in WM(sitpn, d)$, $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $\Delta \in ElDesign(d, \mathcal{D}_\mathcal{H})$, $E_p \in (\mathbb{N} \times \{\uparrow, \downarrow\}) \to Ins(\Delta) \to value$, $\tau \in \mathbb{N}$, $s, s' \in S(sitpn)$, $\sigma_e, \sigma, \sigma_i, \sigma_\downarrow, \sigma' \in \Sigma(\Delta)$, *assume that:*

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

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 Def. 9, then* $\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t$, $t \in Fired(s') \Leftrightarrow \sigma'(id_t)(''fired'') = \mathtt{true}$.

Proof. Given a $t \in T$ and an id_t s.t. $\gamma(t) = id_t$, let us show $\boxed{t \in Fired(s') \Leftrightarrow \sigma'(id_t)(''fired'') = \mathtt{true}.}$ The proof is in two parts:

1. Assuming that $t \in Fired(s')$, let us show $\boxed{\sigma'(id_t)(''fired'') = \mathtt{true}.}$

   By definition of $t \in Fired(s')$, there exists $fset \subseteq T$ s.t. $IsFiredSet(s', fset) \wedge t \in fset$.

   Let us take such an $fset$, and apply Lemma Falling Edge Equal Fired Set to solve the goal.

2. Assuming that $\sigma'(id_t)(''fired'') = \mathtt{true}$, let us show $\boxed{t \in Fired(s').}$

   By definition of $t \in Fired(s')$, let us show that $\boxed{\exists fset \subseteq T \text{ s.t. } IsFiredSet(s', fset) \wedge t \in fset}$

   Assuming that $sitpn$ is a well-defined $SITPN$ (see Section), we can always find an $fset \subseteq T$ such that $\forall s \in S(sitpn)$, $IsFiredSet(s, fset)$ is derivable. Let us take an $fset \subseteq T$ s.t. $IsFiredSet(s', fset)$, and use it to prove the goal by applying Lemma Falling Edge Equal Fired Set.

well-
PN

□

Lemma 5 (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 Def. 9, then $\forall t, id_t$ s.t. $\gamma(t) = id_t$, $t \notin Fired(s') \Leftrightarrow \sigma'_t("fired") = \texttt{false}$.*

Proof. Proving the above lemma is trivial by appealing to Lemma Falling edge equal fired and by reasoning on contrapositives. □

Lemma 6 (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 Def. 9, then $\forall t \in T$, $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $\forall fset \subseteq T$, s.t. $IsFiredSet(s', fset)$, $t \in fset \Leftrightarrow \sigma'(id_t)("fired") = true$.*

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 $\boxed{t \in fset \Leftrightarrow \sigma'(id_t)("fired") = true.}$

By definition of $IsFiredSet(s', fset)$, we have $IsFiredSetAux(s', \emptyset, T, fset)$.
Then, we can appeal to Lemma Falling Edge Equal Fired Set Aux to solve the goal, but first we must prove the following *extra hypothesis* (i.e, one of the premise of Lemma Falling Edge Equal Fired Set Aux):

$\boxed{\begin{array}{l}\forall t' \in T, id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in \emptyset \Rightarrow \sigma'(id_{t'})("fired") = \texttt{true}) \wedge (\sigma'(id_{t'})("fired") = \texttt{true} \Rightarrow t' \in \emptyset \vee t' \in T).\end{array}}$

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. $\boxed{t' \in \emptyset \Rightarrow \sigma'(id_{t'})("fired") = \texttt{true}}$
2. $\boxed{\sigma'(id_{t'})("fired") = \texttt{true} \Rightarrow t' \in \emptyset \vee t' \in T}$

Let us show these two points:

1. Assuming $t' \in \emptyset$, let us show $\boxed{\sigma'(id_{t'})("fired") = \texttt{true}.}$

   $t' \in \emptyset$ is a contradiction.

2. Assuming $\sigma'(id_{t'})("fired") = \texttt{true}$, let us show $\boxed{t' \in \emptyset \vee t' \in T.}$

   By definition, $t' \in T$.

□

Lemma 7 (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 Def. 9, then $\forall t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $\forall fired \subseteq T$, $T_s \subseteq T$, $fset \subseteq T$, assume that:*

- $IsFiredSetAux(s', fired, T_s, fset)$
- *EH (Extra. Hypothesis):*
  $\forall t' \in T$, $id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
  $(t' \in fired \Rightarrow \sigma'(id_{t'})("fired") = \texttt{true}) \wedge (\sigma'(id_{t'})("fired") = \texttt{true} \Rightarrow t' \in fired \vee t' \in T_s)$.

then $t \in fset \Leftrightarrow \sigma'(id_t)("fired") = \texttt{true}$.

Proof. Given a $t \in T$, an $id_t \in Comps(\Delta)$, a $fired, T_s, fset \subseteq T$, and assuming $IsFiredSetAux(s', fired, T_s, fset)$ and EH, let us show $\boxed{t \in fset \Leftrightarrow \sigma'(id_t)("fired") = \texttt{true}.}$
Let us reason by induction on $IsFiredSetAux(s', fired, T_s, fset)$.

- **BASE CASE**: $\boxed{t \in fired \Leftrightarrow \sigma'(id_t)("fired") = \texttt{true}.}$

  In that case, $fired = fset$ and $T_s = \emptyset$, EH looks like this:
  $\forall t' \in T,\ id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
  $(t' \in fired \Rightarrow \sigma'(id_{t'})("fired") = \texttt{true}) \wedge (\sigma'(id_{t'})("fired") = \texttt{true} \Rightarrow t' \in fired \ \vee \ t' \in \emptyset)$.

  From EH, we can deduce $t \in fired \Leftrightarrow \sigma'(id_t)("fired") = \texttt{true}.$

- **INDUCTION CASE**: $\boxed{t \in fset \Leftrightarrow \sigma'(id_t)("fired") = \texttt{true}.}$

  In that case, we have:

  - $IsTopPrioritySet(T_s, tp)$
  - $ElectFired(s', fired, tp, fired')$
  - $FiredAux(s', fired', T_s \setminus tp, fset)$

  $(\forall t' \in T, id_{t'} \in Comps(\Delta)\ s.t.\ \gamma(t') = id_{t'},$
  $(t' \in fired' \Rightarrow \sigma'(id_{t'})("fired") = \texttt{true}) \wedge (\sigma'(id_{t'})("fired") = \texttt{true} \Rightarrow t' \in fired' \ \vee \ t' \in T_s \setminus tp)) \Rightarrow$
  $t \in fset \Leftrightarrow \sigma'_t("fired") = true.$

  Applying the induction hypothesis, then, the new goal is:

  $\forall t' \in T, id_{t'} \in Comps(\Delta)\ s.t.\ \gamma(t') = id_{t'},$
  $(t' \in fired' \Rightarrow \sigma'(id_{t'})("fired") = \texttt{true})$
  $\wedge(\sigma'(id_{t'})("fired") = \texttt{true} \Rightarrow t' \in fired' \ \vee \ t' \in T_s \setminus tp)$

  Apply Lemma Elect Fired Equal Fired to solve the goal.

□

Lemma 8 (Elect Fired Equal Fired). *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 Def. 9, then $\forall fired, fired', T_s, tp,\ fset \subseteq T$, assume that:*

- $IsTopPrioritySet(T_s, tp)$
- $ElectFired(s', fired, tp, fired')$
- $FiredAux(s', fired', T_s \setminus tp, fset)$
- *EH (Extra. Hypothesis):*
  $\forall t' \in T, id_{t'} \in Comps(\Delta)\ s.t.\ \gamma(t') = id_{t'},$
  $(t' \in fired \Rightarrow \sigma'(id_{t'})("fired") = \texttt{true}) \wedge (\sigma'(id_{t'})("fired") = \texttt{true} \Rightarrow t' \in fired \ \vee \ t' \in T_s)$

then $\forall t \in T, id_t \in Comps(\Delta)$ *s.t.* $\gamma(t) = id_t$,
$\big(t \in fired' \Rightarrow \sigma'(id_t)("fired") = \texttt{true}\big) \wedge \big(\sigma'(id_t)("fired") = \texttt{true} \Rightarrow t \in fired' \vee t \in T_s \setminus tp\big)$.

Proof. Given a $t \in T$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, let us show
$\boxed{\big(t \in fired' \Rightarrow \sigma'(id_t)("fired") = \texttt{true}\big) \wedge \big(\sigma'(id_t)("fired") = \texttt{true} \Rightarrow t \in fired' \vee t \in T_s \setminus tp\big).}$

Let us reason by induction on $ElectFired(s', fired, tp, fired')$; there are three cases:

1. **BASE CASE**: $tp = \emptyset$ and $fired = fired'$.
2. **INDUCTIVE CASE**: $tp = \{t_0\} \cup tp_0$ and t_0 is elected to be fired.
3. **INDUCTIVE CASE**: $tp = \{t_0\} \cup tp_0$ and t_0 is not elected to be fired.

Let us prove the goal in these three contexts:

1. **BASE CASE**:

   $\boxed{\big(t \in fired \Rightarrow \sigma'(id_t)("fired") = \texttt{true}\big) \wedge \big(\sigma'(id_t)("fired") = \texttt{true} \Rightarrow t \in fired \vee t \in T_s\big).}$

   Apply EH to solve the goal.

2. **INDUCTIVE CASE**: $tp = \{t_0\} \cup tp_0$ and t_0 is elected to be fired.
   In that case, we have:

   - $IsTopPrioritySet(T_s, \{t_0\} \cup tp_0)$
   - $ElectFired(s', fired \cup \{t_0\}, tp_0, fired')$
   - $IsFiredSetAux(s', fired', T_s \setminus \{t_0\} \cup tp_0, fset)$
   - $t_0 \in Firable(s')$
   - $t_0 \in Sens(s'.M - \sum_{t_i \in Pr(t, fired)} pre(t_i))$ where $Pr(t, fired) = \{t' \mid t' \succ t \wedge t' \in fired\}$
   - EH: $\forall t' \in T,\ id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
     $(t' \in fired \Rightarrow \sigma'(id_{t'})("f") = \texttt{true}) \wedge (\sigma'(id_{t'})("f") = \texttt{true} \Rightarrow t' \in fired \ \vee \ t' \in T_s)$

   $\forall T'_s \subseteq T$,
   $IsTopPrioritySet(T'_s, tp_0) \Rightarrow$
   $IsFiredSetAux(s', fired', T'_s \setminus tp_0, fset) \Rightarrow$
   $\big(\forall t' \in T, id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
   $(t' \in fired \cup \{t_0\} \Rightarrow \sigma'_{t'}("f") = \texttt{true}) \wedge (\sigma'(id_{t'})("f") = \texttt{true} \Rightarrow t' \in fired \cup \{t_0\} \ \vee \ t' \in T'_s)\big) \Rightarrow$
   $\forall t \in T,\ id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$,
   $\big(t \in fired' \Rightarrow \sigma'(id_t)("f") = \texttt{true}\big) \wedge \big(\sigma'(id_t)("f") = \texttt{true} \Rightarrow t \in fired' \vee t \in T'_s \setminus tp_0\big)$

   $\boxed{\forall t \in T,\ id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$,
   $\big(t \in fired' \Rightarrow \sigma'_t("f") = \texttt{true}\big) \wedge \big(\sigma'_t("f") = \texttt{true} \Rightarrow t \in fired' \vee t \in T_s \setminus \{t_0\} \cup tp_0\big)}$

To solve the goal, we can apply the induction hypothesis with $T_s' = T_s \setminus \{t_0\}$; then, there are three points to prove:

(a) $\boxed{IsTopPrioritySet(T_s \setminus \{t_0\}, tp_0)}$

(b) $\boxed{IsFiredSetAux(s', fired', (T_s \setminus \{t_0\}) \setminus tp_0, fset)}$

(c) $\boxed{\forall t' \in T, id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\ (t' \in fired \cup \{t_0\} \Rightarrow \sigma'_{t'}(''f'') = \texttt{true}) \wedge (\sigma'(id_{t'})(''f'') = \texttt{true} \Rightarrow t' \in fired \cup \{t_0\} \vee t' \in T_s \setminus \{t_0\})}$

Let us prove these three points:

(a) $\boxed{IsTopPrioritySet(T_s \setminus \{t_0\}, tp_0)}$

Not provable yet.

(b) $\boxed{IsFiredSetAux(s', fired', (T_s \setminus \{t_0\}) \setminus tp_0, fset)}$.
We know that $(T_s \setminus \{t_0\}) \setminus tp_0 = T_s \setminus (\{t_0\} \cup tp_0)$, and thus
$IsFiredSetAux(s', fired', T_s \setminus (\{t_0\} \cup tp_0), fset)$ is an assumption.

(c) $\boxed{\forall t' \in T, id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\ (t' \in fired \cup \{t_0\} \Rightarrow \sigma'(id_{t'})(''f'') = \texttt{true}) \wedge (\sigma'(id_{t'})(''f'') = \texttt{true} \Rightarrow t' \in fired \cup \{t_0\} \vee t' \in T_s \setminus \{t_0\})}$

Given a $t' \in T$ and an $id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$, let us show

$\boxed{(t' \in fired \cup \{t_0\} \Rightarrow \sigma'(id_{t'})(''f'') = \texttt{true}) \wedge (\sigma'(id_{t'})(''f'') = \texttt{true} \Rightarrow t' \in fired \cup \{t_0\} \vee t' \in T_s \setminus \{t_0\}).}$

The proof is in two parts.

i. Assuming that $t' \in fired \cup \{t_0\}$, let us show $\boxed{\sigma'(id_{t'})(''f'') = \texttt{true}.}$
Case analysis on $t' \in fired \cup \{t_0\}$; there are two cases:
- $t' \in fired$
- $t' = t_0$

Let us prove the goal in these two contexts.
- **CASE** $t' \in fired$: Thanks to EH, we can deduce $\sigma'_{t'}(''f'') = \texttt{true}.$
- **CASE** $t' = t_0$:
  By definition of $id_{t'}$, there exist a $gm_{t'}$, $ipm_{t'}$, $opm_{t'}$ s.t. $\texttt{comp}(id_{t'}, ''transition'', gm_{t'}, ipm_{t'}, opm_{t'}) \in d.cs$.
  By property of the stabilize relation and $\texttt{comp}(id_{t'}, ''transition'', gm_{t'}, ipm_{t'}, opm_{t'}) \in d.cs$:
  $$\sigma(id_{t'})(''f'') = \sigma(id_{t'})(''sfa'') \,.\, \sigma(id_{t'})(''spc'') \tag{1.3}$$
  Rewriting the goal with (1.3): $\boxed{\sigma(id_{t'})(''sfa'') \,.\, \sigma(id_{t'})(''spc'') = \texttt{true}.}$
  Then, we can show that:
  – $\sigma(id_{t'})(''sfa'') = \texttt{true}$ by applying Lemma **??**

– $\sigma(id_{t'})("spc") = \texttt{true}$ by applying Lemma Stabilize Compute Priority Combination After Falling Edge.

ii. Assuming that $\sigma'(id_{t'})("f") = \texttt{true}$, let us show $\boxed{t' \in fired \cup \{t_0\} \ \vee \ t' \in T_s \setminus \{t_0\}.}$
From $\sigma'(id_{t'})("f") = \texttt{true}$ and EH, we can deduce that $t' \in fired \vee t' \in T_s$.
Case analysis on $t' \in fired \vee t' \in T_s$.

- **CASE** $t' \in fired$: then, it is trivial to show $t' \in fired \cup \{t_0\}.$
- **CASE** $t' \in T_s$: We know that $t_0 \in T_s$. Therefore, either $t' \in T_s \setminus \{t_0\}$, or $t' = t_0$, and then, $t' \in fired \cup \{t_0\}.$

3. **INDUCTIVE CASE**: $tp = \{t_0\} \cup tp_0$ and t_0 is not elected to be fired.

- $IsTopPrioritySet(T_s, \{t_0\} \cup tp_0)$
- $ElectFired(s', fired, tp_0, fired')$
- $IsFiredSetAux(s', fired', T_s \setminus \{t_0\} \cup tp_0, fset)$
- $\neg\big(t_0 \in Firable(s') \wedge t_0 \in Sens(s'.M - \sum_{t_i \in Pr(t, fired)} pre(t_i))\big)$
- EH:
  $\forall t' \in T, id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
  $(t' \in fired \Rightarrow \sigma'(id_{t'})("f") = \texttt{true}) \wedge (\sigma'(id_{t'})("f") = \texttt{true} \Rightarrow t' \in fired \ \vee \ t' \in T_s)$

$\forall T'_s \subseteq T,$
$IsTopPrioritySet(T'_s, tp_0) \Rightarrow$
$IsFiredSetAux(s', fired', T'_s \setminus tp_0, fset) \Rightarrow$
$\big(\forall t' \in T, id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
$(t' \in fired \Rightarrow \sigma'(id_{t'})("f") = \texttt{true}) \wedge (\sigma'(id_{t'})("f") = \texttt{true} \Rightarrow t' \in fired \ \vee \ t' \in T'_s)\big) \Rightarrow$
$\forall t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$,
$\big(t \in fired' \Rightarrow \sigma'(id_t)("f") = \texttt{true}\big) \wedge \big(\sigma'(id_t)("f") = \texttt{true} \Rightarrow t \in fired' \vee t \in T'_s \setminus tp_0\big)$

$\forall t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$,
$\big(t \in fired' \Rightarrow \sigma'(id_t)("f") = \texttt{true}\big) \wedge \big(\sigma'(id_t)("f") = \texttt{true} \Rightarrow t \in fired' \vee t \in T_s \setminus \{t_0\} \cup tp_0\big).$

Then, we can apply the induction hypothesis with $T'_s = T_s \setminus \{t_0\}$, then, there are three points to prove:

(a) $IsTopPrioritySet(T_s \setminus \{t_0\}, tp_0)$

(b) $IsFiredSetAux(s', fired', (T_s \setminus \{t_0\}) \setminus tp_0, fset)$

(c) $\forall t' \in T, id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
$(t' \in fired \Rightarrow \sigma'(id_{t'})("f") = \texttt{true}) \wedge (\sigma'(id_{t'})("f") = \texttt{true} \Rightarrow t' \in fired \vee t' \in T_s \setminus \{t_0\})$

Let us prove these three points:

(a) $IsTopPrioritySet(T_s \setminus \{t_0\}, tp_0)$

Not provable yet.

(b) $\boxed{IsFiredSetAux(s', fired', (T_s \setminus \{t_0\}) \setminus tp_0, fset)}$
We know that $(T_s \setminus \{t_0\}) \setminus tp_0 = T_s \setminus (\{t_0\} \cup tp_0)$, and thus
$IsFiredSetAux(s', fired', T_s \setminus (\{t_0\} \cup tp_0), fset)$ is an assumption.

(c) $\boxed{\forall t' \in T, id_{t'} \in Comps(\Delta) \text{ s.t. } \gamma(t') = id_{t'},\\ (t' \in fired \Rightarrow \sigma'(id_{t'})("f") = \texttt{true}) \wedge (\sigma'(id_{t'})("f") = \texttt{true} \Rightarrow t' \in fired \vee t' \in T_s \setminus \{t_0\})}$
Given a $t' \in T$ and an $id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$, let us show

$\boxed{(t' \in fired \Rightarrow \sigma'(id_{t'})("f") = \texttt{true}) \wedge (\sigma'(id_{t'})("f") = \texttt{true} \Rightarrow t' \in fired \vee t' \in T_s \setminus \{t_0\})}$

The proof is in two parts:

i. Assuming that $t' \in fired$, let us show $\boxed{\sigma'(id_{t'})("f") = \texttt{true}.}$
From $t' \in fired$ and EH, $\sigma'(id_{t'})("f") = \texttt{true}.$

ii. Assuming that $\sigma'(id_{t'})("f") = \texttt{true}$, let us show $\boxed{t' \in fired \vee t' \in T_s \setminus \{t_0\}.}$
Thanks to $\sigma'(id_{t'})("f") = \texttt{true}$ and EH, we know that: $t' \in fired \vee t' \in T_s$.
Case analysis on $t' \in fired \vee t' \in T_s$; there are two cases:

- **CASE** $t' \in fired.$
- **CASE** $t' \in T_s$:
  From $IsTopPrioritySet(T_s, \{t_0\} \cup tp_0)$, we can deduce that $t_0 \in T_s$. Therefore, either $t' \in T_s \setminus \{t_0\}$ or $t' = t_0$.
  In the case where $t' = t_0$, we need to show a contradiction by proving $t' \in Firable(s')$ and $t' \in Sens(s'.M - \sum_{t_i \in Pr(t, fired)} pre(t_i))$ based on $\sigma'(id_{t'})("f") = \texttt{true}$.
  By definition of $id_{t'}$, there exist a $gm_{t'}$, $ipm_{t'}$, $opm_{t'}$ s.t. $\texttt{comp}(id_{t'}, "transition", gm_{t'}, ipm_{t'}, opm_{t'}) \in d.cs$.
  By property of the stabilize relation and $\texttt{comp}(id_{t'}, "transition", gm_{t'}, ipm_{t'}, opm_{t'}) \in d.cs$:
  $$\sigma(id_{t'})("f") = \sigma(id_{t'})("sfa") \,.\, \sigma(id_{t'})("spc") = \texttt{true} \tag{1.4}$$
  From $\sigma(id_{t'})("sfa") = \texttt{true}$, and appealing to Lemma **??**, we can deduce $t' \in Firable(s')$.
  From $\sigma(id_{t'})("spc") = \texttt{true}$, and appealing to Lemma Stabilize Compute Priority Combination After Falling Edge, we can deduce $t' \in Sens(s'.M - \sum_{t_i \in Pr(t, fired)} pre(t_i))$.
  Then, as $t' = t_0$, $\neg\big(t_0 \in Firable(s') \wedge t_0 \in Sens(s'.M - \sum_{t_i \in Pr(t, fired)} pre(t_i))\big)$ is a contradiction.

□

Lemma 9 (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 Def. 9, then $\forall t \in T, id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, $\forall fired, fired', T_s, tp, fset \subseteq T$ assume that:*

- $IsTopPrioritySet(T_s, \{t\} \cup tp)$
- $ElectFired(s', fired, tp, fired')$
- $FiredAux(s', fired', T_s \setminus \{t\} \cup tp, fset)$
- EH: $\forall t' \in T, id_{t'} \in Comps(\Delta)$ s.t. $\gamma(t') = id_{t'}$,
  $(t' \in fired \Rightarrow \sigma'(id_{t'})(''f'') = \texttt{true}) \wedge (\sigma'(id_{t'})(''f'') = \texttt{true} \Rightarrow t' \in fired \ \vee \ t' \in T_s)$.
- $t \in Firable(s')$

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

Proof. Given a $t \in T$ and an $id_t \in Comps(\Delta)$ s.t. $\gamma(t) = id_t$, a $fired$, $fired'$, T_s, tp, $fset \subseteq T$ and assuming all the above hypotheses, let us show

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

By definition of id_t, there exist gm_t, ipm_t, opm_t s.t. $\texttt{comp}(id_t, ''transition'', gm_t, ipm_t, opm_t) \in d.cs$.
By property of the stabilize relation and $\texttt{comp}(id_t, ''transition'', gm_t, ipm_t, opm_t) \in d.cs$:

$$\sigma'(id_t)(''spc'') = \prod_{i=0}^{\Delta(id_t)(''ian'')-1} \sigma'(id_t)(''pauths'')[i] \tag{1.5}$$

Rewriting the goal with (1.5):

$$\boxed{t \in Sens(s'.M - \sum_{t_i \in Pr(t,fired)} pre(t_i)) \Leftrightarrow \prod_{i=0}^{\Delta(id_t)(''ian'')-1} \sigma'(id_t)(''pauths'')[i] = \texttt{true}.}$$

Then, the proof is in two parts:

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

Let us prove both sides of the equivalence:

1. Assuming that $t \in Sens(s'.M - \sum_{t_i \in Pr(t,fired)} pre(t_i))$, let us show

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

   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
     $<\texttt{priority\_authorizations}(0) \Rightarrow \texttt{true}> \in ipm_t$.
     By property of the elaboration relation, we have $\Delta(id_t)(''ian'') = 1$, and by property of the stabilize relation, we have $\sigma'(id_t)(''pauths'')[0] = \texttt{true}$.
     Rewriting the goal with $\Delta(id_t)(''ian'') = 1$ and $\sigma'(id_t)(''pauths'')[0] = \texttt{true}$, and simplifying the goal: tautology.

- **CASE** $input(t) \neq \emptyset$:
  Then, let us show an equivalent goal:
  $\boxed{\forall i \in [0, \Delta(id_t)(\text{"}ian\text{"}) - 1],\ \sigma'(id_t)(\text{"}pauths\text{"})[i] = \texttt{true}.}$
  Given an $i \in [0, \Delta(id_t)(\text{"}ian\text{"}) - 1]$, let us show $\boxed{\sigma'(id_t)(\text{"}pauths\text{"})[i] = \texttt{true}.}$
  By construction, $<\texttt{input\_arcs\_number} \Rightarrow |input(t)|> \in gm_t$.
  By property of the elaboration relation, we have $\Delta(id_t)(\text{"}ian\text{"}) = |input(t)|$. 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 gm_p, ipm_p, opm_p s.t. $\texttt{comp}(id_p, \text{"}place\text{"}, gm_p, ipm_p, opm_p) \in d.cs$, and there exist a $j \in [0, |output(p)|]$ and an $id_{ji} \in Sigs(\Delta)$ s.t. $<\texttt{input\_arcs\_valid(i)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in ipm_t$ and $<\texttt{output\_arcs\_valid(j)} \Rightarrow \texttt{id}_{\texttt{ji}}> \in opm_t$.
  Let us take such a $p \in input(t)$, $id_p \in Comps(\Delta)$, gm_p, ipm_p, opm_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, $<\texttt{priority\_authorizations(i)} \Rightarrow \texttt{true}> \in ipm_t$, and by property of the stabilize relation: $\sigma'(id_t)(\text{"}pauths\text{"})[i] = \texttt{true}.$
  - **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, $<\texttt{priority\_authorizations(i)} \Rightarrow \texttt{true}> \in ipm_t$, and by property of the stabilize relation: $\sigma'(id_t)(\text{"}pauths\text{"})[i] = \texttt{true}.$
    * **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.
      $<\texttt{priority\_authorizations(i)} \Rightarrow \texttt{id}'_{\texttt{ji}}> \in ipm_t$ and
      $<\texttt{priority\_authorizations(j)} \Rightarrow \texttt{id}'_{\texttt{ji}}> \in opm_p$.
      By property of the stabilize relation, $\texttt{comp}(id_t, \text{"}transition\text{"}, gm_t, ipm_t, opm_t) \in d.cs$ and $\texttt{comp}(id_p, \text{"}place\text{"}, gm_p, ipm_p, opm_p) \in d.cs$:

$$\sigma'(id_t)(\text{"}pauths\text{"})[i] = \sigma'(id'_{ji}) = \sigma'(id_p)(\text{"}pauths\text{"})[j] \tag{1.6}$$

      Rewriting the goal with (1.6): $\boxed{\sigma'(id_p)(\text{"}pauths\text{"})[j] = \texttt{true}.}$
      By property of the stabilize relation and $\texttt{comp}(id_p, \text{"}place\text{"}, gm_p, ipm_p, opm_p) \in d.cs$:

$$\sigma'(id_p)(\text{"}pauths\text{"})[j] = (\sigma'(id_p)(\text{"}sm\text{"}) \geq \texttt{rsum} + \sigma'(id_p)(\text{"}oaw\text{"})[j]) \tag{1.7}$$

      Let us define the rsum term as follows:

$$\texttt{rsum} = \sum_{i=0}^{j-1} \begin{cases} \sigma'(id_p)(\text{"}oaw\text{"})[i] \texttt{ if } \sigma'(id_p)(\text{"}otf\text{"})[i]. \\ \qquad\qquad\qquad\qquad \sigma'(id_p)(\text{"}oat\text{"})[i] = \texttt{basic} \\ 0 \ otherwise \end{cases} \tag{1.8}$$

Rewriting the goal with (1.7): $\boxed{\sigma'(id_p)("sm") \geq \texttt{rsum} + \sigma'(id_p)("oaw")[j]}$
By definition of $t \in Sens(s'.M - \sum_{t_i \in Pr(t,fired)} pre(t_i))$, we have $s'.M(p) \geq \sum_{t_i \in Pr(t,fired)} pre(p,t_i) + \omega$.
Then, there are three points to prove:

(a) $\boxed{s'.M(p) = \sigma'(id_p)("sm")}$

(b) $\boxed{\omega = \sigma'(id_p)("oaw")[j]}$

(c) $\boxed{\sum_{t_i \in Pr(t,fired)} pre(p,t_i) = \texttt{rsum}}$

Let us prove these three points:

(a) $\boxed{s'.M(p) = \sigma'(id_p)("sm")}$

Appealing to Lemma **??**: $s'.M(p) = \sigma'(id_p)("sm")$.

(b) $\boxed{\omega = \sigma'(id_p)("oaw")[j]}$

By construction, and as $pre(p,t) = (\omega, \texttt{basic})$, we have
$<\texttt{output\_arcs\_weights(j)} \Rightarrow \omega> \in ipm_p$.
By property of the stabilize relation and $\texttt{comp}(id_p, "place", gm_p, ipm_p, opm_p) \in d.cs$:
$\omega = \sigma'(id_p)("oaw")[j]$.

(c) $\boxed{\sum_{t_i \in Pr(t,fired)} pre(p,t_i) = \texttt{rsum}}$

Let us replace the left and right term of the equality by their full definition:

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

Let us define $f(t_i) = \begin{cases} \omega \texttt{ if } pre(p,t_i) = (\omega, \texttt{basic}) \\ 0 \textit{ otherwise} \end{cases}$ and

$$g(i) = \begin{cases} \sigma'(id_p)("oaw")[i] \texttt{ if } \sigma'(id_p)("otf")[i]. \\ \qquad\qquad \sigma'(id_p)("oat")[i] = \texttt{basic} \\ 0 \textit{ otherwise} \end{cases}$$

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

BASE CASE: then, we have $i > j - 1$, and then $j = 0$.

$$\sum_{t_i \in Pr(t,fired)} \begin{cases} \omega \texttt{ if } pre(p,t_i) = (\omega, \texttt{basic}) \\ 0 \ otherwise \end{cases} = 0$$

We know that the priority relation is a strict total order over the transitions of set $output_c(p)$. This ordering is reflected in the ordering of the indexes of output port `priority_authorizations` of place component instances. Thus, in the `priority_authorizations` output port of a place component instance, the element of index 0 is connected to the transition of $output_c(t)$ with the highest firing priority. We know that component id_t is connected to `priority_authorizations(0)` in the output port map of component id_p. By construction, transition t is the transition of $output_c(p)$ with the highest firing priority, i.e, $\nexists t' \in output_c(p)$ s.t. $t' \succ t$.

The following part of the proof is the result of induction over term $\sum_{t_i \in Pr(t,fired)} f(t_i)$.
Induction is not detailled here.

For all transition $t_i \in Pr(t, fired)$, either t_i is not in $output_c(p)$, and thus t_i has no effect in the value of the sum term $\sum_{t_i \in Pr(t,fired)} f(t_i)$; or, $t_i \in output_c(p)$. Then, by definition of $t_i \in Pr(t, fired)$, $t_i \succ t$, which is contradiction with $\nexists t' \in output_c(p)$ s.t. $t' \succ t$.

INDUCTIVE CASE: then, $0 \leq j-1$, and thus $j > 0$.

For all $Pr' \subseteq T$, $g(0) + \sum_{t_i \in Pr'} f(t_i) = g(0) + \sum_{i=1}^{j-1} g(i)$

$$\sum_{t_i \in Pr(t,fired)} f(t_i) = g(0) + \sum_{i=1}^{j-1} g(i).$$

By definition of $g(0)$:

$$\sum_{t_i \in Pr(t,fired)} f(t_i) = \begin{cases} \sigma'(id_p)(\text{"}oaw\text{"})[0] \texttt{ if } \sigma'(id_p)(\text{"}otf\text{"})[0]. \\ \qquad\qquad \sigma'(id_p)(\text{"}oat\text{"})[0] = \texttt{basic} \\ 0 \ otherwise \end{cases} + \sum_{i=1}^{j-1} g(i).$$

Case analysis on the value of $\sigma'(id_p)(\text{"}otf\text{"})[0] \,.\, \sigma'(id_p)(\text{"}oat\text{"})[0] = \texttt{basic}$:

In the case where $\left(\sigma'(id_p)(\text{"}otf\text{"})[0] \,.\, \sigma'(id_p)(\text{"}oat\text{"})[0] = \texttt{basic}\right) = \texttt{false}$, then $g(0) = 0$, and we can use the induction hypothesis with $Pr' = Pr(t, fired)$ to prove the goal.

In the case where $\left(\sigma'(id_p)(\text{"}otf\text{"})[0] \,.\, \sigma'(id_p)(\text{"}oat\text{"})[0] = \texttt{basic}\right) = \texttt{true}$, then $g(0) = \sigma'(id_p)(\text{"}oaw\text{"})[0]$:

$$\sum_{t_i \in Pr(t,fired)} f(t_i) = \sigma'(id_p)(''oaw'')[0] + \sum_{i=1}^{j-1} g(i).$$

By construction, and knowing that $j > 0$ and that the priority relation is a strict total order over the set $output_c(p)$, there exist a $t_0 \in output_c(p)$ s.t. $t_0 \succ t$. Moreover, there exist an $id_{t_0} \in Comps(\Delta)$ s.t. $\gamma(t_0) = id_{t_0}$, and by definition of id_{t_0}, there exist gm_{t_0}, ipm_{t_0} and opm_{t_0} s.t. $\texttt{comp}(id_{t_0}, ''transition'', gm_{t_0}, ipm_{t_0}, opm_{t_0}) \in d.cs$. Finally, there exist an $id_{ft_0} \in Sigs(\Delta)$ s.t. $<\texttt{fired} \Rightarrow \texttt{id}_{\texttt{ft}_0}> \in opm_{t_0}$ and $<\texttt{output\_transitions\_fired}(0) \Rightarrow \texttt{id}_{\texttt{ft}_0}> \in ipm_p$.
By property of the stabilize relation, $\texttt{comp}(id_p, ''place'', gm_p, ipm_p, opm_p) \in d.cs$ and $\texttt{comp}(id_{t_0}, ''transition'', gm_{t_0}, ipm_{t_0}, opm_{t_0}) \in d.cs$:

$$\sigma'(id_{t_0})(''f'') = \sigma'(id_{ft_0}) = \sigma'(id_p)(''otf'')[0] = \texttt{true} \tag{1.9}$$

From EH and $\sigma'(id_{t_0})(''f'') = \texttt{true}$, we have either $t_0 \in fired$ or $t_0 \in T_s$.

❑ In the case where $t_0 \in fired$, then, by definition of $\sum$:

$$f(t_0) + \sum_{t_i \in Pr(t,fired)\setminus\{t_0\}} f(t_i) = \sigma'(id_p)(''oaw'')[0] + \sum_{i=1}^{j-1} g(i).$$

By definition of $t_0 \in output_c(p)$, there exists $\omega \in \mathbb{N}^*$ s.t. $pre(p, t_0) = (\omega, \texttt{basic})$. Thus, we have $f(t_0) = \omega$
By construction, $<\texttt{output\_arcs\_weights}(0) \Rightarrow \omega>$, and by property of the stabilize relation, we have $\sigma'(id_p)(''oaw'')[0] = \omega$. Thus, we can deduce that $g(0) = \omega$, and then we can rewrite the goal in order to apply the induction hypothesis with $Pr' = Pr(t, fired) \setminus \{t_0\}$.

❑ In the case where $t_0 \in T_s$:
As t is a top-priority transition in set T_s, there exists no transition $t' \in T_s$ s.t. $t' \succ t$.
Contradicts $t_0 \succ t$.

2. Assuming that $\prod_{i=0}^{\Delta(id_t)(''ian'')-1} \sigma'(id_t)(''pauths'')[i] = \texttt{true}$, let us show

$$t \in Sens(s'.M - \sum_{t_i \in Pr(t,fired)} pre(t_i)).$$

By definition of $t \in Sens(s'.M - \sum_{t_i \in Pr(t,fired)} pre(t_i))$:

$$\begin{aligned}&\forall p \in P, \omega \in \mathbb{N}^*,\\ &\big((pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})) \Rightarrow s'.M(p) - \sum_{t_i \in Pr(t,fired)} pre(p,t_i) \geq \omega\big)\\ &\wedge \big(pre(p,t) = (\omega, \texttt{inhib}) \Rightarrow s'.M(p) - \sum_{t_i \in Pr(t,fired)} pre(p,t_i) < \omega\big)\end{aligned}$$

Given a $p \in P$ and an $\omega \in \mathbb{N}^*$, let us show

$$\big((pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})) \Rightarrow s'.M(p) - \sum_{t_i \in Pr(t, fired)} pre(p, t_i) \geq \omega\big)$$
$$\wedge \big(pre(p,t) = (\omega, \texttt{inhib}) \Rightarrow s'.M(p) - \sum_{t_i \in Pr(t, fired)} pre(p, t_i) < \omega\big)$$

By construction, there exists an $id_p \in Comps(\Delta)$ s.t. $\gamma(p) = id_p$. By definition of id_p, there exist gm_p, ipm_p, opm_p s.t. $\texttt{comp}(id_p, "place", gm_p, ipm_p, opm_p) \in d.cs$.

There are three different cases:

(a) Assuming that $pre(p,t) = (\omega, \texttt{test})$, let us show $\boxed{s'.M(p) - \sum_{t_i \in Pr(t, fired)} pre(p, t_i) \geq \omega.}$

Then, assuming that the priority relation is well-defined, there exists no transition t_i connected by a `basic` arc to p that verified $t_i \succ t$. 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 output of p.
Then, we can deduce that $\sum_{t_i \in Pr(t, fired)} pre(p, t_i) = 0$.
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 $\boxed{s'.M(p) - \sum_{t_i \in Pr(t, fired)} pre(p, t_i) < \omega.}$

Use the same strategy as above.

(c) Assuming that $pre(p,t) = (\omega, \texttt{basic})$, let us show $\boxed{s'.M(p) - \sum_{t_i \in Pr(t, fired)} pre(p, t_i) \geq \omega.}$

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 $\sum_{t_i \in Pr(t, fired)} pre(p, t_i) = 0$.
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 definition of id_t, there exist gm_t, ipm_t, opm_t s.t. $\texttt{comp}(id_t, "transition", gm_t, ipm_t, opm_t) \in d.cs$.
By construction, there exist $j \in [0, |input(t)| - 1]$, $k \in [0, |output(t)| - 1]$, and $id_{kj} \in Sigs(\Delta)$ s.t. $<\texttt{priority\_authorizations(j)} \Rightarrow \texttt{id}_{\texttt{kj}}> \in ipm_t$ and $<\texttt{priority\_authorizations(k)} \Rightarrow \texttt{id}_{\texttt{kj}}> \in opm_p$. Let us take such an j, k and id_{kj}.
From $\prod_{i=0}^{\Delta(id_t)("ian")-1} \sigma'(id_t)("pauths")[i] = \texttt{true}$, we can deduce that for all $i \in [0, \Delta(id_t)("ian") - 1]$, $\sigma'(id_t)("pauths")[i] = \texttt{true}$.
By construction, $<\texttt{input\_arcs\_number} \Rightarrow |input(t)|> \in gm_t$, and by property of the elaboration relation, we have $\Delta(id_t)("ian") = |input(t)|$. Then, from $j \in [0, |input(t)| -$

$1]$, we can deduce $j \in [0, \Delta(id_t)("ian") - 1]$. And, from $\forall i \in [0, \Delta(id_t)("ian") - 1]$, $\sigma'(id_t)("pauths")[i] = \texttt{true}$, we can deduce $\sigma'(id_t)("pauths")[j] = \texttt{true}$.
By property of the stabilize relation, $\texttt{comp}(id_p, "place", gm_p, ipm_p, opm_p) \in d.cs$ and $\texttt{comp}(id_t, "transition", gm_t, ipm_t, opm_t) \in d.cs$:

$$\sigma'(id_p)("pauths")[k] = \sigma'(id_{kj})\sigma'(id_t)("pauths")[j] = \texttt{true} \tag{1.10}$$

By property of the stabilize relation and $\texttt{comp}(id_p, "place", gm_p, ipm_p, opm_p) \in d.cs$:

$$\sigma'(id_p)("pauths")[k] = (\sigma'(id_p)("sm") \geq \texttt{rsum} + \sigma'(id_p)("oaw")[k]) \tag{1.11}$$

Let us define the `rsum` term as follows:

$$\texttt{rsum} = \sum_{i=0}^{k-1} \begin{cases} \sigma'(id_p)("oaw")[i] \texttt{ if } \sigma'(id_p)("otf")[i]. \\ \qquad\qquad\qquad\qquad \sigma'(id_p)("oat")[i] = \texttt{basic} \\ 0 \textit{ otherwise} \end{cases} \tag{1.12}$$

From (1.10) and (1.11), we can deduce that $\sigma'(id_p)("sm") \geq \texttt{rsum} + \sigma'(id_p)("oaw")[k]$.
Then, there are three points to prove:

A. $\boxed{s'.M(p) = \sigma'(id_p)("sm")}$

B. $\boxed{\omega = \sigma'(id_p)("oaw")[k]}$

C. $\boxed{\sum_{t_i \in Pr(t, fired)} pre(p, t_i) = \texttt{rsum}}$

See 1 for the remainder of the proof.

□

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] 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).

[3] 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).

[4] Timothy Bourke et al. "A Formally Verified Compiler for Lustre". In: (), p. 17.

[5] 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.

[6] 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.

[7] 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).

[8] 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.

[9] 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.

[10] 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.

[11] 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).

[12] 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).

[13] 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.

[14] 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).

[15] Yong Kiam Tan et al. "A New Verified Compiler Backend for CakeML". In: (), p. 14.

[16] 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).

[17] 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).