A Game Semantics for Instantaneous Esterel Reactions*

Joaquín Aguado

Faculty of Information Systems and Applied Computer Sciences, University of Bamberg, An der Weberei 5 (WE5/05.040) 96047, Bamberg, Germany, joaquin.aguado@uni-bamberg.de

The present paper explains the constructive semantics of the synchronous language ESTEREL in terms of winning strategies in finite two–player games. The kernel fragment of ESTEREL considered here corresponds to combinational circuits and includes the statements for signal emission, status test, parallel composition and gives (for the first time) a game–theoretic interpretation to sequential composition and local signal declaration. This modelling shows how non-classical truth values induced by logic games can be used to characterise the constructive single-step semantics for ESTEREL.

Index Terms : ***Synchronous languages, ESTEREL, constructive semantics, causality cycles, game theory***

I. Introduction

The synchronous programming technology has turned into a highly successful approach in the design of embedded and reactive systems, in particular in the automotive and avionics industries and, more generally, in the design of safety–critical systems [1]. In these domains the success of synchronous languages is equally owed to the fact that they are model–based and high–level but above all because these languages are based on a simple and powerful execution model. This model is thought to be scheduled under the regime of an implicit global clock (which may be physical or logical) that delimits a sequence of "present" *instants* or *macro–steps* by ticks. At every macro–step, the system delivers a full response to an external stimulus imposed by the environment. The abstraction that governs and describes this interplay is the *synchrony hypothesis* or a *maximal progress, run to completion* scheduling scheme [2], [3]. This hypothesis assumes that the environment is significantly slower than the system itself. Specifically, in this canonical setting, there is no way in which the environment could bring under control or monitor any activity of the system between two ticks. As a result, the reactions appear to the environment as atomic events in which the set of inputs and recognisable outputs are simultaneous. During an instant, however, internal subsystems interact interchanging and combining information that goes back and forward until environment and system are able to synchronise again when the clock ticks. The postulate of the synchrony hypotheses abstracts all the internal signal exchanges and low–level behaviour needed to produce a response and ensures that this internal operation eventually stops completing a step response. This compactification of the synchronous model reconciles concurrency and determinism, and makes up much of its algebraic appeal. For a more general introduction the reader is referred to [2], [4].

The synchronous reactive macro–step abstraction can be explained through a system model built up from parallel (simultaneous) transitions (as implications) of the form:
$(p_1 \wedge \ldots p_i \wedge \neg n_1 \wedge \cdots \wedge \neg n_j) \supset (v_1 \wedge \cdots \wedge v_k)$,

specifying the reaction *"if all the signals p_i are present and all the signals n_j are absent, then all of the v_k signals are emitted (become present)"*. Thus, inside the system, signals flood all over the place where the transitions await for the appropriate combination of them to get enabled and emit new signals. The presence of negative triggers (a source of non-monotonicity) has resulted in various semantics. It is agreed, in general terms, that a global scheduler selects some (at least one) of the enabled transitions (*i.e.*, those for which *"all its p_i signals are present and all its n_j signals are absent"*) and executes them in parallel. The outcome is an emission (*i.e., "then all of the v_k signals are emitted"*) that enables other transitions, some of which will be chosen by the scheduler to be executed and so on until the system stabilises (ideally) and produces a response. The main issue arises in how to deal with the inhibitive nature of negative triggering conditions (*i.e.*, the signals that must be absent) in order to avoid inconsistences due to circular causality. As a simple example of the causality issue consider a system P consisting of the transitions $t_1 := \neg x \supset y$ and $t_2 := y \supset x$. Assume P starts its execution with an empty stimulus (*i.e.*, no signal is initially given by the environment). Inside P, when the instant begins, only transition t_1 is enabled since x is absent. Thus, t_1 is executed emitting signal y, then transition t_2 gets enabled and fires resulting in the presence of signal x. Nevertheless, from outside P, t_1 and t_2 happen instantaneously. The problem is that the presence of x depends on the present status of y which, in turn, results from the absence of x at the same time.

There is not a single canonical way to handle this. Likely, the full range of possibilities have not been explored yet, but already the causality problem has motivated different solutions adopted in the literature on synchronous languages. Let us mention some of them. In Modecharts [5] negative triggers

*This work has been supported by the German Science Foundation, as part of the PRETSY project DFG HA 4407/6-1

are not allowed at all. Both (synchronous) Statemate [6] and UML [7] only schedule causally independent transitions preventing with this inter–transition communication in a macro–step. Negations with implicit delays refereing to the previous instant have been suggested in the context of concurrent constraint programming [8]. Syntactic restrictions in stratified logic programs that avoid recursion on negations are other possibility [9]. For Statecharts [10], the absence of signals is a local condition (for triggering transitions) that later in the same macro–step could become present leaving aside global consistency [11]. This is like in the example above, where x is assumed to be absent when triggering t_1, but x is emitted later (in the same instant) by t_2. Besides, also for Statecharts, global consistency has been considered an implicit trigger [12], [13]. More specifically, as the chain reaction of firing transitions evolves if it happens that an enabled transition emits a signal previously considered absent then this transition (although enabled) is not triggered. Thus, in the previous example neither t_1 nor t_2 would be triggered because t_2 emits x that must be absent (in the first place) for triggering t_1. Then, this would result in the absence of both signals x; y. The Statecharts semantics of Pnueli & Shalev [14] is based on a search of consistent scheduling sequences of transitions by speculating on the absence of signals and backtracking when required. In ESTEREL [2], [15], [16], [17], the language studied in this paper, synchrony and causality are treated separately, so only conflict–free and deterministic programs are accepted where negations are considered "positive" absences. This is in line with the view that consistency in computing a response should not depend on any particular cleverness of the scheduler. Scheduling under every possible input should be confluent and produce the same uniquely determined result.

The variety of semantics arising from the different options of handling negation cannot be simply carried out within the framework of classical logic or simple set-theoretical models, which justifies the game–theoretic approach of this paper. This should be not surprising since classical logic is not fine enough to model all the intensional aspects of scheduling under inhibiting as well as enabling effects. For instance, the single truth–value *false* cannot adequately model the meaning of negation $\neg x$ when x is initially absent but occurring later. What is surprising is that for certain coherent scheduling regimes it is possible to maintain the abstract propositional viewpoint, *i.e.*, avoid the complications of modelling scheduling sequences in detail, simply by choosing a constructive (intuitionistic) logic interpretation. For instance, in [18] it was shown that the simple twist of replacing the classical two–valued by an intuitionistic interpretation of signals (specifically, three–valued Gödel logic) suffices to obtain a fully abstract and compositional model for the original macro step semantics of Statecharts as given by Pnueli & Shalev [14]. Following a similar direction, the present paper explains the constructive semantics of ESTEREL naturally in terms of winning strategies in finite two—player games. This modelling shows how non–classical truth values induced by logic games can be used to characterise the constructive single–step semantics for ESTEREL–like languages.

The classical theory of games, originally developed in descriptive set theory, has emerged as a surprisingly versatile mathematical tool. Moreover, the intensionality of the game model has reconciliated the algebraic and operational views in the semantics of proofs and programming languages [19]. The power of the games model rests on its ability to handle combinatorially complex situations, such as the alternate nesting of quantifiers, in a natural and intuitive fashion [20]. Perhaps the most prominent examples of successful application are the game–theoretic solution of the full–abstraction problem for the functional language PCF [21] (which had been open for a long time) and the game–theoretic analysis of proofs in multiplicative linear logic [22], [23], [24], [25]. This has led to new approaches in the field of control–flow analysis [26], integrating imperative, object–oriented, higher–order functional, and concurrent features.

Games are a convincing metaphor not only in functional programming but also in the field of reactive–systems modeling. This is because the interaction between a reactive system and its environment has a strong analogy in the moves between a player and his or her opponent in a simple two–player maze game. This interaction problem is then solved by providing a winning strategy that in turn may be understood as a system reaction. In this paper we report on a novel application of this metaphor to the specific interaction problem that arises in synchronous programming under the synchrony hypothesis, namely the characterisation of *present* and *absent* signals within a system's reaction under a given environment. Specifically, using Berry's language ESTEREL [2] as an example, we prove that the underlying intricate constructive semantics of reactions can be captured in a very natural game–theoretic manner. Thus, the objective of this paper is to show that programs written in the kernel fragment of ESTEREL corresponding to combinational circuits can be understood very naturally as maze game boards, where signals are represented by rooms. Intuitively, the presence and absence of signals in the reaction instant that is described by a combinational program P, is "negotiated" between the system (the starting player) and its environment (the opponent). The system tries to prove a signal's presence and the environment its absence. Thus, signal s must (cannot) be emitted in P if and only if room s in the maze M associated with P is a winning (losing) position. If the status of signal s is undefined, then and only then room s is a draw position.

2. Playing the Maze Game

In this section we present a two–player and perfect information game, called the *maze game*. A maze can be thought as a graph that encodes the transitions of a reactive (synchronous) component. The players take turns to move a token from room (vertex) to room through the corridors (edges) of the maze. The winning conditions define which rooms are winning, losing or draw positions, this allows us to classify signals as present, absent and non–constructive.

In the maze game, the board consists of *rooms* which

are connected by one–way *corridors* and the game figure is just a single token. Corridors can be of two types: *visible* and *secret*. Placing the token in some arbitrary room, the starting player, say Jaakko, may move the token from one room to the next through arbitrarily many secret corridors; however, as soon as Jaakko moves through a visible corridor, his turn ends. Control passes on to the opponent, say Leon, who may continue in a similar fashion from the current position of the token on the board[1]. If the token gets stuck in a *dungeon, i.e.*, a room with no outgoing corridors, the current player loses and the opponent wins. Hence, the objective of the game is to drive the opponent into a *dungeon*. Obviously, a game board can be modeled as a finite directed graph, such as the one depicted in Fig. 1, where nodes are rooms, solid edges represent visible corridors, and dashed edges represent secret corridors.

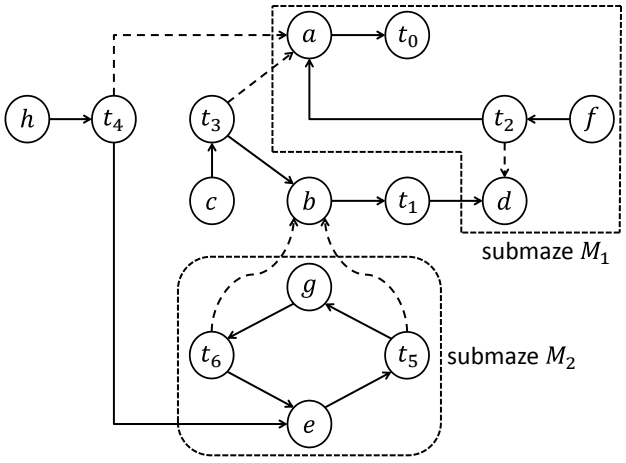


Fig. 1 : Example game board M_{ex}.

Given a game board and choosing an initial room for the token, this room may now be classified according to whether the starting player Jaakko (*i*) has the possibility always to win (if he plays cleverly), (*ii*) must always lose (no matter how cleverly he plays), or (*iii*) can at best reach a draw by ensuring that Leon can never force him into a dungeon. For instance, using the board M_{ex} illustrated in Fig. 1 and initially placing the token in room b, Jaakko can move the token to room t_1 through a visible corridor, thereby handing over control to Leon. Now Leon can *only* move the token into dungeon d, again through a visible corridor, such that control passes back to Jaakko who instantaneously loses. This implies that room b is a *losing position* (for the starting player Jaakko). However, if instead the token is initially placed in room t_3, then Jaakko has two strategies for winning. On the one hand, Jaakko may move the token through a visible corridor to room b where, as we have just seen, player Leon will necessarily lose after two more turns. On the other hand, Jaakko may use the secret corridor to move the token into room a and in the same turn further into dungeon t_0 via a visible corridor, thus winning the game. Jaakko is said to have a winning strategy from room t_3, and t_3 is referred to as a winning position. However, a game may also end in a draw. For example, suppose the token is placed initially in room t_5. Then Jaakko has several alternatives, but one of these leaves him in the hands of Leon. Indeed, if Jaakko moves the token to room b, Leon can place the token into dungeon d. Observe that if the token is instead placed in t_6, the situation is similar in the sense that moving the token in room b will result in losing the game. Now, assuming that both players want to win and that they both know that placing the token in room b is the worst option, it follows that they will keep moving the token all the time through submaze M_2 of Fig. 1. Since inside M_2 it is always possible to avoid room b, the game can continue indefinitely in this fashion leading to a draw. Hence rooms *e, g,* t_5 and t_6 are referred to as a draw positions.

3. Pure Esterel

Before turning our attention to game semantics, it is first necessary to specify a language in which these ideas can be realised. The language we are going to employ, as an example through the paper and as a vehicle of concretisation, is the synchronous language ESTEREL [2]. This section formally presents the combinational fragment of ESTEREL which we are concerned with and recalls its constructive behavioral semantics as defined by Berry in [17].

3.1 The Language

ESTEREL is an imperative, textual and parallel synchronous programming language, which is tailored for programming control dominant systems. An ESTEREL program has an interface which distinguish input and output *signals*. The information contained in a signal is restricted to a *presence* or *absence status*. An *environment stimulus* (or *input event*) defines the status of every input signal and a *system response* (or *output event*) gives status to all the output signals. The abstract syntax of the ESTEREL fragment, which specifies single reactions and will be used in the remainder, is defined by the BNF of Fig. 2, where s stands for a signal name taken from some (finite) universe $\mathcal{S}$.

$$
\begin{array}{rcll}
P & ::= & 0 & \texttt{nothing} \\
 & | & !s & \texttt{emit } s \\
 & | & s^{+}?(P) & \texttt{present } s \texttt{ then } P \texttt{ end} \\
 & | & s^{-}?(P) & \texttt{present } s \texttt{ else } P \texttt{ end} \\
 & | & P \,|\, P & P \texttt{ || } P \\
 & | & P ; P & P \texttt{ ; } P \\
 & | & P \backslash s & \texttt{signal } s \texttt{ in } P \texttt{ end}
\end{array}
$$

Fig. 2 : ESTEREL fragment Syntax.

ESTEREL's more general choice statement "present s then P_1 else P_2 end" can be recovered in our syntax by the term $s^{+}?(P_1) \mid s^{-}?(P_2)$ [18]. By default sequential composition (;) binds tighter than parallel composition (|). As in many

[1] The players are named after the logicians Jaakko Hintikka, who pioneered the field of semantic games for logic, and Leon Henkin, who first introduced game–theoretic interpretations of quantifiers.

languages, brackets can be used to group statement arbitrary. The language manipulates control flow and signal status. The statement 0 has no effect and terminates immediately. An emit statement !s broadcast instantaneously signal s (*i.e.*, its status is set to present) and terminates. The branching of control $s^+?(P)$ ($s^-?(P)$) tests the status of signal s, if this is present (absent) then P is immediately executed. Parallel composition $P_1 \mid P_2$ executes the statements of P_1 and P_2 in parallel. If, at any given point in the execution, one of the components terminates then the computation continues with the other component alone. The statement terminates instantaneously if and only if both P_1 and P_2 terminate. Sequential composition $P_1 ; P_2$ executes P_1 until it terminates. If so, the control is transferred instantaneously to P_2 which determines the behaviour of the full composition from then on. In a local signal declaration $P \backslash s$ the statement of P are executed with a fresh signal s that overrides any other occurrence of s that might already exist. This statement terminates when P terminates but the status of the local signal s is not available outside P [27].

In this paper we do away with input signals $i \in I = \{i_1, \ldots, i_n\} \subseteq \mathcal{S}$. This is possible since the behavior of P under I is equivalent to the behavior of $P \mid !i_{j_1} \mid \ldots \mid !i_{j_m}$, where the indexes $j_1; \ldots; j_m \in \{1, \ldots, n\}$ are exactly those for which signal i_{j_k} is present in I [18]. Finally, we further assume that the finite set $\mathcal{S}$ of signals includes all signals of the combinational ESTEREL programs that one wishes to reason about. This restriction is a mere technical convenience and could be dropped by maintaining for every program the finite set of relevant signals, *i.e.*, a "signal sort."

3.2 Behavioural Semantics

The behavioural semantics for ESTEREL considers *statuses* of signals. Each signal $s \in \mathcal{S}$ can either be known to be present, *i.e.*, have status s^+, or known to be absent, *i.e.*, have status s^-, or have an unknown status. For any set $S \subseteq \mathcal{S}$ we let S^+ stand for $\{s^+ \mid s \in S\}$ and S^- for $\{s^- \mid s \in S\}$. *Consistent* sets $E \subseteq \mathcal{S}^+ \cup \mathcal{S}^-$ of signal statuses such that $\nexists s.\ s^+ \in E$ and $s^- \in E$ are referred to as events, with ε denoting the set of all events.

Given a combinational ESTEREL program P and an event E, to compute the signal statuses we need three functions: *must*(P,E), *cannot*(P,E) and t(P,E). The first two functions determine the sets of signals that must and cannot be emitted (*i.e.*, be present or absent) in P, respectively, relative to the knowledge of the signal statuses in E. The *must* and *cannot* computations are displayed in Fig. 3 and 4. Note that the *cannot*$^+$ has the same definition as the *cannot* except for $P \backslash s$. The predicate t(P,E) is true if P is known to terminate in environment E. This predicate is defined in Fig. 5. Notice that ; and | have almost the same definition, except that *must*(P_1 ; P_2, E) depends on a termination condition as given by the predicate t(P,E).

The functions *must*, *cannot* and t are monotonic in E for subset inclusion. Since $(\varepsilon, \subseteq)$ is a finite $\cap$–semi–lattice, the function

$$\mathit{esterel}(P)(E) := \mathit{must}(P,E)^+ \cup \mathit{cannot}(P,E)^-$$

has a least fixed point

$$\mu\mathit{esterel}(P) = \bigcup_{i \in \mathbb{N}} \mathit{esterel}(P)^i(\emptyset)$$

this least fixed point defines the behavioral semantics of P.

In [17] the construction of absent signals is based on the complement of the *cannot* sets, called *can* sets. Our equivalent

$$\begin{aligned}
\mathit{must}(0,E) &:= \emptyset \\
\mathit{must}(!s,E) &:= \{s\} \\
\mathit{must}(s^+?(P),E) &:= \begin{cases} \mathit{must}(P,E) & \text{if } s^+ \in E \\ \emptyset & \text{otherwise} \end{cases} \\
\mathit{must}(s^-?(P),E) &:= \begin{cases} \mathit{must}(P,E) & \text{if } s^- \in E \\ \emptyset & \text{otherwise} \end{cases} \\
\mathit{must}(P_1 \mid P_2,E) &:= \mathit{must}(P_1,E) \cup \mathit{must}(P_2,E) \\
\mathit{must}(P_1 ; P_2,E) &:= \begin{cases} \mathit{must}(P_1 \mid P_2,E) & \text{if } t(P_1,E) \\ \mathit{must}(P_1,E) & \text{otherwise} \end{cases} \\
\mathit{must}(P \backslash s,E) &:= \\
&\begin{cases} \mathit{must}(P, E \cup \{s^+\}) \backslash s & \text{if } s \in \mathit{must}(P, E \backslash s) \\ \mathit{must}(P, E \cup \{s^-\}) \backslash s & \text{if } s \in \mathit{cannot}^+(P, E \backslash s) \\ \mathit{must}(P, E \backslash s) \backslash s & \text{otherwise} \end{cases}
\end{aligned}$$

Fig. 3. ESTEREL Must

formulation brings out an important structural invariant. Firstly, *cannot*(P,E) is the logical dual of *must*(P,E), obtained by interchanging $\cap$ for $\cup$, $\emptyset$ for $\mathcal{S}$, and $\{s\}$ for $\mathcal{S} \setminus \{s\}$. Secondly, *must* and *cannot* are exclusive: *must*(P,E) $\cap$ *cannot*(P,E) = $\emptyset$ for any event E. In general, however, *must*(P,E) $\cup$ *cannot*(P,E) $\neq \mathcal{S}$, so *must* and *cannot* are not necessarily complements. This is analogous to the situation in two–player games: we will show below that *must* (*cannot*) corresponds to the construction of winning (losing) positions for the starting player. Elements neither in *must*(P,E) nor in *cannot*(P,E) indicate draw positions.

4. Formalising Mazes

This section formalises our two–player maze games introduced in Section 2.

4.1 Mazes

Mazes are essentially finite graphs with two kinds of directed edges, namely *visible* and secret edges. For our purposes, it is convenient to formally represent these graphs as *systems of unfolding rules* $M := (x \Leftarrow m_x)_{x \in V}$ in a language

$$
\begin{array}{lcl}
cannot(0, E) & := & \mathcal{S} \\
cannot(!s, E) & := & \mathcal{S}\backslash\{s\} \\
cannot(s^+?(P), E) & := & \begin{cases} \mathcal{S} & \text{if } s^- \in E \\ cannot(P, E) & \text{otherwise} \end{cases} \\
cannot(s^-?(P), E) & := & \begin{cases} \mathcal{S} & \text{if } s^+ \in E \\ cannot(P, E) & \text{otherwise} \end{cases} \\
cannot(P_1 \,|\, P_2, E) & := & cannot(P_1, E) \cap cannot(P_2, E) \\
cannot(P_1 ; P_2, E) & := & cannot(P_1, E) \cap cannot(P_2, E) \\
cannot(P\backslash s, E) & := & \\
\multicolumn{3}{c}{\begin{cases} cannot(P, E \cup \{s^-\})\backslash s & \text{if } s \in cannot(P, E\backslash s) \\ cannot(P, E\backslash s)\backslash s & \text{otherwise} \end{cases}} \\
cannot^+(P\backslash s, E) & := & \\
\multicolumn{3}{c}{\begin{cases} cannot^+(P, E \cup \{s^+\})\backslash s & \text{if } s \in must(P, E\backslash s) \\ cannot^+(P, E \cup \{s^-\})\backslash s & \text{if } s \in cannot^+(P, E\backslash s) \\ cannot^+(P, E\backslash s)\backslash s & \text{otherwise} \end{cases}}
\end{array}
$$

Fig. 4 : ESTEREL Cannot.

$$
\begin{array}{lcl}
t(0, E) & := & true \\
t(!s, E) & := & true \\
t(s^+?(P), E) & := & \begin{cases} t(P, E) & \text{if } s^+ \in E \\ true & \text{if } s^- \in E \\ false & \text{otherwise} \end{cases} \\
t(s^-?(P), E) & := & \begin{cases} t(P, E) & \text{if } s^- \in E \\ true & \text{if } s^+ \in E \\ false & \text{otherwise} \end{cases} \\
t(P_1 \,|\, P_2, E) & := & t(P_1, E) \wedge t(P_2, E) \\
t(P_1 ; P_2, E) & := & t(P_1, E) \wedge t(P_2, E) \\
t(P\backslash s, E) & := & t(P, E\backslash s)
\end{array}
$$

Fig. 5 : Termination Predicate.

of mazes, for some finite set $\mathcal{V}$ of variables representing rooms and maze terms m_x. *Maze terms* are defined in a process–algebraic fashion, which provides us with sufficient structure for proving the paper's main results. The syntax of maze terms is given by the following BNF:

$$m \quad ::= \quad 0 \;\mid\; x \;\mid\; \iota.m \;\mid\; \tau.m \;\mid\; \sum_{i \in I} m_i \;\mid\; \mu x.m.$$

Intuitively, 0 represents a dungeon, $\iota.m$ ($\tau.m$) represents a room with a visible (secret) corridor to room m. Term $\Sigma_{i\in I} m_i$ represents a room that merges all the rooms m_i with $i \in I$ and we write $m_1 + m_2$ for $\Sigma_{i\in\{1,2\}}\, m_i$. Also if $x \Leftarrow m_x$ is the unfolding rule defining x then x corresponds to the term m_x and $\mu x.m_x$ is the least fixed–point solution for x. In the remainder, we let $\mathcal{M}$ stand for the set of all maze terms. Besides we write $m\{m'/x\}$ for the syntactic substitution operation that replaces all free occurrences of x by term m' in m.

For each room x in any given maze M we would like to determine whether it is a winning position (for the starting player). The game–theoretic semantics of maze M requires the introduction of a *labeled transition system* $\langle \mathcal{M}, \{\iota, \tau\}, \longrightarrow \rangle$, where $\mathcal{M}$ is the set of states (or *rooms*), $\{\iota, \tau\}$ is the alphabet with ι encoding a visible action and τ a *secret action*, and $\longrightarrow \subseteq \mathcal{M} \times \{\iota, \tau\} \times \mathcal{M}$ is the transition relation representing valid moves (or *corridors*) between rooms. The transition relation is defined by the rules in Fig. 6, where γ ranges over $\{\iota, \tau\}$.

$$\frac{}{\gamma.m \xrightarrow{\gamma} m} \qquad \frac{m_j \xrightarrow{\gamma} m_j'}{\sum_{i\in I} m_i \xrightarrow{\gamma} m_j'}\; j \in I$$

$$\frac{m \xrightarrow{\gamma} m'}{x \xrightarrow{\gamma} m'}\; x \Leftarrow m \qquad \frac{m\{\mu x.m/x\} \xrightarrow{\gamma} m'}{\mu x.m \xrightarrow{\gamma} m'}$$

Fig. 6 : Transition Relation of Mazes.

Essentially, this labeled transition system reflects the game graphs of Section. 2, with dungeons being traps that have no outgoing transition. In the following, we write $m \longrightarrow$ for $\exists m'\, \exists \gamma.\; m \xrightarrow{\gamma} m'$. Note that operators ".", "Σ","$\Leftarrow$" and "μ" correspond to the process–algebraic operators *prefix, choice, recursion* and *fixed-point*. As it may be expected a dungeon is the identity (neutral) element of the choice operator since 0 has no transitions and, moreover, $0 = \sum_{i\in\emptyset} m_i$.

Example 1 : *Let us specify the maze in Fig. 1 relative to the program's signals, also called* named rooms, i.e., $V = \{a, b, c, d, e, f, g, h\}$. *The other* rooms $\{t_0, t_1, \ldots, t_6\}$ *are referred to as* unnamed rooms *and are represented implicitly as subterms in the corresponding system of unfolding rules* $M_{ex} := (x \Leftarrow m_x)_{x\in V}$ with:

$$
\begin{array}{ll}
a \Leftarrow \iota.0 & e \Leftarrow \iota.(\tau.b + \iota.g) \\
b \Leftarrow \iota.\iota.d & f \Leftarrow \iota.(\tau.d + \iota.a) \\
c \Leftarrow \iota.(\iota.b + \tau.a) & g \Leftarrow \iota.(\tau.b + \iota.e) \\
d \Leftarrow 0 & h \Leftarrow \iota.(\iota.e + \tau.a)
\end{array}
$$

Observe that, for any $x \Leftarrow m_x$, *applying the operational rules to* m_x *results in the part of the graph starting from room x. Specifically, for* $f \Leftarrow m_f$ *and* $m_f = \iota.(\tau.d + \iota.a)$, *the first* ι *in the term corresponds to the visible corridor connecting f and the unnamed room* $t_2 = \tau.d + \iota.a$. *From* t_2, *either a secret corridor can be taken to dungeon d, or a visible corridor* ι *can be followed reaching room a.*

4.2 Game–Theoretic Semantics

We now turn our attention to the game–theoretic semantics of our two-player maze game. For convenience, we

will name the players simply A and B. We begin by defining the notions *dungeon*, *path*, and *turn*. Firstly, room m is a dungeon if $m \not\longrightarrow$. Secondly, a *path* π through a maze M is a sequence of transitions $(m_i \xrightarrow{\gamma_i} m_{i+1})_{0\le i<k}$, where $k \in \mathbb{N} \cup \{\omega\}$ is referred to as the *length* $|\pi|$ of π. We say that π is finite in case $k < \omega$; otherwise, π is infinite. A path π is *maximal* if it is either infinite, or if it is finite and $m_{|\pi|} \not\longrightarrow$. We abbreviate π's finite prefix $(m_i \xrightarrow{\gamma_i} m_{i+1})_{0\le i<j}$ of length $j \in \mathbb{N}$ by π^j. Finally, given a finite (prefix of a) path π and assuming player A always starts off the game, we can determine the player *turn*(π) whose turn it is in the final room $m_{|\pi|}$ as follows, where $\overline{A} := B$ and $\overline{B} := A$:

$$turn(\pi) := \begin{cases} turn(\pi') & \text{if } |\pi| > 0 \text{ and } \pi = \pi' \cdot \xrightarrow{\tau} \\ \overline{turn(\pi')} & \text{if } |\pi| > 0 \text{ and } \pi = \pi' \cdot \xrightarrow{\iota} \\ A & \text{otherwise, i.e., } |\pi| = 0\,. \end{cases}$$

A maze play is determined by the players' strategies. A *strategy* is a (partial) function $\alpha : \mathcal{M} \rightharpoonup \{\iota, \tau\} \times \mathcal{M}$ such that, for all $m \in \mathcal{M}$, if $\alpha(m) = (\alpha_1(m); \alpha_2(m))$ is defined then $m \xrightarrow{\alpha_1(m)} \alpha_2(m)$. Note that a strategy of a player does not depend on the opponent's strategy or on a play's history. Given strategies α and β for players A and B, respectively, the play $play_M(\alpha, \beta, m)$ in maze M starting in room m with player A is a maximal path $\pi = (m_i \xrightarrow{\gamma_i} m_{i+1})_{0\le i<k}$ in M such that $m_0 = m$, $m_{i+1} = \alpha_2(m_i)$ and $\gamma_i = \alpha_1(m_i)$ if $turn(\pi^i) = A$, or $m_{i+1} = \beta_2(m_i)$, $\gamma_i = \beta_1(m_i)$ if $turn(\pi^i) = B$.

The operational semantics of maze $M = (x \Leftarrow m_x)_{x\in V}$ considers, for each room x, whether player A or B has a winning strategy, or neither of them. Intuitively, A (B) has a winning strategy, if he or she is always able to drive player B (A) into a dungeon, no matter which strategy B (A) employs and always assuming that player A starts off the game. Formally, player A has a winning strategy for room x in M if $\exists\alpha\forall\beta.\ |play_M(\alpha,\beta,x)| < \omega$ and $B = turn(play_M(\alpha, \beta, x))$. Dually, player B has a winning strategy for room x in M if $\exists\beta\forall\alpha.\ |play_M(\alpha,\beta,x)| < \omega$ and $A = turn(play_M(\alpha, \beta, x))$. If the selected starting player A has a winning strategy for room x, then x is simply called a *winning position*. If player B has a winning strategy, then x is a *losing position*. If both players can always avoid dungeons, thus engaging in infinite plays, neither player wins and the play ends in a draw. Accordingly, a position that is neither a winning nor a losing position is referred to as a *draw position*.

Example 2. *Consider again maze M_{ex} of Fig. 1. Suppose that the strategy α of player A is chosen such that A moves the token, through a secret corridor whenever there is one. The strategy β of player B is that B prefers overall visible corridors whenever possible. For example, $\alpha(t_4) = a$, $\alpha(a) = t_0$, $\alpha(t_5) = b$, $\alpha(b) = t_1$, $\alpha(t_6) = b$, $\beta(e) = t_5$, $\beta(g) = t_6$ and $\beta(t_1) = d$. Thus, the play $play_{Mex}(\alpha, \beta, t_4)$ is the path $t_4 \xrightarrow{\tau} a \xrightarrow{\iota} t_0$, along which A wins. In contrast, if player A chooses e in the first turn, B would have selected the move to t_5, from where A moves to b and then to t_1, from where B moves to the dungeon d and player A loses. The play in that case would be $t_4 \xrightarrow{\iota} e \xrightarrow{\iota} t_5 \xrightarrow{\tau} b \xrightarrow{\iota} t_1 \xrightarrow{\iota} d$. The best player A can do under this circumstances is to change his preferences for secret corridors and thus set $\alpha(t_5) = g$ and $\alpha(t_6) = e$. In that case the play amounts to a draw along the infinite path $h \xrightarrow{\iota} e \xrightarrow{\iota} t_5 \xrightarrow{\tau} g \xrightarrow{\iota} t_6 \xrightarrow{\iota} e \xrightarrow{\tau} t_5 \xrightarrow{\iota} g \ldots$.*

4.3 Strategies and Esterel

Technically, strategies within a maze M correspond to the *must*– and *cannot*–analysis for the associated program P, which is at the heart of ESTEREL's behavioral semantics. A simple way to make the connection is to read each *visible* (*secret*) corridor as a *present–else statement* (*present–then statement*). We then get an exact correlation between ESTEREL's declarative computation of *must*– and *cannot*–sets of signals [17] and an inductive computation of winning and losing positions in the game graph. We illustrate this correspondence using the sub–maze M_1 in our example of Fig. 1. The program P_1 associated with M_1 is:

$t_2^-?(!f) \,|\, d^+?(!t_2) \,|\, a^-?(!t_2) \,|\, t_0^-?(!a)$

We reason along the fixed–point computation of P_1's declarative semantics and start with the empty environment in which no signal is known to be present or absent, thus, $must^0 = cannot^0 = \emptyset$. Since no emission is unguarded, the first iteration yields no present signals, *i.e.*, $must^1 = \emptyset$; but since there are no emit statements for d or t_0, we get $cannot^1 = \{d, t_0\}$ immediately. In game terms this corresponds to identifying both rooms d and t_0 in M_1 as positions in which A loses right away. The fact that a is connected to t_0 by a visible corridor means that A now has a strategy to win a, because A can move into t_0 where B loses. In the computation of ESTEREL's declarative semantics for P_1, this is the second iteration step: since t_0 is known as absent, the emit statement in $t_0^-?(!a)$ is executed and a becomes present. We thus get $must^2 = \{a\}$ and $cannot^2 = \{d, t_0\}$. Additionally, we know that the statement $d^+?(!t_2)$ is not executed in the current instant. In the game, this amounts for marking the secret corridor from t_2 to d as useless for any winning strategy for t_2. There is no point for any player in going across to d since the player keeps the turn and thus loses in d. It may still be possible to win by moving from t_2 to a. However, with the extra information just obtained, namely that a is a winning position, we conclude that t_2 is in fact a losing position. For moving from t_2 to a does not help either since the opponent would get the turn in a and win. In the ESTEREL approximation sequence, t_2 indeed enters the *cannot* set in the third iteration step: $cannot^3 = \{d, t_0, t_2\}$. This is clear since $a \in must^2$ and $d \in cannot^2$. Hence the only two statements that could emit t_2 in P_1 are both switched off. The *must* set does not change, so $must^3 = must^2 = \{a\}$. The fourth iteration step identifies f as emitted from the fact that $t_2 \in cannot^3$. For this means, the statement $t_2^-?(!f)$ is executed. In game terms, room f is clearly a winning position as t_2 is a losing position. Hence, we obtain $must^4 = \{a, f\}$ and $cannot^4 = cannot^3 = \{d, t_0, t_2\}$ as the fixed point of ESTEREL's constructive analysis for P_1. To sum up, we see that $must^{n+1}$ ($cannot^{n+1}$) is the set of rooms that can be won (must be lost) by the starting player in at most n moves.

The example in Fig. 1 also illustrates how constructiveness of ESTEREL reactions is reflected in the game model. We have seen above that the submaze M_2 of M_{ex} contains only draw positions. The associated ESTEREL program P_2 can be

written as a parallel composition of four present statements, as suggested above, or equivalently in a more compact form as:

$$g^{+}?(!e) \mid e^{+}?(!g)$$

where the two rooms t_5 and t_6 are no longer represented as signals but are implicit in the nested present statements. As an aside, we will see below that our intermediate states t_i in game graphs are necessary to express conjunctive behavior. The *must*– and *cannot*–analysis for P_2 indeed leaves signals e and g constructively undecided. To justify emission of either e or g, both e and g would have to be present in the first place to activate the outmost present statements, which is causally unreasonable. We cannot justify the absence of e and g, causally, either. For example, in order to deactivate the inner emits through one of the outer guarding present conditions we would need that one of e and g is absent, which is causally problematic. Overall, there is only one logically coherent solution, namely that both e and g are absent, yet this solution is not causal. Since this is the only logically coherent solution, the *logical behavioral semantics* of ESTEREL would return this as the response, whereas the constructive semantics rejects it. In our maze game, a logically coherent solution amounts to a "speculative" assignment of 0 (losing) and 1 (winning) markings to rooms so that (*i*) a room is marked 1 exactly if one of its successor rooms that is accessible via a visible corridor is marked 0, or if a successor room connected via a secret corridor is marked 1; and (*ii*) a room is marked 0 if all rooms connected via visible corridors are marked 1 and if all rooms connected through a secret corridor are marked 0. In Fig. 1, $e = g = 0$ and $t_5 = t_6 = 1$ is the only logically coherent marking for M_2. Although this might suggest that both e and g are losing positions for the starting player, it is clear that this cannot be realized by any finite strategy of the opponent.

5. Programs and Mazes

This section first formally presents a translation of combinational ESTEREL programs P into mazes M with the property that winning (losing) positions in M correspond exactly to signals that must (cannot) be emitted in P. Also this section provides an alternative denotational characterisation of the operational notion of winning, losing and draw positions.

5.1 Representing Programs as Mazes

With each ESTEREL program P, we associate a maze $\langle\!\langle P \rangle\!\rangle := (a \Leftarrow \langle\!\langle P \rangle\!\rangle_a\{0/\delta\})_{a \in \mathcal{S} \cup \{\lambda\}}$ where the elements in the set $\mathcal{S} \cup \{\lambda, \delta\}$ play the role of term variables representing distinguished rooms. Note that in $\langle\!\langle P \rangle\!\rangle$ there is no rule of the form $\delta \Leftarrow \langle\!\langle P \rangle\!\rangle_\delta$ for the *connecting variable*. In general, term $\langle\!\langle P \rangle\!\rangle_a$ with $a \in \mathcal{S} \cup \{\lambda\}$ describes the game conforming to P that can be played starting in room a modulo some conditions (dependencies) yet to be defined where the instances of δ (if any) appear. Let us call a term *open* when it has occurrences of δ, otherwise it is *closed*. The game description of an open term $\langle\!\langle P \rangle\!\rangle_a$ can still be extended from some *connecting* rooms (*i.e.*, the δ's) to other rooms. This is done by a syntactic substitution that replaces all the instances of δ by a fresh term m. If this m is open then the substitution can continue in the same fashion until δ gets replaced by a closed term. Observe that each term of $\langle\!\langle P \rangle\!\rangle_a\{0/\delta\}$ from $\langle\!\langle P \rangle\!\rangle$ is closed. Henceforth, connecting rooms are schematically represented as dashed nodes.

Example 3. *Let* $\langle\!\langle P \rangle\!\rangle_x$ *be the open term* $\iota.\delta$ *and let us compute:*

$$\langle\!\langle P \rangle\!\rangle_x\{\delta + \tau.b/\delta\}\{\delta + \iota.a/\delta\}\{0/\delta\}$$

step by step as in Fig.7. From the resulting closed term $\iota.(\tau.b + \iota.a)$*, it follows that room x is a winning position iff room a is winning too and room b is losing. From this configuration (*i.e.*, a is winning and b is losing), two observations can be made: (i) room x becomes a losing position if the status of either a or b is switched from winning (losing) to losing (winning) and (ii) room x becomes a draw when a or b are set to draw. It is worth observing that the obtained closed term corresponds to* $\langle\!\langle a^{+}?(b^{-}?(!x)) \rangle\!\rangle_x\{0/\delta\}$.

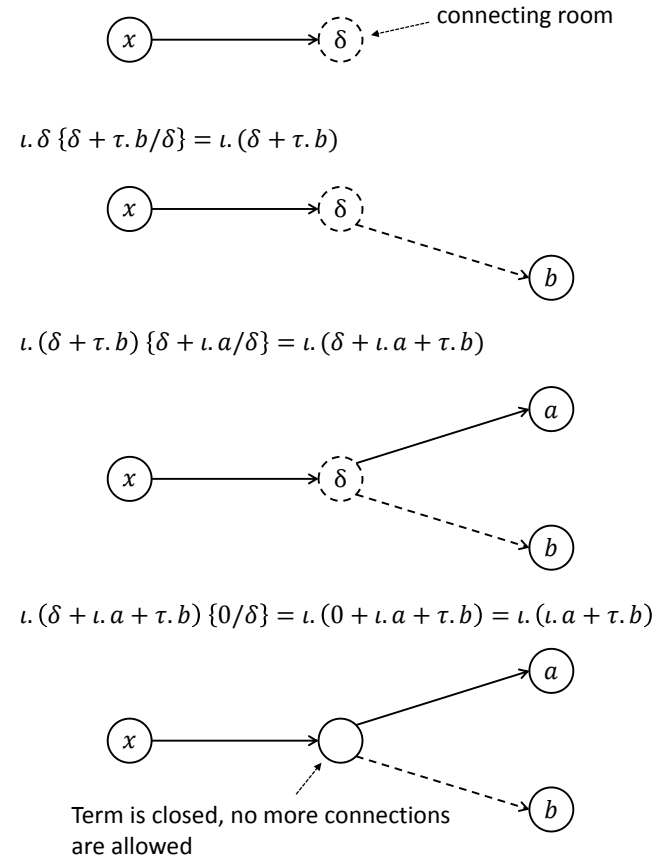


Fig. 7 : Connecting Rooms

For any given program P, the *termination variable* λ in $\langle\!\langle P \rangle\!\rangle_\lambda$ corresponds to a room whose winning (draw) status indicates whether P terminates (or not). By construction, room λ can never be a losing position. Formally, $\langle\!\langle P \rangle\!\rangle_a$ for $a \in \mathcal{S}$ and $\langle\!\langle P \rangle\!\rangle_\lambda$ are defined along the structure of P in Fig. 8 and 9 respectively.

$$
\begin{aligned}
\langle\!\langle 0 \rangle\!\rangle_a &:= 0 \\
\langle\!\langle !s \rangle\!\rangle_a &:= \begin{cases} \iota.\delta & \text{if } s = a \\ 0 & \text{otherwise} \end{cases} \\
\langle\!\langle s^+?(P) \rangle\!\rangle_a &:= \langle\!\langle P \rangle\!\rangle_a \{\delta + \iota.s/\delta\} \\
\langle\!\langle s^-?(P) \rangle\!\rangle_a &:= \langle\!\langle P \rangle\!\rangle_a \{\delta + \tau.s/\delta\} \\
\langle\!\langle P_1 \,|\, P_2 \rangle\!\rangle_a &:= \tau.\langle\!\langle P_1 \rangle\!\rangle_a + \tau.\langle\!\langle P_2 \rangle\!\rangle_a \\
\langle\!\langle P_1 \,;\, P_2 \rangle\!\rangle_a &:= \tau.\langle\!\langle P_1 \rangle\!\rangle_a + \iota.(\iota.\langle\!\langle P_1 \rangle\!\rangle_* + \iota.\langle\!\langle P_2 \rangle\!\rangle_a) \\
\langle\!\langle P \backslash s \rangle\!\rangle_a &:= \begin{cases} 0 & \text{if } s = a \\ \langle\!\langle P \rangle\!\rangle_a \{\mu s.\langle\!\langle P \rangle\!\rangle_s / s\} & \text{otherwise} \end{cases}
\end{aligned}
$$

Fig. 8 : Maze terms for a signal a 2 S

$$
\begin{aligned}
\langle\!\langle 0 \rangle\!\rangle_\lambda &:= \iota.\delta + \tau.\delta \\
\langle\!\langle !s \rangle\!\rangle_\lambda &:= \iota.\delta + \tau.\delta \\
\langle\!\langle s^+?(P) \rangle\!\rangle_\lambda &:= \langle\!\langle P \rangle\!\rangle_\lambda \{\delta + \iota.(\delta + \tau.s)/\delta\} \\
\langle\!\langle s^-?(P) \rangle\!\rangle_\lambda &:= \langle\!\langle P \rangle\!\rangle_\lambda \{\delta + \iota.(\delta + \iota.s)/\delta\} \\
\langle\!\langle P_1 \,|\, P_2 \rangle\!\rangle_\lambda &:= \iota(\iota.\langle\!\langle P_1 \rangle\!\rangle_\lambda + \iota.\langle\!\langle P_2 \rangle\!\rangle_\lambda) \\
\langle\!\langle P_1 \,;\, P_2 \rangle\!\rangle_\lambda &:= \iota(\iota.\langle\!\langle P_1 \rangle\!\rangle_\lambda + \iota.\langle\!\langle P_2 \rangle\!\rangle_\lambda) \\
\langle\!\langle P \backslash s \rangle\!\rangle_\lambda &:= \langle\!\langle P \rangle\!\rangle_\lambda \{\mu s.\langle\!\langle P \rangle\!\rangle_s / s\}
\end{aligned}
$$

Fig. 9 : Maze terms for termination

As an example, the translation $\langle\!\langle P \rangle\!\rangle$ of the ESTEREL program

$$!a \,|\, d^+?(!b) \,|\, a^+?(d^-?(!f)) \,|\, a^-?(b^+?(!c) \,|\, e^+?(!h))$$
$$b^-?(g^+?(!e) \,|\, e^+?(!g))$$

generates the unfolding rules of Example 1 and thus the maze in Fig. 1. Note that these have been slightly optimised using the equivalence $m \equiv \tau.m + \Sigma\,\tau.0$, as well as not including the unfolding rule $\lambda \Leftarrow \langle\!\langle P \rangle\!\rangle_\lambda \{0/\delta\}$ in the representations.

Nothing and Emission

Intuitively, program 0 that cannot emit any signal, must correspond to a dungeon in which player A loses right from the start. In the maze $\langle\!\langle 0 \rangle\!\rangle$ we have that $a \Leftarrow 0$ for all $a \in \mathcal{S}$, so every room a is a losing position which corresponds to the *cannot* set for 0.

The forced emission of a signal a leads to immediately success for A in room a and loss in any other room. The reason is that in the behavioural semantics of $!a$, signal a is in the *must* set and all the other signals are in the *cannot* set. The maze $\langle\!\langle !a \rangle\!\rangle$ reflects this by letting $a \Leftarrow \iota.\delta\{0/\delta\} = \iota.0$ and $b \Leftarrow 0\{0/\delta\} = 0$ for any $b \neq a$ with $a, b \in \mathcal{S}$.

For both programs 0 and $!a$, room $\lambda \Leftarrow \iota.\delta + \tau.0\ \{0/\delta\} = \iota.0 + \tau.0$ is a winning position which indicates termination. In general, the status of λ (from any $\langle\!\langle P \rangle\!\rangle_\lambda$) can only be winning or draw. For $\langle\!\langle 0 \rangle\!\rangle_\lambda$ and $\langle\!\langle !a \rangle\!\rangle_\lambda$, we can see that any substitution of δ in $\iota.\delta + \tau.0$ by a term corresponding to a winning or losing position would give winning status to λ. Substituting δ in $\iota.\delta + \tau.0$ by a term resulting in a draw would make λ also a draw and this will indicate that the program does not terminate.

Signal Test and Choice

Room $a \in \mathcal{S}$ is a winning position in $\langle\!\langle s^+?(P) \rangle\!\rangle_a$ if player A has winning strategy for both room s in $\langle\!\langle s^+?(P) \rangle\!\rangle_a$ and room a in $\langle\!\langle P \rangle\!\rangle_a$. The latter condition implies that there is at least one $!a$ that is executed inside the structure of P. From Fig. 8, any such $\langle\!\langle !a \rangle\!\rangle_a$ give us $\iota.\delta$ and this δ can be replaced just in two ways, namely $\{\delta + \iota.s/\delta\}$ and $\{\delta + \tau.s/\delta\}$ for some $s \in \mathcal{S}$. Thus any emission of a occurring in P will turn in $\langle\!\langle P \rangle\!\rangle_a$ into a room, say $\breve{a}$, of the form $\iota.\ (\delta + \sum_{r \in I} \gamma.r)$ with $\gamma \in \{\iota, \tau\}$ and $\varnothing \subseteq I \subseteq \mathcal{S}$. The case when $I = \varnothing$ refers to the term $\iota.\delta$ on which no substitution of δ has been performed yet. From any such $\breve{a}$, player A (using the visible corridor ι) can pass the turn to B in a room, say $\breve{x}$, representing $\sum_{r \in I} \gamma.r$. In $\langle\!\langle P \rangle\!\rangle_a\{0/\delta\}$, any such $\breve{a}$ ($\breve{x}$) becomes a room, say $\breve{a}_0$ ($\breve{x}_0$), of the shape $\tau.\sum_{r \in I} \gamma.r$ ($\sum_{r \in I} \gamma.r$). Moreover, in $\langle\!\langle s^+?(P) \rangle\!\rangle_a\{0/\delta\}$, any $\breve{a}$ turns into a room $\breve{a}_1$ of the form $\tau.\sum_{r \in I} \gamma.r + \iota.s = \iota.(\breve{x}_0 + \iota.s)$ and any $\breve{x}$ becomes room $\breve{x}_1$ of the shape $\breve{x}_0 + \tau.s$. So player A in $\breve{a}_1$ first gives the control to B in $\breve{x}_1$, who in turn freely decides whether A must continue with his play in s or in $\breve{x}_0$. Therefore, $\breve{a}_1$ is a winning position when room s is winning and $\breve{x}_0$ is losing. However, that $\breve{x}_0$ is losing immediately implies that $\breve{a}_0$ is winning in the maze corresponding to P. This encoding of conjunction into mazes is justified by the observation that signal a is in the *must* set of $s^+?(P)$ when s is known to be present *and* a is also in the *must* set for P. Analogously, a program $s^-?(P)$ that is negatively guarded by signal s must emit a if, in room a, player B has a winning strategy for s in $\langle\!\langle a^+?(P) \rangle\!\rangle_a$, and if player A has a winning strategy for a in $\langle\!\langle P \rangle\!\rangle_a$.

In the computation of terms for termination of Fig. 9, there is just one form on which the connecting variable δ could be introduced (*i.e.*, $\iota.\delta + \tau.\delta$) and two ways on which δ can be replaced. These substitutions correspond precisely to the positive and negative tests of signal status, namely $\langle\!\langle s^+?(P) \rangle\!\rangle_\lambda$ and $\langle\!\langle s^-?(P) \rangle\!\rangle_\lambda$.

In order to illustrate the mechanics described in Fig. 9, consider a program P_{cond} of the form $s_0^+?(s_1^-?(\ldots s_{n-1}^-?(s_n^+?(!x))))$ and let $S_{pos} = \{s_0, \ldots, s_n\}$ and $S_{neg} = \{s_1, \ldots, s_{n-1}\}$ be the sets of all testing signals positively or negatively guarded in P_{cond} respectively. Then obtain $\langle\!\langle P_{cond} \rangle\!\rangle_\lambda\{0/\delta\}$ as in Fig. 10, where an acyclic submaze is identified as D. The set of deciding rooms in D is $Z = \{z_i = \tau.s_i \mid s_i \in S_{posg}\} \cup \{z_i = \iota.s_i \mid s_i \in S_{neg}\}$. The set of corridors (all ι) among the rooms in Z are the deciding corridors that belong

to submaze *D*.

For explaining the purpose of the deciding rooms, let us assume, for the moment, that the deciding corridors are not in *D*. Thus, $z_i = \tau.s_i$ is winning (losing) when s_i is winning (losing) and positively guarded. Conversely, $z_i = \iota.s_i$ is winning (losing) when s_i is losing (winning) and negatively guarded. Otherwise, $z_i = \gamma.s_i$ with $\gamma \in \{\iota, \tau\}$ is draw when s_i is a draw independently of the guard. That is to say the status of a deciding room z_i indicates whether the test on signal s_i succeeds. This encoding establishes the basic elements required by the termination predicate from Fig. 5. In other words, a program of the form $s^+?(P)$ must terminate when *s* is known to be absent (test fails) *or* if *s* is present (test succeeds) and *P* terminates. Otherwise (*i.e.*, the status of *s* is not decided), the program does not terminate (a draw). The case for $s^-?(P)$ is symmetric.

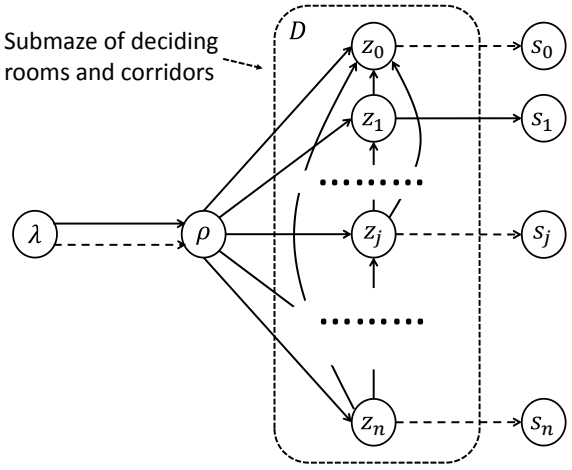


Fig. 10 : Termination for nested tests of signals

The role of the deciding corridors has to do with determining termination from a sequence of nested conditionals like the one in program P_{cond}. Deciding corridors can be structured around the sets $Z_i^< = \{z_k \in Z \mid k < i\}$ and $Z_i^> = \{z_k \in Z \mid k > i\}$ relative to some arbitrary deciding room z_i. Thus, from any $z_i \in Z$ there is a deciding corridor to every room in $Z_i^<$. As a consequence, there is a deciding corridor from every room in $Z_i^>$ to z_i. If all deciding rooms are winning, this means that all positive and negative signal tests have succeeded. Moreover, in this case, the deciding corridors do not affect in any manner the status of any deciding room. For example, take $z_j = \tau.s_j$ with $s_j \in S_{pos}$ [2] where s_j must be winning since we assume that all tests should be successful. But then z_j is winning (*i.e.*, the corresponding player can move to s_i and win from there) independently of the fact that every $z_i \in Z_i^<$ is also a winning position. So, when every deciding room is winning, the room identified as ρ in Fig. 10 is a losing position which makes λ winning position. This indicates that program P_{cond} terminates when every $s_i \in S_{pos}$ is present and every $s_i \in S_{neg}$ is absent.

For a deciding room to be a losing position, say $z_j = \tau.s_j$ with $s_j \in S_{pos}$, s_j has to be losing and every $z_i \in Z_i^<$ must be winning. If so, every room $z_i \in Z_i^>$ becomes automatically winning. This means that if there is a deciding room which is losing, then all the other deciding rooms are winning. Observe that this losing (deciding) room z_j will correspond to the test failure of s_j and this will be the first to occur in the sequence of tests $s_0^+?(s_1^-?(\ldots))$ of the program P_{cond}. Also the losing status of z_j will make ρ a winning position which, in turn, implies that λ is a winning room in this case. This says that P_{cond} terminates as soon as a signal test fails independently of the other tests that have not yet been performed which exactly corresponds to what the termination predicate defines in Fig. 5.

From the structure of *D*, if a deciding room z_i is a draw then it has to be the case that any room $z_i \in Z_i^<$ is either winning or draw. Otherwise (*i.e.*, $z_i \in Z_i^<$ is losing), z_j will be a winning position. A room $z_i \in Z_i^>$ cannot be a losing position because from this position a player can always take the deciding corridor to the draw position z_j and avoid losing from z_i. Hence each room $z_i \in Z_i^>$ is a winning or a draw position. Therefore, if one of the deciding rooms is a draw then every other deciding room is a winning position or a draw. If there are several draws among the deciding rooms, take the first draw $z_j = \tau.s_j$ in the sequence $z_0, z_1, \ldots$ ordered according to the succession of tests $s_+^0?(s_1^-?(\ldots))$ of P_{cond}. So, all the rooms $z_i \in Z_i^<$ have to be winning, otherwise either z_j will not be the first draw occurring in the sequence or z_j will not be a draw at all. Because z_j is a draw and all its deciding successors are winning position then s_j has to be a draw. This means that it is not known whether the status of s_j is present or absent. Therefore, P_{cond} does not terminate as it is suggested by the termination predicate of Fig. 9. In the maze of Fig. 10, if all deciding rooms are winning positions or draws then both ρ and λ are draws. As we have already mentioned, the condition of no termination for a program is precisely witnessed when λ is a draw.

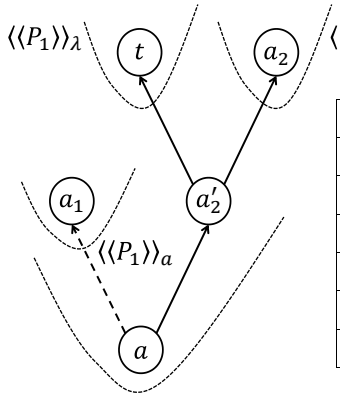
Sequential and Parallel Composition.

In a parallel composition $P_1 \mid P_2$, an emission of a signal *a* may occur in either component P_1 or P_2. Thus, this is encoded in $\langle\!\langle P_1 \mid P_2 \rangle\!\rangle_a$ by a free choice for the leading player. Specifically, the term $\tau.\langle\!\langle P_1 \rangle\!\rangle_a + \tau.\langle\!\langle P_2 \rangle\!\rangle_a$ of *a* for the parallel composition corresponds to a room which will be winning if the room representing *a* of either P_1 or P_2 is winning. This models the *must* computation for the parallel composition (Fig. 3) which basically establishes that a signal is present if it is present in either component. The same room *a* of the parallel composition is losing when both rooms representing *a* of P_1 and P_2 are losing. This matches the *cannot* analysis of the parallel composition (Fig. 4) that says that a signal cannot

[2] Henceforth, we are going to take an arbitrary deciding room $z_j = \tau.s_j$ with $s_j \in S_{pos}$. That is the deciding room corresponding to a signal that is positively guarded in the program. The cases for $z_k = \iota.s_k$ with $s_k \in S_{neg}$ are symmetrical.

be emitted by the parallel composition when it cannot be emitted by both components. The case of a draw for the room a of this construct occurs when the room representing a of either P_1 or P_2 is losing and the other is draw *or* when both are draws. This reflects the fact that a signal is neither present or absent when it is not emitted by any of the components *and* it cannot be emitted by at most one of the components.

The game of a sequential composition $\langle\!\langle P_1 \,;\, P_2 \rangle\!\rangle_a$ starting from room a will have the structure presented in Fig.11. Here, a_1 (a_2) indicates the starting room of $\langle\!\langle P_1 \rangle\!\rangle_a$ ($\langle\!\langle P_2 \rangle\!\rangle_a$), room t corresponds to the termination of P_1 and a_2' is an intermediate representation that acts as a witness for a_2 with respect to t. The idea is that the path between a and a_2 acts like a τ corridor as long as P_1 terminates *or* room a_2 is not a winning position. For verifying this assume that the play always goes from a to a_2', *i.e.*, forget about the τ corridor from a to a_1 for the moment. First, fix t to be winning: (*i*) if a_2 is **winning** (*losing*) then a_2' is losing (winning) and a must be **winning** (*losing*) and (*ii*) when a_2 is a draw then a_2' and a_2 are also **draws**. Second, let t be a draw: if a_2 is **losing** (*draw*) then witness a_2' is winning (*draw*) and, therefore, a is losing (*draw*). If we take into account the corridor from a to a_1 and since the path between a and a_2 can be interpreted as a corridor, we will have the same structure as the one of the parallel composition. This basically implies that a given signal a is in the *must* (*cannot*) set of both $P_1 \,;\, P_2$ and $P_1 \mid P_2$ when (*i*) P_1 terminates or (*ii*) signal a is **not** in the *must* set of P_2 which is another way of expressing the analysis of Fig. 3 and 4 for the cases covered so far. On the other hand, if t is a draw and a_2 is a winning position, this will make room a_2' a draw. Then room a can only be a winning position if a_1 is a winning position too. Otherwise, a becomes a draw. This says that the *must* set of $P_1 \,;\, P_2$ is the *must* set of P_1 when P_1 does not terminate which exactly corresponds to the definition of Fig. 3.



a_1	t	a_2	a
W	χ	χ	W
L	W	χ	χ
L	D	W	D
L	D	L	L
L	D	D	D
D	D	χ	D

$$\langle\langle P_1; P_2\rangle\rangle_a \;\coloneqq\; \tau.\langle\langle P_1\rangle\rangle_a + \iota.(\iota.\langle\langle P_1\rangle\rangle_\lambda + \iota.\langle\langle P_2\rangle\rangle_a)$$

Fig. 11 : Sequential Composition

The table in Fig. 11 summarises all possible combinations of statuses for the rooms in the maze. Winning, losing and draw appear there as W, L and D respectively. Variable χ corresponds to any of these three statuses. Notice that the table does not take into account the cases when a_1 is a draw and t is winning. The reason is that program P_1 cannot terminate when a signal in there is neither present or absent. Hence if a_1 is a draw then t must be also a draw which is reflected in the last row of the table.

Termination for parallel and sequential compositions is straightforward. For both constructs, λ gets defined by $\iota.(\iota.\langle\!\langle P_1 \rangle\!\rangle_\lambda + \iota.\langle\!\langle P_2 \rangle\!\rangle_\lambda)$. This means that the player A can give the turn to player B in $\iota.\langle\!\langle P_1 \rangle\!\rangle_\lambda + \iota.\langle\!\langle P_2 \rangle\!\rangle_\lambda$ who can choose to make A play from $\langle\!\langle P_1 \rangle\!\rangle_\lambda$ or $\langle\!\langle P_2 \rangle\!\rangle_\lambda$. In this form, λ could only be a winning position if the starting rooms of $\langle\!\langle P_1 \rangle\!\rangle_\lambda$ and $\langle\!\langle P_2 \rangle\!\rangle_\lambda$ are winning position. This codification specifies that parallel (sequential) composition terminates when both of its component terminate.

Local Signals.

A signal declaration $P\backslash s$ introduces a locally defined signal s that is available for broadcast inside program P only. As an example consider the program $P := (P_1 \setminus s) \mid P_2$ where $P_1 := s^+?(!a) \mid !s$ and $P_2 := s^+?(!b)$. In P_2, signal s refers to a different incarnation than the one used inside $P_1\backslash s$, whose scope is restricted by the local signal declaration. Consequently, the internal emission of s in $P_1\backslash$s will trigger the emission of signal a in P_1 but not that of signal b in P_2.

The simplest way of interpreting local signal declarations in terms of mazes is to rename signal s in $\langle\!\langle P \rangle\!\rangle$ into some fresh signal name s', and to make sure that s' is never used outside of $\langle\!\langle P\backslash s \rangle\!\rangle$. While this "naming–apart" technique is a simple solution for a compiler, the technique employed in the paper is algebraically more satisfactory. Intuitively, $\langle\!\langle P\backslash s \rangle\!\rangle$ is the same as solving the recursive unfoldings characterising maze $\langle\!\langle P \rangle\!\rangle$ with respect to signal s. If $s \Leftarrow m_s$ is the unfolding rule defining s, the least fixed–point solution for s is $\mu s.m_s$. This recursive term for the game starting in room s is then used, or substituted, wherever s is referenced in the unfoldings defined by $\langle\!\langle P \rangle\!\rangle$ for all the other rooms different from s. Thus, s gets eliminated. In addition, the unfolding for s in $\langle\!\langle P \rangle\!\rangle$ is "reset" to 0.

Example 4. *Consider again the programs P_1 and P_2 from our previous discussion. Fig. 12 and 13 present, respectively, the maze corresponding to $\langle\!\langle P_1 \mid P_2 \rangle\!\rangle$ and $\langle\!\langle (P_1\backslash s) \mid P_2 \rangle\!\rangle$ without considering termination. Nevertheless, the exclusion of λ in these mazes does not affect the status obtained for the rooms since the programs are composed in parallel. Also, in these mazes, for any signal, say a, room a_1 (a_2) is the initial room of $\langle\!\langle P_1 \rangle\!\rangle_a$ ($\langle\!\langle P_2 \rangle\!\rangle_a$). As it can be seen in Fig. 12, the emission of s in the program makes the three rooms a, b and s winning positions. However, in the maze of Fig. 13, the emission in P_1 of (local) s is completely encapsulated, so this uniquely affects the part of the maze corresponding to P_1. Thus, room a becomes a winning position. The signal name s remaining in the maze now refers to a fresh signal that is per default not emitted and hence a losing position which makes room b a losing position too.*

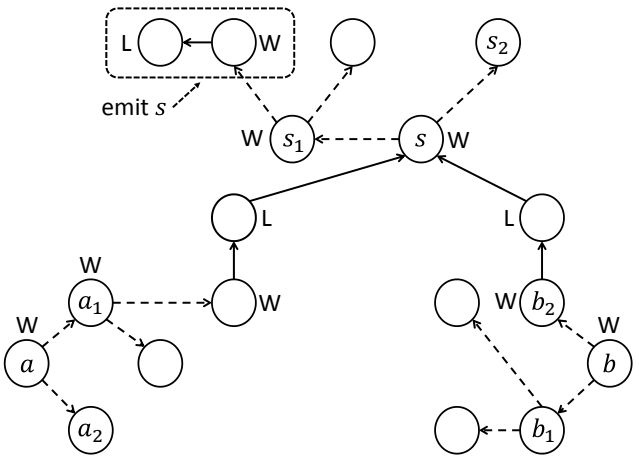


Fig. 12 : Maze $\langle\!\langle P_1 \mid P_2 \rangle\!\rangle$

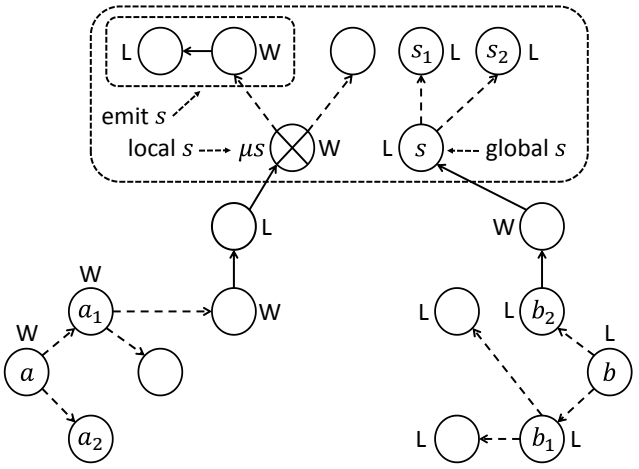


Fig. 12 : Maze $\langle\!\langle P_1 \mid P_2 \rangle\!\rangle$

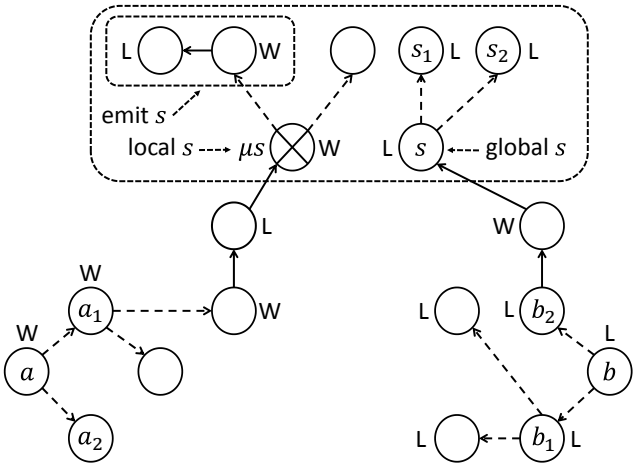


Fig. 13 : Maze $\langle\!\langle (P_1 \setminus s) \mid P_2 \rangle\!\rangle$

Note that treating local signal declarations via fixed–point operators has required a slightly more general translation of $s^+?(P)$ and $s^-?(P)$ than the one given in [39]. This observation has already been made by Berry in [17]. Consider, *e.g.*, a positive guard $c+?(P\setminus s)$ which makes winning inside $P\setminus s$ dependent on winning signal c. This implies that any room in the maze of $P\setminus s$ can only be won under the extra condition that c can be won as well. This applies to all rooms inside $P\setminus s$, even to the "local" room s. To give the opponent a chance to force the starting player into room c at any point where an emission occurs inside $P\setminus s$, we extend the maze inside $P\setminus$s by a corridor (ι in this case) from all the connecting rooms δ to c. In such rooms the opponent can choose to challenge the starting player into room c or accept to continue in $P\setminus$s.

Example 5. *Consider the program* $c^+?(P\setminus s)$ *where* $P := !s \mid s^-?(!c)$ *taken from [17]. Fig. 14(top) shows term* $\langle\!\langle P\setminus s \rangle\!\rangle_c$ *as a maze. As we can see the emission of (local) s is not guarded while the emission of c depends on (local) s to be absent. If we were to consider just* $P\setminus s$ *that would be alright. But in the context of* $c^+?(P\setminus s)$ *we cannot conclude that c is absent on the grounds that in* $P\setminus s$ *the (local) s is emitted and this avoids the emission of c. This is forbidden in ESTEREL since this argument executes speculatively the emission of (local) s. In our model,* $\langle\!\langle c^+?(P\setminus s) \rangle\!\rangle_c = \langle\!\langle P\setminus s \rangle\!\rangle_c \{\delta + \iota.c/\}$ *would be as in Fig. 14(bottom). There, the emission of (local) s depends on the status of c in the first place. This creates a cycle in the maze and all the rooms (except the dungeons) are draws.*

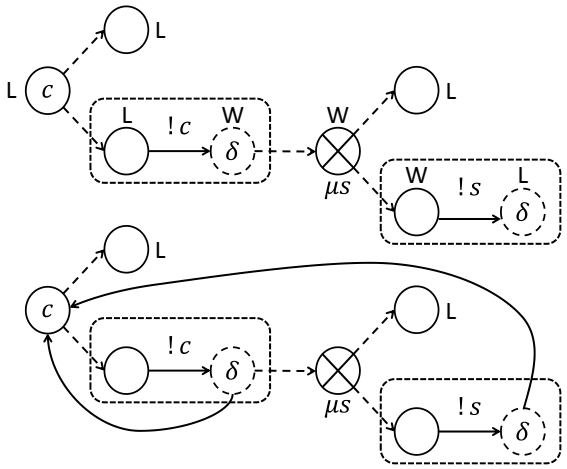


Fig. 14 : Local signal guarded by a test

For $P\setminus s$ the termination room λ corresponds to the initial room of $\langle\!\langle P \rangle\!\rangle_\lambda$ on which the (local) room s has been substituted via the fixed–point operator. This is the same as saying that $P\setminus s$ terminates as long as P terminates when all the references to s in P are considered local.

5.2 Denotational Characterisation of Mazes

The semantics of mazes in terms of winning and losing positions may also be captured denotationally. The denotational approach relies on two predicates *win* and *lose* that take as parameters a maze term m and an *environment* X.

Adapting notational conventions from ESTEREL, we define an environment X as a set of signed variables x^+ and x^-, with the meaning that plus–tagged (minus–tagged) variables represent rooms that are known to be winning (losing) positions. Since a single room cannot be both a winning and a losing position, an environment must not contain both x^+ and x^-, for any x. Predicate *win* (*lose*) now holds for m and X if m corresponds to a winning (losing) position relative to X, respectively. Formally, these predicates are defined as the least predicates respecting the following rules:

$$\begin{array}{lll} win(x, X) & \text{if} & x^+ \in X \\ win(\iota.m, X) & \text{if} & lose(m, X) \\ win(\tau.m, X) & \text{if} & win(m, X) \\ win(m_1 + m_2, X) & \text{if} & win(m_1, X) \text{ or } win(m_2, X) \end{array}$$

$$\begin{array}{lll} lose(0, X) & & \\ lose(x, X) & \text{if} & x^- \in X \\ lose(\iota.m, X) & \text{if} & win(m, X) \\ lose(\tau.m, X) & \text{if} & lose(m, X) \\ lose(m_1 + m_2, X) & \text{if} & lose(m_1, X) \text{ and } lose(m_2, X). \end{array}$$

Intuitively, the dungeon 0 is always a losing position. Room $m_1 + m_2$ is a winning position if at least one of m_1 or m_2 is, since $m_1 + m_2$ essentially gives a free choice to the leading player whether to continue with m_1 or m_2. Dually, $m_1 + m_2$ is a losing position if both m_1 and m_2 are. Room $\iota.m$ can only be left by the visible corridor ι to m, thereby giving control to the opponent. Hence, $\iota.m$ is a winning (losing) position for the leading player if m is a losing (winning) position for the opponent. Traversing a secret corridor does not change a player's turn, so $\tau.m$ is a winning (losing) position for the leading player if m is. Thus, by an appropriate choice of visible and secret corridors out of a room x we can make the winning (losing) predicate for x an arbitrary disjunctive (conjunctive) combination of negated and non-negated winning conditions of the immediate successor rooms. This is useful for normal-form representations and explains why we introduce secret corridors into the games model.

We now define a function on environments:

$$maze(M)(X) := win(M,X)^+ \cup lose(M,X)^-.$$

Here, $win(M,X)$ denotes

$$\{x \in \mathcal{V} \mid win(m_x, X), x \Leftarrow m_x \text{ in } M\}$$

and $lose(M,X)$ stands for

$$\{x \in \mathcal{V} \mid lose(m_x, X), x \Leftarrow m_x \text{ in } M\}$$

Additionally, for any subset $V \subseteq \mathcal{V}$ of variables, V^+ denotes the set $\{x^+ \mid x \in V\}$ and V^- the set $\{x^- \mid x \in V\}$. It can easily be proved that function $maze(M)$ is monotonic. Hence, the least fixed point $\mu maze(M)$ of $maze(M)$ exists, which is taken to be the denotational semantics of maze M. Moreover, because our universe of variables is finite, one may iteratively compute $\mu maze(M) = \cup_{i \in \mathbb{N}} maze(M)^i(\varnothing)$.

Example 6. *Let us obtain the sets of winning, losing, and draw positions for maze M_{ex} of Fig. 1. Initially, the predicates that hold are $lose(0,\varnothing)$ and $win(\iota.0, \varnothing)$. Since $a \Leftarrow \iota.0$ and $d \Leftarrow 0$ we get $maze(M_{ex})(\varnothing) = \{a^+, d^-\}$. In this environment, $f \in win(M_{ex}, \{a^+, d^-\})$ since $f \Leftarrow \iota.(\iota.a + \tau.d)$ and $lose(\iota.a + \tau.d, \{a^+, d^-\})$ the latter essentially results from $win(\iota.a, \{a^+, d^-\})$ and the fact that $lose(\tau.d, \{a^+, d^-\})$. Next we derive c, $h \in lose(M_{ex}, \{a^+, d^-\})$ from $c \Leftarrow \iota.(\tau.a + \iota.b)$ and $h \Leftarrow \iota.(\tau.a + \iota.e)$. Here $win(\tau.a, \{a^+, d^-\})$ implies that $win(\tau.a + \iota.b, \{a^+, d^-\})$ and $win(\tau.a + \iota.e, \{a^+, d^-\})$ both hold. Then we obtain $b \in lose(M_{ex}, \{a^+, d^-\})$ because $b \Leftarrow \iota.\iota.d$ and $win(\iota.d, \{a^+, d^-\})$ results from $lose(d, \{a^+, d^-\})$. This shows that $maze(M_{ex})^2(\varnothing) = \{a^+, f^+, b^-, c^-, d^-, h^-\}$.*

Another iteration confirms this as least fixed point, namely

$$\mu maze(M_{ex}) = \{a^+, f^+, b^-, c^-, d^-, h^-\}.$$

Theorem 1 (Coincidence). *Let M be a maze and x be a variable.*

- *x is a winning position in M if and only if $x^+ \in \mu maze(M)$.*
- *x is a losing position in M if and only if $x^- \in \mu maze(M)$.*

The proof of this theorem can be adapted from results on finite symmetric and memory-free games [28]. The only slight twist is that our setting allows for two types of transitions in game graphs, namely visible and secret ones.

Proposition 1. *Let P be a combinational ESTEREL program and E an event.*

1. $must(P, E) = \{a \in \mathcal{S} \mid win(\langle\!\langle P \rangle\!\rangle_a, E)\}$
2. $cannot(P;E) = \{a \in \mathcal{S} \mid lose(\langle\!\langle P \rangle\!\rangle_a, E)\}$

This proposition states the desired one-to-one relation between the *must*– and *cannot*–analysis in combinational ESTEREL programs and the determination of winning and losing positions in their corresponding mazes. Its proof can be conducted by induction on the structure of ESTEREL programs. As a consequence of the proposition, the functions *esterel*(*P*) and maze($\langle\!\langle P \rangle\!\rangle$) coincide. This immediately proves the following theorem.

Theorem 2 (Game–Theoretic Characterization). *For every combinational Esterel program P, we have $\mu esterel(P) = \mu maze(\langle\!\langle P \rangle\!\rangle)$.*

6. Related Work

As the paper shows, games provide a powerful and intuitively rather appealing setting for studying non-monotonic problems with co– and contravariant logical dependencies. Such problems abound in Computer Science. Many of these arise from the need to handle open systems and maintain a compositional system-environment distinction. If the interaction with the environment has both enabling as well as inhibiting effects on the response of a system then the input–output semantics of an individual component necessarily involves non–monotonic functions. When such systems are composed and each component acts both as a system and as (part of) the environment at the same time, causality cycles can occur that are not easy to resolve algebraically since for non–monotonic functions the standard least or greatest fixed point approach breaks down. Here, as Jaakko Hintikka has argued for logic and set theory [29], game theory – with its strong intensional notion of truth – can be used.

The games we are using are non-classical in the sense that they are not necessarily determined. The reader may find some background material on classical games in [28], [30].

The changes in our setting over the classical definitions are that we (*i*) allow for two types of moves in game graphs, *i.e.*, visible and secret transitions, and (*ii*) admit draw positions, *i.e.*, there may not exist a winning strategies for either player. The latter feature is in contrast to the classical games used in automata theory and descriptive–set theory [30], where the absence of a winning strategy for one player automatically implies the existence of a winning strategy for the other. Moreover, linear logic games [31] differ from our work in the sense that these are symmetric while our games are not precisely because draw positions model non–constructiveness of a system response. Besides, our strategies, unlike those used in type theory [32] or proof nets [33], do not have computational meaning ("proofs", "program") themselves. They are extensional in that they generate constructive truth–values for the interpretation of signals. Our work differs from related work of De Alfaro and Henzinger [34], [35] where the game board (*interface automata*) represents explicit synchronisation dynamics. In our case of synchronous step responses we are interested only in the stationary behaviour, so the execution sequences and interleaving on the game board (*mazes*) are abstracted away.

In order to keep a strong link with synchronous programming, our interpretation of reactive components as specifications of intensional logic is not as general as it perhaps could be. Specifically, we do not consider signal un–emissions (*i.e.*, negated actions). Logical game–theory would not have difficulties to handle them, yet their practical relevance is not obvious. In particular, there seems to be an intrinsic asymmetry between signal presence and signal absence in synchronous programming. Signal presence can be enforced by explicit signal emissions (*e.g.*, the statement emit *s* in ESTEREL) while signal absence is derived as the lack of emissions (there is no statement such as unemit *s* in ESTEREL). We note that the asymmetry between s and $\neg s$ on the action side of a transition is also reflected in the fact that data communication in synchronous programming is invariably associated with signal presence rather than signal absence.

Notions of constructiveness based on games and winning conditions also play an important role in *Normal Logic Programming* (LP), which extends standard definite Horn clause programming by permitting negative literals in clause bodies and queries. Various types of models based on three- and many-valued interpretations have been developed in the literature for normal logic programs. We refer the reader to [36] for a survey of the classic results. There is, however, an important methodological difference between logic programming and synchronous languages: Since synchronous programs often model embedded and reactive systems with some degree of (low–level) asynchrony, it is essential that non–determinism and concurrency are represented adequately. LP, on the other hand is based on a strong sequential execution model, which even constrains the order in which clauses and literals are executed. In this sense the work presented here aims at a rather more general setting than what is considered in LP. In another sense, though, our scope is more restricted, *viz.* in considering only propositional programs. We believe that generating constructive models of Horn clauses from winning conditions may provide further insights into the relationship between operational and denotational semantics of LP. At the propositional level the maze responses are related to the *three-valued models* of Fitting as defined in [36].

7. Conclusions

Game theory handles cyclic systems of non-monotonic behaviours by capturing the system and environment dichotomy through the binary polarity of player and opponent, so that the swapping of roles gives constructive (intensional) meaning to negation. In this paper we have demonstrated the versatility of this idea for the semantics of step responses in synchronous programming, particularly in ESTEREL. We have described the reaction of a composite system to stimuli from its environment as a game played by the individual sub–systems in which these negotiate between themselves the final outcome. This negotiation is governed by game rules determining a constructive response. Concretely, this paper presents a game-theoretic semantics for combinational ESTEREL programs, *i.e.*, for the kernel fragment of ESTEREL corresponding to combinational circuits. Our approach translates combinational programs into finite two-player games in such a way that ESTEREL's *must*– and *cannot*-analysis of signal statuses could be rephrased as the computation of winning strategies.

The constructive reference semantical framework for the synchronous programming language ESTEREL includes the behavioral, operational, circuit-based, and model-theoretic semantics [17], [18]. Since these approaches are equivalent, our results (correspondence between the game-theoretic and behavioural semantics) show that the game interpretation complements and brings a fresh view to the existing ESTEREL's semantical framework. Although the immediate benefits of the game-theoretic setting for improvements on the algorithmic side may be modest, its main contribution refers directly to (*i*) a novel didactic perspective and (*ii*) the possibility of analysing seemingly separate semantical frameworks and causality issues (not only specific to ESTEREL) from a single perspective. Let us be more specific regarding these points.

ESTEREL's constructive approach has been traditionally based in the physics of electronic designs. In fact, the very notion of a constructive program is deeply rooted on this idea, namely a program is considered to be constructive iff its corresponding circuit electrically stabilises for all (gate and wire) delays (even in the presence of combinational cycles) [2], [37]. Thus, most ESTEREL's compilers concentrate on the translation of programs into constructive circuits and associated optimisations techniques. However, according to Berry [17], the way from the behavioural semantics to Boolean circuits is far from trivial. The game approach, on the other hand, provides a different operational view to the formal definition of what a program means (*i.e.*, behavioural semantics) directed to professionals not versed in hardware. Concretely, this is done by given emphasis to more intuition

(game graphs). Thus, the maze game offers an alternative for familiarising software engineers, developers and programmers with this intricate constructive semantics and related notions like constructiveness and causality analysis.

This paper extends our previous work [38] to a larger fragment of ESTEREL, which includes sequential composition and local signal declaration. We argue that this is more than just a small extension and involves several considerations. First, causality analysis is a technique for verifying constructiveness of programs even in the presence of instruction re-orderings done by the run–time system. Obviously, these interleavings are outside the control of any compiler or instruction set. Now, in particular, ESTEREL's causality analysis verifies determinism and boundedness (at the macro-step) under any arbitrary scheduling that preserves emit-test (write-read) dependencies. Second, the consideration of a sequential composition and local declaration for ESTEREL forces the causality analysis to verify that the ordering and locality of the statements (as specified by the programmer) are sequentially and globally consistent, respectively, under ESTEREL's run-time scheduling model. Technically, this is equivalent to check delay-insensitivity for (cyclic) combinational circuits subject to non-inertial delays, or provability of bounded response in constructive modal logic. In order to handle and integrate all these aspects while keeping an intuitive nature, our extension has required a non trivial enrichment of the information content held by terms and new definitions of the terms for all constructs.

In a more recent work [39] we have identified various levels of semantics for synchronous responses in a game–theoretic fashion which correspond to increasing restrictions on winning conditions. Each level is associated with a particular degree of computational constructiveness reflecting a characteristic operational interpretation of system execution. These levels represent classical, coherent (inertial), Statecharts (Pnueli & Shalev) and ESTEREL responses respectively. The analysis of [39], however, does not consider neither sequential composition or local signal declarations. In this form, the extension of this paper can be applied directly to other synchronous interpretations incorporating the semantic principles of synchrony and causality. For instance, using the framework of this paper, it is pretty much straightforward to define a Pnueli & Shalev Statecharts-like semantics that incorporates sequential composition by simply considering the winning conditions of [39]. Yet another possibility is that studying the notion of *sequentially constructive concurrency* (SC) that we have proposed recently [40] but in a game-theoretic manner. SC is a less conservative interpretation of synchrony regarding sequential composition as compared to ESTEREL, which is applicable in situations where the compiler has more control on the concurrent run-time behaviour. Specifically, SC allows some re-orderings of write-read statements provided that the sequential constructs of the program give enough scheduling information to avoid causality conditions. Since the mazes of this paper include sequential composition, it is now possible to investigate how SC semantics can be modelled in our game-theoretic framework by establishing the appropriate SC winning conditions or otherwise.

Finally, we believe that our approach is amenable to a rigorous algebraic treatment since we present mazes in a process-algebraic fashion both in style of a term-based syntax and a transition-systems-based semantics. This would provide a compositional semantics for our games and allow for the minimisation of mazes. Note that the translation from combinational ESTEREL programs into mazes presented in this paper does not apply any optimisations.

References

[1] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone, "The synchronous languages twelve years later," in *Proceedings of the IEEE*, Special Issue on Embedded Systems, vol. 91, January 2003, pp. 64-83.

[2] G. Berry, "The Foundations of Esterel," in *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000, pp. 425-454.

[3] N. Halbwachs, "Synchronous programming of reactive systems, a tutorial and commented bibliography," in *Tenth International Conference on Computer-Aided Verification, CAV '98*. Vancouver (B.C.): LNCS 1427, Springer Verlag, June 1998.

[4] ——, *Synchronous Programming of Reactive Systems*. Berlin, Heidelberg: Springer-Verlag, 2010.

[5] F. Jahanian and A. Mok, "Modechart: A specification language for real-time systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 12, pp. 933-947, 1994.

[6] D. Harel and A. Naamad, "The STATEMATE semantics of Statecharts," *ACM Transactions on Software Engineering*, vol. 5, no. 4, pp. 293-333, October 1996.

[7] N. Leveson, M. Heimdahl, H. Hildreth, and J. Reese, "Requirements specification for process-control systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 9, pp. 684-707, September 1994.

[8] V. Saraswat, R. Jagadeesan, and V. Gupta, "Foundations of timed concurrent constraint programming," in *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science (LICS 1994)*. Paris, France: IEEE Computer Society Press, July 1994, pp. 71-80.

[9] K. Apt, H. Blair, and A. Walker, "Towards a theory of declarative knowledge," in *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Los Altos, CA: Morgan Kaufmann, 1988, pp. 89-148.

[10] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231-274, 1987.

[11] C. Huizing, R. Gerth, and W. de Roever, "Modeling Statecharts behavior in a fully abstract way," in *13th Colloquium on Trees in Algebra and Programming (CAAP '88)*, ser. LNCS, M. Dauchet and M. Nivat, Eds., vol. 299. Nancy, France: Springer-Verlag, March 1988, pp. 271-294.

[12] G. L¨ uttgen, M. von der Beeck, and R. Cleaveland, "Statecharts via process algebra," in *10th International Conference on Concurrency Theory (CONCUR '99)*, ser. LNCS, J. Baeten and S. Mauw, Eds., vol. 1664. Eindhoven, The Netherlands: Springer-Verlag, August 1999, pp. 399-414.

[13] A. Maggiolo-Schettini, A. Peron, and S. Tini, "Equivalences of Statecharts," in *7th International Conference on Concurrency Theory (CONCUR '96)*, ser. LNCS, U. Montanari and V. Sassone, Eds., vol. 1119. Pisa, Italy,: Springer-Verlag, August 1996, pp. 687-702.

[14] A. Pnueli and M. Shalev, "What is in a step: On the semantics of Statecharts," in *Theoretical Aspects of Computer Software (TACS '91)*, ser. LNCS, T. Ito and A. Meyer, Eds., vol. 526. Sendai, Japan: Springer-Verlag, September 1991, pp. 244-264.

[15] G. Berry and L. Cosserat, "The ESTEREL Synchronous

Programming Language and its Mathematical Semantics," in *Seminar on Concurrency, Carnegie-Mellon University*, ser. LNCS, vol. 197. Springer-Verlag, 1984, pp. 389-448.

[16] G. Berry and G. Gonthier, "The Esterel synchronous programming language: Design, semantics, implementation," *Science of Computer Programming*, vol. 19, no. 2, pp. 87-152, 1992.

[17] G. Berry, *The Constructive Semantics of Pure Esterel*. Draft Book, July 1999.

[18] G. Lüttgen and M. Mendler, "Towards a Model-Theory for Esterel," *Electronic Notes in Theoretical Computer Science*, vol. 65, no. 5, pp. 95-109, 2002.

[19] S. Abramsky, "Games in the semantics of programming languages," in *Proceedings of the 11th Amsterdam Colloquium*, P. Dekker, M. Stokhof, and Y. Venema, Eds. ILLC, Dept. of Philosophy, University of Amsterdam, 1997, pp.1-6.

[20] A. Pietarinen and G. Sandu, "Games in philosophical logic," *Nordic Journal of Philosophical Logic*, vol. 4, pp. 143-173, 1999.

[21] J. M. E. Hyland and C.-H. L. Ong, "On full abstraction for PCF: I, II and III," *Inform. and Comput.*, vol. 163, no. 2, pp. 285-408, 2000. [Online]. Available: http://dx.doi.org/10.1006/inco.2000.2917

[22] S. Abramsky and R. Jagadeesan, "Games and full completeness for multiplicative linear logic," *Journal of Symbolic Logic*, vol. 59, no. 2, pp. 543-574, 1994.

[23] P. Baillot, V. Danos, T. Ehrhard, and L. Regnier, "Believe it or not, AJM's games model is a model of classical linear logic," in *LICS'97*, 1997, pp. 68-75.

[24] S. Abramsky and P.-A. Mellies, "Concurrent games and full completeness," in *Proceedings of the Fourteenth International Symposium on Logic in Computer Science*. Computer Society Press of the IEEE, 1999, pp. 431-442.

[25] J.-Y. Girard, "Locus solum: From the rules of logic to the logic of rules," *Mathematical Structures in Computer Science*, vol. 11, no. 3, pp. 301-506, June 2001.

[26] P. Malacaria and C. Hankin, "Generalised flowcharts and games," in ICALP, ser. Lecture Notes in Computer Science, K. G. Larsen, S. Skyum, and G. Winskel, Eds., vol. 1443. Springer, 1998, pp. 363-374.

[27] D. Potop-Butucaru, S. A. Edwards, and G. Berry, *Compiling Esterel*, 1st ed. Springer Publishing Company, Incorporated, 2007.

[28] G. Schmidt and T. Str ¨ohlein, "On kernel of graphs and solutions of games: A synopsis based on relations and fixpoints," *SIAM J. Algebraic Discrete Methods*, vol. 6, pp. 54-65, 1985.

[29] J. Hintikka, *The Principles of Mathematics Revisited*. Cambridge University Press, 1996.

[30] W. Thomas, "On the synthesis of strategies in infinite games," in STACS '95, ser. LNCS, vol. 900, 1995, pp. 1-13.

[31] R. Accorsi, P. Dr, and J. van Benthem, "Lorenzen's games and linear logic," 1999.

[32] J.-Y. Girard, P. Taylor, and Y. Lafont, *Proofs and types*. New York, NY, USA: Cambridge University Press, 1989.

[33] M. Horbach, "Proof nets for intuitionistic logic-Diploma Thesis," Saarbr ¨ ucken, Germany, 2006.

[34] L. de Alfaro and T. A. Henzinger, "Interface automata," in *Joint 8th European Software Engineering Conference and 9th International Symposium on Foundations of Software Engineering (ESEC/FSE'01)*, 2001, pp. 109-120.

[35] L. de Alfaro, T. A. Henzinger, and M. Stoelinga, "Timed interfaces," in *International Workshop on Embedded Software* (EMSOFT 2002). Springer, 2002, pp. 108-122.

[36] J. C. Shepherdson, "Logics for negation as failure," in *Logic from Computer Science*, Y. N. Moschovakis, Ed. Springer, 1991, pp. 521-583.

[37] D. Potop-Butucaru, S. A. Edwards, and G. Berry, *Compiling Esterel*. Springer-Verlag, 2007.

[38] J. Aguado, G. L¨ uttgen, and M. Mendler, "A-maze-ing Esterel," in *Synchronous Languages, Applications, and Programming (SLAP '03)*, ser. Electronic Notes in Theoretical Computer Science, A. Girault, F. Maraninchi, and E. Rutten, Eds., vol. 88. Porto, Portugal: Elsevier Science, July 2003, pp. 21-37.

[39] J. Aguado and M. Mendler, "Constructive Semantics for Instantaneous Reactions," in *Theoretical Computer Science*, vol. 412, March 2011, pp. 931-961.

[40] R. von Hanxleden, M. Mendler, J. Aguado, B. Duderstadt, I. Fuhrmann, C. Motika, S. Mercer, and O. O'Brien, "Sequentially constructive concurrency: a conservative extension of the synchronous model of computation," in *DATE*, 2013, pp. 581-586.

About the Author

Joaquín Aguado, Ph.D

Since 2002 has been working in various academic positions (researcher, lecturer, academic advisor) at the Informatics Theory Group of the University of Bamberg in Germany.