A Condensed Goal-Independent Bottom-Up Fixpoint Semantics Modeling the Behavior of *tccp*

M. Comini, L. Titolo, A. Villanueva

Abstract

In this paper, we present a new compositional bottom-up semantics for the *Timed Concurrent Constraint Language* (*tccp* in short). Such semantics is defined for the full language. In particular, is able to deal with the non-monotonic characteristic of the language, which constitutes a substantial additional technical difficulty w.r.t. other compositional denotational semantics present in literature (which do not tackle the full language).

The semantics is proved to be (correct and) fully abstract w.r.t. the full behavior of *tccp*, including infinite computations. This is particularly important since *tccp* has been defined to model reactive systems.

The overall of these features makes our proposal particularly suitable as the basis for the definition of semantic-based program manipulation tools (like analyzers, debuggers or verifiers), especially in the context of reactive systems.

1 Introduction

The concurrent constraint paradigm (*ccp* in short; [23]) is a simple but powerful model for concurrent systems. It is different from other programming paradigms mainly due to the notion of store-as-constraint that replaces the classical store-as-valuation model. In this way, the languages from this paradigm can easily deal with partial information: an underlying constraint system handles constraints on system variables.

In the literature, much effort has been devoted to the development of *appropriate* denotational semantics for languages in the *ccp* paradigm (e.g. [13, 8, 14]). Compositionality and fully abstraction are two highly desirable properties for a semantics, since they are needed for many purposes (for example as the basis for program analysis or verification tasks). A fully abstract model can be considered *the* semantics of a language [13].

In [8], the difficulties for handling nondeterminism and infinite behavior in the *ccp* paradigm was investigated. The authors showed that the presence of nondeterminism and synchronization require relatively complex structures for the denotational model of (non timed) *ccp* languages. In most *ccp* languages, nondeterminism is defined in terms of a global choice, which poses more difficulties than a local-choice model [14].

Within the *ccp* family, [10] introduced the *Timed Concurrent Constraint Language* (*tccp* in short) by adding to the original *ccp* model the notion of time and the ability to capture the absence of information. With these features, it is possible to specify behaviors typical of reactive systems such as *timeouts* or *preemption* actions, but they also make the language non-monotonic. For timed concurrent constraint languages, the presence of non-determinism, local variables and also of timing constructs which are able to handle negative information significantly complicates the definition of compositional, fully-abstract semantics. Moreover, infinite behaviors (which become natural in the timed extensions) are an additional nightmare.

For the definition of a compositional semantics, the solutions to all the mentioned difficulties (both for *ccp* and *tccp*) has been traditionally based on the introduction of *severe* restrictions on the language. However, since we are interested in applying the semantics to develop (semantics-based) program manipulation tool (like debuggers, verifiers and analyzers) for us this solution is simply not feasible.

Thus we have developed a new (small-step) *compositional* semantics which is (correct and) *fully abstract* w.r.t. the small-step behavior of *full tccp*. It is based on the evaluation of agents over a denotation for a set of process declarations D, obtained as least fixpoint of a (continuous) immediate consequence operator $\mathcal{D}[\![D]\!]$. The key idea to be able to handle *tccp*'s non-monotonicity actually comes from a tentative to define a *condensed* semantics for *tccp*. In the context of *ccp* a semantics is condensed if the denotation $\mathcal{A}[\![A]\!]$ of an agent A contains the minimal information such that the semantics of A for any initial store c can be obtained by merging c into $\mathcal{A}[\![A]\!]$.

The resulting idea is to enrich behavioral timed traces with information about the essential conditions that the store must (or must not) satisfy in order to make the program proceed with one or another execution branch. Thus, we associate conditions to the store of each computation step and then we collect (only) the most general hypothetical computations. These conditions are constructed by using the information in the guards of the ask and now constructs of a program.

In this way, we obtain a condensed semantics which deals with non-monotonicity, since into denotations we have the *minimal* information that has to be used to exploit computations arising from absence of information.

Note that, like it has been done in all proposals for semantics of *ccp*, to define our semantics we make the assumption of *closed world*. This means that the specified system models also the environment so that no information can be added by an external element during the computation (apart from the initial store). In practice, this assumption means that, for example, if we are defining a system that interacts with users, the actions that the user can perform must be modeled in advance, usually as an additional process declaration.

Since *tccp* was originally defined to model reactive systems, that many times include systems that do not terminate *with a purpose*, we have developed our semantics to distinguish among *terminating computations*, *suspending computations*, and *non-terminating computations*. This improves the original semantics for *tccp* defined in [10] which merges suspending and non-terminating computations. In particular, terminating computations are those that reach a point in which no agents are pending to be executed. In such cases, we can consider (as [10] does) that the last computed store is the output of the computation.

Suspending computations are those that reach a point in which there are some agents pending to be executed, but there is not enough information in the store to entail the conditions that would make them evolve. In [10], these computations are identified with terminating computations, thus the last computed store is also considered the output of the computation. We (can and want) distinguish these two kinds of computations since, conceptually, a suspended computation has not completely finished its execution, and, in some cases, it could be a symptom of a system error. Finally, non-terminating computations are those that do not suspend and do not terminate, i.e., those that continue to have agents to be executed (e.g. a loop with a true condition). Also in this case, we are able to distinguish these computations from the suspended computations, in particular in the case when the non-terminating computation does not modify the store (and thus one could think that it is suspended).

To conclude, we also define a big-step semantics (by abstraction of our small-step semantics) which tackles also outputs of infinite computations. We prove that its fragment for finite computations is (essentially) isomorphic to the traditional big-step semantics of [10]. Moreover, we also formally prove that it is not possible to have a correct input-output semantics which is defined *solely* on the information provided by the input/output pairs (i.e., some more information into denotations is needed).

Organization of the paper. The rest of the paper is organized as follows. Section 2 recalls the foundations of the *tccp* language. Section 3 introduces our small-step denotational semantics. It also includes illustrative examples for the main concepts. Then Section 4 introduces our big-step semantics for *tccp* and formally relates it to the (original) one of [10]. Finally, Section 5 concludes.

To improve readability of the paper, the most technical results and the proofs (of all results) are shown in Appendix A.

2 Preliminaries

The languages defined within the *ccp* paradigm (as extensions of the original model of Saraswat in [26]) are parametric w.r.t. a cylindric constraint system. The constraint system handles the data information of the program in terms of constraints.

2.1 Cylindric constraint systems

The notion of constraint systems able to handle queries with existential quantified variables was first described in [23]. A more elegant formalization, the cylindric constraint systems, was introduced in [26], where a *hiding* operator is defined in terms of a general notion of existential quantifier. However, since we are dealing with *tccp*, in this work we prefer to use the formalization of [10].

A *cylindric constraint system* is an algebraic structure of the form $\mathbf{C} = \langle \mathcal{C}, \leq, \otimes, tt, \mathit{ff}, \mathit{Var}, \exists \rangle$ where *Var* is a denumerable set of variables and such that:

1. $(\mathcal{C}, \leq)$ is a complete algebraic lattice where $\otimes$ is the *lub* operator, and ff and tt are respectively the greatest and least element of $\mathcal{C}$.

2. For each $x \in Var$ there exist a *cylindric operator* $\exists_x : \mathcal{C} \to \mathcal{C}$ such that, for any $c, d \in \mathcal{C}$,

$$\begin{array}{ll} c \vdash \exists_x c & c \vdash d \Rightarrow \exists_x c \vdash \exists_x d \\ \exists_x(\exists_y c) = \exists_y(\exists_x c) & \exists_x(c \otimes \exists_x d) = \exists_x c \otimes \exists_x d \end{array}$$

Following the standard terminology and notation, instead of $\leq$ we often use its inverse relation, denoted $\vdash$ and called *entailment*. Moreover, $\oplus$ denotes the *glb* of $(\mathcal{C}, \leq)$.

We also abuse of notation and, given $C \subseteq \mathcal{C}$, write $\exists_x C$ for $\{\exists_x c \mid c \in C\}$.

We can find in the literature several examples of cylindric constraint systems that are useful when modeling data structures, logic programs or other specific domains [27, 8, 9, 2]. In the illustrative examples throughout the paper we will use, for the sake of simplicity, the following classical cylindric constraint system. Constraints of $\mathcal{L}$ are formed by taking equivalence classes, modulo logical equivalence $\Leftrightarrow$, of finite conjunctions of either linear disequalities (strict and not) or equalities over $\mathbb{Z}$ and $Var = \{x, y, \dots\}$ (e.g. $x > 4$, $y \geq 10 \wedge w < -3$, ...). The entailment relation is implication $\Rightarrow$ (thus, the order in the lattice is $\Leftarrow$). The *lub* is conjunction $\wedge$ and $\exists_x$ is the operation which removes, after information has been propagated within a constraint, all conjuncts referring to variable x (e.g. $\exists_x(x = y \wedge x > 3) = y > 3$). It can be easily verified that $\mathbf{L} \coloneqq \langle \mathcal{L}, \Leftarrow, \wedge, true, false, Var, \exists \rangle$ is a cylindric constraint system.

2.2 Timed Concurrent Constraint Programming

The *tccp* language, introduced in [10], is particularly suitable to specify both reactive and time critical systems. In *tccp*, the computation progresses as the concurrent and asynchronous activity of several agents that can (monotonically) accumulate information in a *store*, or query information from that store. The notion of time is introduced by defining a discrete and global clock[1] and progresses depending on the kind of agents as defined in the following.

Given a cylindric constraint system $\mathbf{C} = \langle \mathcal{C}, \leq, \otimes, tt, ff, Var, \exists \rangle$ and a set of process symbols Π, the syntax of agents is given by the following grammar:

$$A ::= \mathsf{skip} \mid \mathsf{tell}(c) \mid A \parallel A \mid \exists x\, A \mid \sum_{i=1}^{n} \mathsf{ask}(c_i) \to A \mid \mathsf{now}\, c\; \mathsf{then}\, A\; \mathsf{else}\, A \mid p(x_1, \dots, x_m)$$

where c, $c_1, \dots, c_n$ are finite constraints in $\mathcal{C}$; $p_{/m} \in \Pi$ and $x, x_1, \dots, x_m \in Var$.

A *tccp* program P is an object of the form $D\,.\,A$, where A is an agent, called *initial agent*, and D is a set of *process declarations* of the form $p(\vec{x}) \coloneq A$ (for some agent A), where $\vec{x}$ denotes a generic tuple of variables.

The following definition introduces the operational semantics of the language. It is slightly different from the original one in [10]. In particular, we have introduced conditions in specific rules (namely Rules **R2**, **R4** and **R10**) in order to detect when the store becomes ff. This modification follows the philosophy of computations defined in [27], where computations that reach an inconsistent store are considered failure computations. In [10], this check is not explicitly done. In our context, we are interested in detecting when a computation reaches ff; however, once ff is reached, no action can modify the store (ff

[1] Differently from other languages where time is explicitly introduced by defining new *timing* agents.

$$\frac{}{\langle \mathsf{tell}(c),\, d\rangle \to \langle \mathsf{skip},\, c \otimes d\rangle}\; d \neq f\!f \tag{R1}$$

$$\frac{}{\langle \sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i,\, d\rangle \to \langle A_j,\, d\rangle}\; j \in [1,n],\, d \vdash c_j,\, d \neq f\!f \tag{R2}$$

$$\frac{\langle A,\, d\rangle \to \langle A',\, d'\rangle}{\langle \mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B,\, d\rangle \to \langle A',\, d'\rangle}\; d \vdash c \tag{R3}$$

$$\frac{\langle A,\, d\rangle \not\to}{\langle \mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B,\, d\rangle \to \langle A,\, d\rangle}\; d \vdash c,\, d \neq f\!f \tag{R4}$$

$$\frac{\langle B,\, d\rangle \to \langle B',\, d'\rangle}{\langle \mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B,\, d\rangle \to \langle B',\, d'\rangle}\; d \nvdash c \tag{R5}$$

$$\frac{\langle B,\, d\rangle \not\to}{\langle \mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B,\, d\rangle \to \langle B,\, d\rangle}\; d \nvdash c \tag{R6}$$

$$\frac{\langle A,\, d\rangle \to \langle A',\, d'\rangle \quad \langle B,\, d\rangle \to \langle B',\, c'\rangle}{\langle A \parallel B,\, d\rangle \to \langle A' \parallel B',\, d' \otimes c'\rangle} \tag{R7}$$

$$\frac{\langle A,\, d\rangle \to \langle A',\, d'\rangle \quad \langle B,\, d\rangle \not\to}{\langle A \parallel B,\, d\rangle \to \langle A' \parallel B,\, d'\rangle} \tag{R8}$$

$$\frac{\langle A,\, l \otimes \exists_x d\rangle \to \langle B,\, l'\rangle}{\langle \exists^l x\, A,\, d\rangle \to \langle \exists^{l'} x\, B,\, d \otimes \exists_x l'\rangle} \tag{R9}$$

$$\frac{}{\langle p(x),\, d\rangle \to \langle A,\, d\rangle}\; p(\vec{x}) \coloneq A \in D,\, d \neq f\!f \tag{R10}$$

Figure 1: The transition system for *tccp*.

is the greatest element in the domain), and the guards in the program agents are always entailed, thus the computation from that instant has little interest.

Definition 2.1 (Operational semantics of *tccp*) *The* operational semantics *of* tccp *is formally described by a transition system $T = (Conf, \to)$. Configurations in Conf are pairs $\langle A,\, c\rangle$ representing the agent to be executed (A) and the current global store (c). The transition relation $\to \subseteq Conf \times Conf$ is the least relation satisfying the rules of Figure 1. Each transition step takes exactly one time-unit. In the sequel $\to^*$ denotes the reflexive and transitive closure of the relation $\to$.*

As usually done, we assume that the tccp *system is closed under the usual structural equivalence relation where the parallelism operator is commutative and associative, and agents $A \parallel$ skip and A are equivalent.*

As can be seen from the rules, the skip agent represents the successful termination of the computation. The $\mathsf{tell}(c)$ agent adds the constraint c to the current store and then stops. It takes one time-unit, thus the constraint c is visible to other agents from the following time instant. The store is updated by means of the $\otimes$ operator of the constraint system. The choice agent $\sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i$ consults the store and non-deterministically executes (at the following time instant) one of the agents A_i whose corresponding guard c_i is entailed by the current store; otherwise, if no guard is entailed by the store, the agent suspends.

The conditional agent $\mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, B$ behaves in the current time instant like A (respectively B) if c is (respectively is not) entailed by the store. Note that, because of the ability of *tccp* to handle partial information, $d \nvdash c$ is not equivalent to $d \vdash \neg c$. Thus, the else branch is taken not only when the condition is falsified, but also when there is not enough information to entail the condition. This characteristic is known in the literature as the ability to process "negative information" [24, 25].

$A \parallel B$ models the parallel composition of A and B in terms of maximal parallelism (in contrast to the interleaving approach of *ccp*), i.e., all the enabled agents of A and B are executed at the same time. The agent $\exists x\, A$ makes variable x local to A. To this end, it uses the $\exists$ operator of the constraint system. More specifically, it behaves like A with x considered local, i.e., the information on x provided by the external environment is hided to A, and the information on x produced by A is hided to the external world. In [10], an auxiliary construct $\exists^d x$ is used to explicitly show the store local to A. In particular, in Rule **R9**, the store d in the agent $\exists^d x\, A$ represents the store local to A. This auxiliary construct is linked to the hiding one by setting the initial local store to tt, thus $\exists x\, A \coloneqq \exists^{tt} x\, A$.

Finally, the agent $p(x)$ takes from D a declaration of the form $p(\vec{x}) \coloneq A$ and then executes A at the following time instant. For the sake of simplicity, we assume that sets of declarations D are closed w.r.t. renaming of parameter names, i.e., if $p(\vec{x}) \coloneq A \in D$ then, for any $y \in Var$, also $p(\vec{x}) \coloneq A\{\vec{x}/\vec{y}\} \in D$ [2].

3 Modeling the small-step operational behavior of *tccp*

In this section, we introduce a new condensed, compositional, bottom-up denotational semantics which is correct (and fully abstract) w.r.t. the small-step (operational) behavior of *tccp* (Definition 3.1).

In order to introduce such semantics, we need first to define some (technical) notions. In the sequel, all definitions are parametric w.r.t. a cylindric constraint system $\mathbf{C} = \langle \mathcal{C}, \leq, \otimes, tt, ff, Var, \exists \rangle$. We denote by $\mathbb{A}^{\Pi}_{\mathbf{C}}$ the set of agents and $\mathbb{D}^{\Pi}_{\mathbf{C}}$ the set of sets of process declarations built on signature Π and constraint system $\mathbf{C}$. By ϵ we denote the empty sequence; by $last(s)$ the last element of a non-empty sequence s; by $s_1 \cdot s_2$ the concatenation of two sequences s_1, s_2. We also abuse notation and, given a set of sequences S, by $s_1 \cdot S$ we denote $\{s_1 \cdot s_2 \mid s_2 \in S\}$.

Let us formalize first the notion of behavior of a set D of process declarations in terms of the transition system described in Figure 1. It collects all the small-step computations associated to D as the set of (all the prefixes of) the sequences of computation steps (in terms of sequences of stores), for all possible initial agents and stores.

Definition 3.1 *The* small-step (observable) behavior *of $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ is defined as:*

$$\mathcal{B}^{ss}[\![D]\!] \coloneqq \bigcup_{\forall c \in \mathbf{C}, \forall A \in \mathbb{A}^{\Pi}_{\mathbf{C}}} \mathcal{B}^{ss}[\![D\, .\, A]\!]_c \qquad \textit{where}$$

[2] This assumption is equivalent to use the diagonal elements of the constraint system: given the agent $p(x)$ and a declaration of the form $p(\vec{y}) \coloneq A$, we *diagonalize* the agent A before execution, i.e., we execute $\exists^{d_{x_1 y_1} \otimes \cdots \otimes d_{x_n y_n}} \vec{x}\, A$ at the following time instant.

$$\mathcal{B}^{ss}[\![D\,.\,A]\!]_c \coloneqq \left\{c \cdot c_1 \cdot \dots \cdot c_n \;\middle|\; \langle A,\, c\rangle \to \langle A_1,\, c_1\rangle \to \ldots \to \langle A_n,\, c_n\rangle\right\} \cup \{\epsilon\}$$

and $\to$ *is the transition relation given in Figure 1.*

We call the sequences in $\mathcal{B}^{ss}[\![D\,.\,A]\!]_c$ behavioral timed traces *or simply* traces *(when clear from the context).*

We denote by $\approx_{ss}$ *the equivalence relation between process declarations induced by* $\mathcal{B}^{ss}$*, namely* $D_1 \approx_{ss} D_2 \Leftrightarrow \mathcal{B}^{ss}[\![D_1]\!] = \mathcal{B}^{ss}[\![D_2]\!]$.

Our goal is to define a semantics $\mathcal{S}$ which is fully abstract w.r.t. $\approx_{ss}$ (i.e., $\mathcal{S}[\![D_1]\!] = \mathcal{S}[\![D_2]\!] \iff D_1 \approx_{ss} D_2$) and has all the properties mentioned in the introduction (i.e., being compositional, condensed, goal-independent and bottom-up). Usually, a condensed (collecting) semantics is obtained by defining denotations based only on the most general traces (i.e., those for the weakest store). Thanks to this, the size of denotations is reduced significantly. The problem in following this approach in the *tccp* case is that $\mathcal{B}^{ss}$ is not condensing since not all behavioral sequences can be retrieved from the most general ones. This is due to the ask, now and hiding constructs. For instance, consider the agent $A \coloneqq \mathsf{now}\ x = 3\ \mathsf{then}\ \mathsf{tell}(z = 0)\ \mathsf{else}\ \mathsf{tell}(z = 1)$. Given the initial store $x \geq 3$, we obtain the trace $x \geq 3 \cdot (x \geq 3 \wedge z = 1)$, while for the stronger initial store $x = 3$ we obtain the trace $x = 3 \cdot (x = 3 \wedge z = 0)$, which is not comparable to the former (since $z = 0 \not\Rightarrow z = 1$ and $z = 1 \not\Rightarrow z = 0$). Hence, the latter trace cannot be obtained from the former trace, which has been generated for a *more general* store. Indeed, in general, in *tccp*, given $X \coloneqq \mathcal{B}^{ss}[\![D\,.\,A]\!]_c$ (the set of traces for an agent A with initial store c), if we compute $\mathcal{B}^{ss}[\![D\,.\,A]\!]_d$ with a stronger initial store d ($d \vdash c$), then some traces of X may disappear and, what is more critical, new traces, *which are not instances* of the ones in X, can appear. This characteristic is known, in the community of the *ccp* paradigm [8, 25], as "non-monotonicity of the *tccp* language" but it can also be expressed as "the *tccp* language is not condensing".

Because of non-monotonicity of *tccp*, $\mathcal{B}^{ss}$ is also not compositional. For instance, consider the agents $A_1 \coloneqq \mathsf{tell}(x = 1)$ and

$$A_2 \coloneqq \mathsf{ask}(\mathit{true}) \to \mathsf{now}\ (x = 1)\ \mathsf{then}\ \mathsf{tell}(y = 0)\ \mathsf{else}\ \mathsf{tell}(y = 1)$$

For each c, $\mathcal{B}^{ss}[\![\varnothing\,.\,A_1]\!]_c = \{c \cdot (x = 1 \wedge c)\}$. Moreover, for each c that implies[3] $x = 1$, $\mathcal{B}^{ss}[\![\varnothing\,.\,A_2]\!]_c = \{c \cdot c \cdot (y = 0 \wedge c)\}$ while, when $c \not\Rightarrow (x = 1)$, $\mathcal{B}^{ss}[\![\varnothing\,.\,A_2]\!]_c = \{c \cdot c \cdot (y = 1 \wedge c)\}$. Now, for the parallel composition of these agents $A_1 \parallel A_2$, $\mathcal{B}^{ss}[\![\varnothing\,.\,A_1 \parallel A_2]\!]_{\mathit{true}} = \{\mathit{true} \cdot (x = 1) \cdot (x = 1 \wedge y = 0)\}$ which cannot be computed by *merging* the traces of A_1 and A_2.

Thus, it does not come as a surprise that for all non-monotonic languages of the *ccp* paradigm, the compositional semantics that have been written [27, 24, 8, 9, 14, 20, 16, 21, 15] are not defined for the full language, either because they avoid the constructs that cause non-monotonicity or because they restrict their use. Hence, the ability to handle non-monotonicity (and thus the full language without any limitation) is certainly one of the strengths of our proposal.

The examples above shows why, due to the non-monotonicity of *tccp*, in order to obtain a compositional (and condensed) semantics *for the full language* it is not possible to follow the traditional strategy and collect in the semantics the traces associated to the weakest initial store. Note that, having a semantics with these properties, besides the theoretical interest, it is also a matter

[3]In this Cylindric Constraint System the entailment is logical implication.

of pragmatical relevance since they are the key factors for having an effective implementation which computes the semantics.

Actually, we have found the solution to the problem just mentioned by trying to solve another (related) problem. Since in a top-down (goal-dependent) approach the (initial) current store is propagated, then the decisions regarding a conditional or choice agent (where the computation evolves depending on the entailment of the guards in the current store) can be taken immediately. However, if we want to define a fixpoint semantics which builds the denotations bottom-up we have the problem that, while we are building the fixpoint, we do not know the current store yet. Thus, it is impossible to know *which* execution branch *has to be* taken in correspondence of a program's guard.

Our idea is to enrich behavioral timed traces with information about the essential conditions that the store must (or must not) satisfy in order to make the program proceed with one or another execution branch. Thus, we associate conditions to the store of each computation step and then we collect (only) the most general hypothetical computations. These conditions are constructed by using the information in the guards of the ask and now constructs of a program. We will see that this solves both the problem of constructing the semantics bottom-up and of having a compositional and condensed semantics coping with non-monotonicity.

3.1 The semantic domain

Let us start by introducing the notion of condition, that is the base to build our denotations. Intuitively, we need "positive conditions" for branches related to the entailment of guards and "negative conditions" for non-entailment, i.e., for the branches where the current store does not entail the associated condition.

Definition 3.2 (Conditions) *A* condition η *is a pair* $\eta = (\eta^+, \eta^-)$ *where*

- $\eta^+ \in \mathbf{C}$ *is called* positive condition*, and*
- $\eta^- \in \wp(\mathbf{C})$ *is called* negative condition*.*

Additionally, a condition is valid *when* $\eta^+ \neq \mathit{ff}$*,* $\mathit{tt} \notin \eta^-$ *and* $\forall c \in \eta^- . \, \eta^+ \nvdash c$*.*

We denote $\Lambda_{\mathbf{C}}$ *the set of all conditions and* $\Delta_{\mathbf{C}}$ *the subset of valid ones.*

The conjunction of two conditions $\eta_1 = (\eta_1^+, \eta_1^-)$ *and* $\eta_2 = (\eta_2^+, \eta_2^-)$ *is defined (by abuse of notation) as* $\eta_1 \otimes \eta_2 := (\eta_1^+ \otimes \eta_2^+, \eta_1^- \cup \eta_2^-)$*.*

Two conditions are called incompatible *if their conjunction is not valid.*

A store $c \in \mathbf{C}$ *is* consistent *with* η*, written* $c \gg \eta$*, if* $\eta^+ \otimes c \neq \mathit{ff}$ *and* $\forall h \in \eta^- . \, c \nvdash h$*. Moreover, we say that* c satisfies η*, written* $c \Vdash \eta$*, when* $c \vdash \eta^+$ *and* $\forall h \in \eta^- . \, c \nvdash h$*.*

We extend the existential quantification operator of the constraint system to conditions as $\exists_x (\eta^+, \eta^-) := (\exists_x \eta^+, \exists_x \eta^-)$*.*

Due to the partial nature of the constraint system, for negative conditions we cannot use the disjunction $\bigoplus_{i=1}^n c_i$ instead of $\{c_1, \ldots, c_n\}$ since we can have a store c such that $c \vdash \bigoplus_{i=1}^n c_i$ while $\forall i . \, c \nvdash c_i$. For instance, we can have two guards $x > 2$ and $x \leq 2$ and it may happen that the current store does not satisfy any of them, but their disjunction (*true*) is entailed by any store.

Clearly, if a store different from *ff* satisfies a condition, then it is also consistent with that condition. If two conditions are incompatible, then there exists no constraint $c \in \mathbf{C} \setminus \{\mathit{ff}\}$ that entails simultaneously both conditions.

Now we are ready to enrich with conditions the notion of trace.

Definition 3.3 (Conditional state) *A* conditional state *is either*

conditional store: *a pair $\eta \rightarrowtail c$, for each $\eta \in \Lambda_{\mathbf{C}}$ and $c \in \mathbf{C}$, or*

stuttering: *the construct $stutt(C)$, for each finite $C \subseteq \mathbf{C} \setminus \{tt\}$, or*

end of a process: *the construct $\boxtimes$.*

In a conditional store $t = \eta \rightarrowtail c$, the constraint c is the store *of t.*
We say that $\eta \rightarrowtail c$ is valid *if η is valid.*
We extend $\exists_x$ to conditional states as $\exists_x \big((\eta^+, \eta^-) \rightarrowtail c\big) := \exists_x(\eta^+, \eta^-) \rightarrowtail \exists_x c$, $\exists_x\, stutt(C) := stutt(\exists_x C)$ and $\exists_x \boxtimes := \boxtimes$.

The conditional store $\eta \rightarrowtail c$ is used to represent a hypothetical computation step where η is the condition that the current store must satisfy in order to make the computation proceed. Moreover, c represents the information that is added by the agent to the global store up to the current time instant.

The stuttering $stutt(C)$ is needed to model the suspension of the computation due to an ask construct, i.e., it represents the fact that there is no guard in C (the guards of a choice agent) entailed by the current store. This construct allows us to distinguish a suspended computation from an infinite loop that does not modify the store.

Definition 3.4 (Conditional trace) *A* conditional trace *is a (possibly infinite) sequence $t_1 \cdots t_n \cdots$ of valid conditional states (where $\boxtimes$ can be used only as a terminator) that respects the following properties:*

Monotonicity: *for each $t_i = \eta_i \rightarrowtail c_i$ and $t_j = \eta_j \rightarrowtail c_j$ such that $j \geq i$, $c_j \vdash c_i$.*
Consistency: *for each $t_i = \eta_i \rightarrowtail c_i$ and t_{i+1} either $(\eta^+_{i+1}, \eta^-_{i+1}) \rightarrowtail c_{i+1}$ or $stutt(\eta^-_{i+1})$, $\forall c^- \in \eta^-_{i+1}.\, c_i \nvdash c^-$.*

We denote by $\mathbf{CT}$ the set of all conditional traces.
The limit store *of a (finite or infinite) trace s is the lub of the stores (of the conditional stores) of s.*
A finite conditional trace that is ended with $\boxtimes$ as well as an infinite conditional trace is said, respectively, failed *or* (finitely) successful *depending if its limit store c is ff or not. Such c is called* computed result.
A sequence (of conditional states) that does not satisfy these properties is called an invalid trace.

Each conditional trace models a hypothetical *tccp* computation where the initial store is implicitly tt and, for each time instant, we have a conditional state where each condition represents the information that the global store has to satisfy in order to proceed to the next time instant.

The Monotonicity property is needed since in *tccp*, as well as in *ccp* but not in all its extensions, each store in a computation entails the successive ones. Note that because of this, for any finite conditional trace $t_1, \dots, t_n$ whose sequence of stores (of the conditional stores) is $c_1, \dots, c_m$ $(m \leq n)$, the limit store $\otimes_{i=1}^{m} c_i = c_m$ and thus the computed answer is just the last store c_m.

The Consistency property affirms that the store of a given conditional store cannot be in contradiction with the condition associated to the successive conditional state.

Note that finite conditional traces not ending in $\boxtimes$ are partial traces that can still evolve and thus they are always a prefix of a longer conditional trace.

Example 3.5 ______

It is easy to verify that the sequence $r_1 \coloneqq (\mathit{true}, \varnothing) \rightarrowtail y = 0 \cdot (x > 2, \varnothing) \rightarrowtail y = 0 \wedge z = 3 \cdot \boxtimes$ is a valid conditional trace. The first component of the trace states that in the first time instant the store $y = 0$ is computed in any case (the condition $(\mathit{true}, \varnothing)$ is always satisfied). The second component requires the constraint $x > 2$ to be satisfied by the (global) store in order to proceed by adding to the next state the information $z = 3$.

Instead, the conditional trace $r_2 \coloneqq (\mathit{true}, \varnothing) \rightarrowtail x = 0 \cdot (x = 0, \varnothing) \rightarrowtail \mathit{tt} \cdot \boxtimes$ is invalid since the Monotonicity property does not hold because $\mathit{tt} \nvdash x = 0$. Also $r_3 \coloneqq (\mathit{true}, \varnothing) \rightarrowtail x = 0 \cdot \mathit{stutt}(\{x \geq 0\}) \cdot \boxtimes$ is an invalid conditional trace: it does not satisfy the Consistency property since $x = 0$ implies the (only) negative condition in the successive conditional state $(x \geq 0)$.

Definition 3.6 (Semantic domain) *A set $R \subseteq \mathbf{CT}$ is* closed by prefix *if for each $r \in R$, all the prefixes p of r (denoted as $p \leq_{\mathit{pref}} r$) are also in R.*

We denote the domain of non-empty *sets of conditional traces that are closed by prefix as $\mathbb{P}$ (i.e., $\mathbb{P} \coloneqq \{R \subseteq \mathbf{CT} \mid R \neq \varnothing, r \in R \Rightarrow \forall p \leq_{\mathit{pref}} r. p \in R\}$).*

We order elements in $\mathbb{P}$ by set inclusion $\subseteq$.

It is worth noting that $(\mathbb{P}, \subseteq, \bigcup, \bigcap, \mathbf{CT}, \{\epsilon\})$ is a complete lattice.

This conceptual representation is pretty simple, especially to understand the lattice structure, considered the fact that we admit infinite traces. However, each prefix-closed set contains a lot of redundant traces, which are quite inconvenient for (some) technical definitions. Thus, we will often use an equivalent representation obtained by considering the crown of prefix-closed sets. Namely, given $P \in \mathbb{P}$, we remove all the prefixes of a trace in the set with the function $\mathit{maximal}(P) \coloneqq \{r \in P \mid \nexists p \in P \smallsetminus \{r\}. r \leq_{\mathit{pref}} p\}$. Let $\mathbf{M} \coloneqq \mathit{maximal}(\mathbf{CT})$, $\mathbb{M} \coloneqq \{\mathit{maximal}(P) \mid P \in \mathbb{P}\}$ and call *maximal conditional trace sets* the elements of $\mathbb{M}$. The inverse of *maximal* is, for each $M \in \mathbb{M}$,

$$\mathit{prefix}(M) \coloneqq \{p \mid p \leq_{\mathit{pref}} r, r \in M\}\} \tag{3.1}$$

The order of $\mathbb{M}$ is induced from the one in $\mathbb{P}$ as $M_1 \sqsubseteq M_2 \Leftrightarrow \mathit{prefix}(M_1) \subseteq \mathit{prefix}(M_2)$ which is equivalent to say that $M_1 \sqsubseteq M_2 \Leftrightarrow \forall r_1 \in M_1 \exists r_2 \in M_2. r_1 \leq_{\mathit{pref}} r_2$. We define the *lub* $\bigsqcup$ and the *glb* $\bigsqcap$ of $\mathbb{M}$ analogously. It can be proven that $(\mathbb{P}, \subseteq) \underset{\mathit{maximal}}{\overset{\mathit{prefix}}{\leftrightarrows}} (\mathbb{M}, \sqsubseteq)$ is an *order-preserving isomorphism*, so $(\mathbb{M}, \sqsubseteq, \bigsqcup, \bigsqcap, \mathbf{M}, \{\epsilon\})$ is also a complete lattice.

Although this second representation is very convenient for technical definitions, it is not very suited for examples. For instance, different maximal traces have frequently (significant) common prefixes; hence, some parts have to be written twice and, more important, it can be difficult to visualize the repetition (obfuscating the comprehension). Thus, in our examples we will use an equivalent representation in terms of prefix trees. Namely, we will use trees with (non root) nodes labeled with conditional states. Given $P \in \mathbb{P}$, $\mathit{tree}(P)$ builds the prefix tree of P, obtained by combining all the sequences that have a prefix in common in the same path. Let $\mathbb{T} \coloneqq \{\mathit{tree}(P) \mid P \in \mathbb{P}\}$. The inverse of *tree* is the function $\mathit{path} \colon \mathbb{T} \to \mathbb{P}$ which returns the set of all possible paths starting from the root. Let $\trianglelefteq$ be the order on $\mathbb{T}$ induced by the order on $\mathbb{P}$: $T_1 \trianglelefteq T_2 \Leftrightarrow path(T_1) \subseteq path(T_2)$. We define the *lub* and *glb* of $\mathbb{T}$ in a similar way. It can be proven that $(\mathbb{P}, \subseteq) \overset{path}{\underset{tree}{\leftrightarrows}} (\mathbb{T}, \trianglelefteq)$ is an order-preserving isomorphism, so also $(\mathbb{T}, \trianglelefteq)$ is a complete lattice. In the sequel we will use the representation which is most convenient in each case.

3.2 Fixpoint Denotations of Programs

The technical core of our semantics definition is the agent semantics evaluation function (Definition 3.16, page 15) which, given an agent A and an interpretation $\mathcal{I}$ (for the process symbols of A), builds the maximal conditional traces associated to A. To define it, we need first to introduce some auxiliary semantic functions.

Definition 3.7 (Propagation Operator) *Let $r \in \mathbf{M}$ and $c \in \mathbf{C}$. We define by structural induction the* propagation *of c in r, written $r{\downarrow_c}$, as $\boxtimes{\downarrow_c} = \boxtimes$, $\epsilon{\downarrow_c} = \epsilon$ and*

$$((\eta^+,\eta^-) \rightarrowtail d \cdot r'){\downarrow_c} = \begin{cases} (\eta^+ \otimes c, \eta^-) \rightarrowtail d \otimes c \cdot (r'{\downarrow_c}) & \text{if } c \gg (\eta^+,\eta^-),\, d \otimes c \neq \mathit{ff} \\ (\eta^+ \otimes c, \eta^-) \rightarrowtail \mathit{ff} \cdot \boxtimes & \text{if } c \gg (\eta^+,\eta^-),\, d \otimes c = \mathit{ff} \end{cases}$$

$$(stutt(\eta^-) \cdot r'){\downarrow_c} = stutt(\eta^-) \cdot (r'{\downarrow_c}) \qquad \text{if } \forall c^- \in \eta^-.\, c \nvdash c^-$$

We abuse notation and denote $R{\downarrow_h}$ the point-wise extension of ${\downarrow_h}$ to sets of conditional traces.

This operator is used in the semantics of constructs that add new information to traces. By definition, the *propagation operator* $\downarrow$ is a partial function $\mathbf{M}\times\mathbf{C} \to \mathbf{M}$ that instantiates a conditional trace with a given constraint and checks the consistency of the new information with the conditional states in the trace. This information needs to be propagated also to the successive (i.e., future) conditional states in order to maintain the monotonicity of the store.

Example 3.8

Given the conditional trace $r := (true, \varnothing) \rightarrowtail x > 10 \cdot (true, \varnothing) \rightarrowtail x > 20 \cdot \boxtimes$, the propagation of $y > 2$ in r is $(y > 2, \varnothing) \rightarrowtail x > 10 \wedge y > 2 \cdot (y > 2, \varnothing) \rightarrowtail x > 20 \wedge y > 2 \cdot \boxtimes$.

For $r' := (true, \{y > 0\}) \rightarrowtail true \cdot \boxtimes$ the propagation of $y > 2$ in r' is not defined since $y > 2 \not\gg (true, \{y > 0\})$.

Finally, given the conditional trace $r'' := (true, \varnothing) \rightarrowtail y < 0 \cdot \boxtimes$, the propagation of $y > 2$ in r'' produces the conditional trace $(y > 2, \varnothing) \rightarrowtail false \cdot \boxtimes$ since $y > 2 \gg (true, \varnothing)$ and $y < 0 \wedge y > 2 = false$.

Note that the consecutive propagation of two constraints $(r{\downarrow_c}){\downarrow_{c'}}$ is equivalent to $r{\downarrow_{(c \otimes c')}}$ (as stated formally in Lemma A.2).

Definition 3.9 (c-compatible) *$r \in \mathbf{M}$ is said to be* compatible *w.r.t. $c \in \mathbf{C}$ (r is c-compatible) if, for each $(\eta^+,\eta^-) \rightarrowtail d$ in r, $c \gg (\eta^+,\eta^-)$, and for each $stutt(\eta^-)$ in r, $c \nvdash c^-$ for all $c^- \in \eta^-$.*

When r is not c-compatible w.r.t. c, the store c is in contradiction with a condition of some conditional state of r and then $r{\downarrow_c}$ is not defined.

The following auxiliary operator is used in the definition of the semantics of the parallel construct. Intuitively, the parallel operator combines (with maximal parallelism) the information coming from two conditional traces. It checks the satisfiability of the conditions and the consistency of the resulting stores.

Definition 3.10 (Parallel composition) *The* parallel composition *partial operator* $\bar{\|} : \mathbf{M} \times \mathbf{M} \to \mathbf{M}$ *is the commutative closure of the following partial operation defined by structural induction as:* $r \mathbin{\bar{\|}} \epsilon := r$, $r \mathbin{\bar{\|}} \boxtimes := r$ *and*

$$(stutt(\eta_1^-) \cdot r_1') \mathbin{\bar{\|}} (stutt(\eta_2^-) \cdot r_2') := stutt(\eta_1^- \cup \eta_2^-) \cdot (r_1' \mathbin{\bar{\|}} r_2')$$

Moreover, if $\eta_1 \otimes \eta_2$ *is valid,* r_1' *is* c_2*-compatible and* r_2' *is* c_1*-compatible, then*

$$(\eta_1 \rightarrowtail c_1 \cdot r_1') \mathbin{\bar{\|}} (\eta_2 \rightarrowtail c_2 \cdot r_2') := \begin{cases} \eta_1 \otimes \eta_2 \rightarrowtail c_1 \otimes c_2 \cdot \\ \qquad ((r_1' \downarrow_{c_2}) \mathbin{\bar{\|}} (r_2' \downarrow_{c_1})) & \text{if } c_1 \otimes c_2 \neq f\!f \\ \eta_1 \otimes \eta_2 \rightarrowtail f\!f \cdot \boxtimes & \text{if } c_1 \otimes c_2 = f\!f, \end{cases}$$

Finally, if $\forall c^- \in \eta_2^- . \eta_1^+ \nvdash c^-$ *and* r_2' *is* c_1*-compatible, then*

$$((\eta_1^+, \eta_1^-) \rightarrowtail c_1 \cdot r_1') \mathbin{\bar{\|}} (stutt(\eta_2^-) \cdot r_2') := (\eta_1^+, \eta_1^- \cup \eta_2^-) \rightarrowtail c_1 \cdot (r_1' \mathbin{\bar{\|}} (r_2' \downarrow_{c_1}))$$

Clearly, by definition, $\bar{\|}$ is commutative. Moreover, because of $\otimes$ associativity, $\bar{\|}$ is also associative. It is worth noting that, if one of the traces is not compatible with the propagated constraint, then the parallel composition is not defined.

Example 3.11

Consider $r_1 := (true, \varnothing) \rightarrowtail y > 2 \cdot (y > 2, \varnothing) \rightarrowtail y > 2 \cdot \boxtimes$ and $r_2 := (z = 1, \varnothing) \rightarrowtail z = 1 \cdot \boxtimes$. Since r_1 and r_2 do not share variables, the compatibility checks always succeed and then $r_1 \mathbin{\bar{\|}} r_2 = (z = 1, \varnothing) \rightarrowtail y > 2 \wedge z = 1 \cdot (y > 2 \wedge z = 1, \varnothing) \rightarrowtail y > 2 \wedge z = 1 \cdot \boxtimes$.

Consider now $r_3 := stutt(\{y > 0\}) \cdot (y > 0, \varnothing) \rightarrowtail y > 0 \wedge z = 3 \cdot \boxtimes$. Traces r_1 and r_3 share the variable y, and it can be seen that the information regarding y in the two traces is consistent, thus $r_1 \mathbin{\bar{\|}} r_3 = (true, \{y > 0\}) \rightarrowtail y > 2 \cdot (y > 2, \varnothing) \rightarrowtail y > 2 \wedge z = 3 \cdot \boxtimes$.

Finally, consider $r_4 := (true, \varnothing) \rightarrowtail true \cdot (true, \{y > 0\}) \rightarrowtail true \cdot \boxtimes$. This trace, in the second time instant, requires that the constraint $y > 0$ cannot be entailed by the current store. However, the trace r_1 states, at the same time instant, that $y > 2$. This is the reason because $r_1 \mathbin{\bar{\|}} r_4$ is not defined.

Note that $\downarrow$ distributes over $\bar{\|}$, in the sense that $(r_1 \mathbin{\bar{\|}} r_2)\downarrow_c = (r_1\downarrow_c) \mathbin{\bar{\|}} (r_2\downarrow_c)$ (as stated formally in Lemma A.3).

The last auxiliary operator that we need is the hiding operator $\bar{\exists} : Var \times \mathbf{M} \to \mathbf{M}$ which, intuitively, hides the information regarding a given variable in a conditional trace.

Definition 3.12 (Hiding operator) *Given* $r \in \mathbf{M}$ *and* $x \in \mathcal{V}$*, we define the* hiding of x in r*, written* $\bar{\exists}_x r$*, by structural induction:*

$$\bar{\exists}_x r := \begin{cases} \exists_x \big((\eta^+, \eta^-) \rightarrowtail c\big) \cdot \bar{\exists}_x r' & \text{if } r = (\eta^+, \eta^-) \rightarrowtail c \cdot r' \\ stutt(\exists_x \eta^-) \cdot \bar{\exists}_x r' & \text{if } r = stutt(\eta^-) \cdot r' \\ r & \text{if } r = \epsilon \text{ or } r = \boxtimes \end{cases}$$

We distinguish two special classes of conditional traces.

Definition 3.13 (Self-sufficient and x-self-sufficient conditional trace) *$r \in \mathbf{M}$ is said to be* self-sufficient *if the first condition is $(tt, \varnothing)$ and, for each $t_i = \eta_i \rightarrowtail c_i$ and $t_{i+1} = \eta_{i+1} \rightarrowtail c_{i+1}$, $c_i \Vdash \eta_{i+1}$ (each store satisfies the successive condition).*

Moreover, r is self-sufficient w.r.t. $x \in \mathcal{V}$ *(*x-self-sufficient*) if $\exists_{Var \smallsetminus \{x\}} r$ is self-sufficient.*

Definition 3.13 is stronger than Definition 3.4 since the latter does not require satisfiability but just consistency of the store w.r.t. the conditions. Thus this definition demands that for self-sufficient conditional traces, no additional information (from other agents) is needed in order to *complete* the computation. In an x-self-sufficient conditional trace the same happens but only considering information about variable x.

Example 3.14

The conditional trace r_1 of Example 3.5 is not self-sufficient since $y = 0 \nVdash x > 2$.

Now consider a variation where we add the information $x = 4$ to the stores, namely $r_2 := (true, \varnothing) \rightarrowtail y = 0 \wedge x = 4 \cdot (x > 2, \varnothing) \rightarrowtail y = 0 \wedge z = 3 \wedge x = 4 \cdot \boxtimes$. It is easy to see that r_2 is a self-sufficient conditional trace, essentially because we add enough information in the first store to satisfy the second condition, i.e., $y = 0 \wedge x = 4 \Vdash (x > 2, \varnothing)$.

Moreover, r_2 is also x-self-sufficient since $\exists_{Var \smallsetminus \{x\}} r_2 = (true, \varnothing) \rightarrowtail x = 4 \cdot (x > 2, \varnothing) \rightarrowtail x = 4 \cdot \boxtimes$, which is a self-sufficient trace.

3.2.1 Interpretations

Now we introduce the notion of interpretation, which is used to give meaning to process calls, by associating to each process symbol a set of (maximal) conditional traces "modulo variance".

Definition 3.15 (Interpretations) *Let $\mathbb{MGC} := \{p(x) \mid p \in \Pi, \vec{x}$ are distinct variables$\}$. We call any $\pi \in \mathbb{MGC}$* most general call.

Two functions $I, J \colon \mathbb{MGC} \to \mathbb{M}$ are variants, *denoted by $I \cong J$, if for each $\pi \in \mathbb{MGC}$ there exists a variable renaming ρ such that $I(\pi)\rho = J(\pi\rho)$.*

An interpretation *is a function $\mathcal{I} \colon \mathbb{MGC} \to \mathbb{M}$ modulo variance*[4]*.*

The semantic domain $\mathbb{I}_\Pi$ (or simply $\mathbb{I}$ when clear from the context) is the set of all interpretations ordered by the pointwise extension of $\sqsubseteq$ (which by an abuse of notation we denote by $\sqsubseteq$).

Essentially, we define the semantics of each predicate in Π over formal parameters whose names are actually irrelevant. It is important to note that $\mathbb{MGC}$ has the same cardinality of Π (and is thus finite) and therefore each interpretation is a finite collection (of possibly infinite elements).

In the following, any $\mathcal{I} \in \mathbb{I}_\Pi$ is implicitly considered as an arbitrary function $\mathbb{MGC} \to \mathbb{M}$ obtained by choosing an arbitrary representative of the elements of $\mathcal{I}$ generated by $\cong$. Actually, in the following, all the operators that we use on $\mathbb{I}_\Pi$ are also independent of the choice of the representative. Therefore, we can define any operator on $\mathbb{I}_\Pi$ in terms of its counterpart defined on functions $\mathbb{MGC} \to \mathbb{M}$. Moreover, the application of an interpretation $\mathcal{I}$ to a most general call π, denoted by $\mathcal{I}(\pi)$, is the application $I(\pi)$ of any representative I of $\mathcal{I}$ which is

[4]i.e., a family of elements of $\mathbb{M}$ indexed by $\mathbb{MGC}$ modulo variance.

defined exactly on π. For example, if $\mathcal{I} = (\lambda p(x,y).\,\{(true,\varnothing) \rightarrowtail x = y\})/_{\cong}$ then $\mathcal{I}(p(u,v)) = \{(true,\varnothing) \rightarrowtail u = v\}$.

The partial order on $\mathbb{I}$ formalizes the evolution of the computation process. $(\mathbb{I}, \sqsubseteq)$ is a complete lattice and its least upper bound and greatest lower bound are the pointwise extension of $\bigsqcup$ and $\bigsqcap$, respectively. The bottom element is $\bot_{\mathbb{I}} := \lambda\pi.\,\{\epsilon\}$.

Since $\mathbb{MGC}$ is finite, we will often explicitly write interpretations by cases, like

$$\mathcal{I} := \begin{cases} \pi_1 \mapsto T_1 \\ \vdots \\ \pi_n \mapsto T_n \end{cases} \qquad \text{representing} \qquad \begin{array}{c} \mathcal{I}(\pi_1) := T_1 \\ \vdots \\ \mathcal{I}(\pi_n) := T_n \end{array}$$

3.2.2 Semantics Evaluation Function of Agents

We are finally ready to define the evaluation function of an agent A w.r.t. an interpretation $\mathcal{I}$, which computes the set of (maximal) conditional traces associated to the agent A. It is important to note that the computation does not depend on an initial store. Instead, the weakest (most general) condition for each agent is (computed and) accumulated in the traces.

Definition 3.16 (Semantics Evaluation Function for Agents) *Given $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$ and $\mathcal{I} \in \mathbb{I}_{\Pi}$, we define the* semantics evaluation $\mathcal{A}[\![A]\!]_{\mathcal{I}}$ *by structural induction as follows.*

$$\mathcal{A}[\![\mathsf{skip}]\!]_{\mathcal{I}} := \boxtimes \tag{3.2}$$

$$\mathcal{A}[\![\mathsf{tell}(c)]\!]_{\mathcal{I}} := (tt,\varnothing) \rightarrowtail c \cdot \boxtimes \tag{3.3}$$

$$\mathcal{A}[\![A \parallel B]\!]_{\mathcal{I}} := \bigsqcup\big\{r_A \;\bar{\parallel}\; r_B \mid r_A \in \mathcal{A}[\![A]\!]_{\mathcal{I}},\, r_B \in \mathcal{A}[\![B]\!]_{\mathcal{I}}\big\} \tag{3.4}$$

$$\mathcal{A}[\![\exists x\, A]\!]_{\mathcal{I}} := \bigsqcup\big\{\,\bar{\exists}_x\, r \mid r \in \mathcal{A}[\![A]\!]_{\mathcal{I}},\, r \textit{ is x-self-sufficient}\big\} \tag{3.5}$$

$$\mathcal{A}[\![p(x)]\!]_{\mathcal{I}} := (tt,\varnothing) \rightarrowtail tt \cdot \mathcal{I}(p(x)) \tag{3.6}$$

$$\mathcal{A}[\![\sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i]\!]_{\mathcal{I}} := \mathit{lfp}_{\mathbb{M}}\, \lambda R.\Big(\mathit{stutt}(\{c_1,\dots,c_n\}) \cdot R \sqcup \tag{3.7}$$

$$\bigsqcup\big\{(c_i,\varnothing) \rightarrowtail c_i \cdot (r{\downarrow_{c_i}}) \,\big|\, 1 \le i \le n,\, r \in \mathcal{A}[\![A_i]\!]_{\mathcal{I}},\, r\;\, c_i\textit{-compatible}\big\}\Big) \tag{3.8}$$

$\mathcal{A}[\![\mathsf{now}\; c \;\mathsf{then}\; A \;\mathsf{else}\; B]\!]_{\mathcal{I}} :=$

$$\{(c,\varnothing) \rightarrowtail c \cdot \boxtimes \mid \boxtimes \in \mathcal{A}[\![A]\!]_{\mathcal{I}}\} \sqcup \tag{3.9a}$$

$$\begin{aligned} \bigsqcup\{(\eta^+ \otimes c, \eta^-) \rightarrowtail d \otimes c \cdot (r{\downarrow_c}) \mid (\eta^+,\eta^-) \rightarrowtail d \cdot r \in \mathcal{A}[\![A]\!]_{\mathcal{I}}, \\ d \otimes c \neq \mathit{ff},\, \forall c^- \in \eta^-.\, \eta^+ \otimes c \nvdash c^-,\, r\;\, c\textit{-compatible}\} \sqcup \end{aligned} \tag{3.9b}$$

$$\begin{aligned} \bigsqcup\{(\eta^+ \otimes c, \eta^-) \rightarrowtail \mathit{ff} \cdot \boxtimes \mid (\eta^+,\eta^-) \rightarrowtail d \cdot r \in \mathcal{A}[\![A]\!]_{\mathcal{I}}, \\ d \otimes c = \mathit{ff},\, \forall c^- \in \eta^-.\, \eta^+ \otimes c \nvdash c^-,\, r\;\, c\textit{-compatible}\} \sqcup \end{aligned} \tag{3.9c}$$

$$\begin{aligned} \bigsqcup\{(c, \eta^-) \rightarrowtail c \cdot (r{\downarrow_c}) \mid \mathit{stutt}(\eta^-) \cdot r \in \mathcal{A}[\![A]\!]_{\mathcal{I}}, \\ \forall c^- \in \eta^-.\, c \nvdash c^-,\, r\;\, c\textit{-compatible}\} \sqcup \end{aligned} \tag{3.9d}$$

$$\bigsqcup\{(tt, \{c\}) \rightarrowtail tt \cdot \boxtimes \mid \boxtimes \in \mathcal{A}[\![B]\!]_{\mathcal{I}}\} \sqcup \tag{3.9e}$$

$$\bigsqcup\{(\eta^+, \eta^- \cup \{c\}) \rightarrowtail d \cdot r \mid (\eta^+,\eta^-) \rightarrowtail d \cdot r \in \mathcal{A}[\![B]\!]_{\mathcal{I}},\, \eta^+ \nvdash c\} \sqcup \tag{3.9f}$$

$$\bigsqcup\{(tt, \eta^- \cup \{c\}) \rightarrowtail tt \cdot r \mid \mathit{stutt}(\eta^-) \cdot r \in \mathcal{A}[\![B]\!]_{\mathcal{I}}\} \tag{3.9g}$$

We now explain in detail each (non immediate) case of the definition.

(3.3) For the $\mathsf{tell}(c)$ agent we have condition $(tt, \varnothing)$ since c must be added to the store in any case (in the current time instant). Next the computation terminates (with the end of process symbol $\boxtimes$).

(3.5) The hiding construct must hide the information about x from all traces that cannot be altered by the presence of external information about x, thus the hiding operation is applied just to x-self-sufficient conditional traces.

(3.6) The semantics of process call $p(x)$ simply delays by one time instant the traces for $p(x)$ in interpretation $\mathcal{I}$ by prefixing them with $(tt, \varnothing) \rightarrowtail tt$.

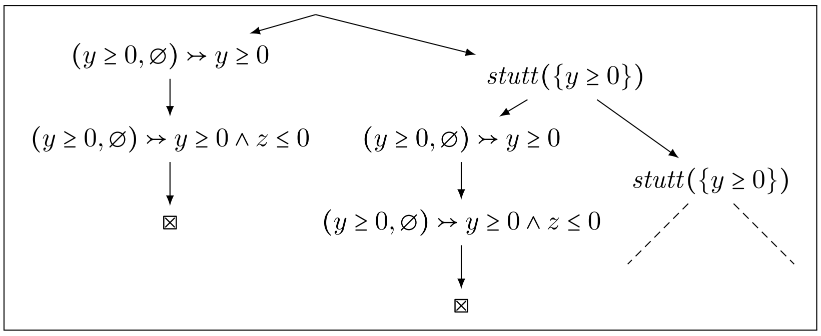
(3.8) The semantics for the non-deterministic choice collects, for each guard c_i, a conditional trace of the form $(c_i, \varnothing) \rightarrowtail c_i \cdot (r{\downarrow_{c_i}})$. This trace requires that c_i has to be satisfied by the current store (positive part of the condition in the first state). Then the constraint c_i is propagated to the trace r (the continuation of the computation, which belongs to the semantics of A_i). Note that the requirement of c_i-compatibility ensures that $r{\downarrow_{c_i}}$ is defined.

Furthermore, we collect the stuttering traces, which correspond to the case when the computation suspends. Traces representing this situation are of the form $\mathit{stutt}(\{c_1, \dots, c_n\}) \cdot r$ where r is, recursively, an element of the semantics of the choice agent.

(3.9) The definition for the conditional agent $\mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B$ is in principle similar to the previous case. However, since the now construct must be instantaneous, in order to correctly model the timing of the agent we have seven cases depending on the possible forms of the first conditional state of the semantics of A (respectively B), on the value of the resulting store (ff or not) and on the fact that the guard c is satisfied or not in the current time instant.

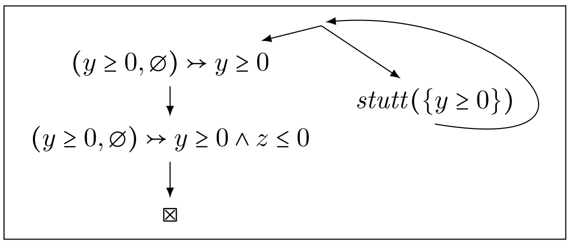
(3.9a)–(3.9d) represent the case in which the guard c is satisfied by the current store. In this case, the agent now must behave instantaneously as A. For this reason, we distinguish four different cases corresponding to the possible form of conditional traces associated to A. In particular, (3.9a) corresponds to the case when the computation of A ends, thus also the computation of the conditional must be ended. In (3.9b), the information added (in one step) by A is compatible with the condition and with the rest of the computation and, moreover, does not produce ff when merged with the current store d. (3.9c) stops the conditional trace since the the information produced by A added to the current store produces the inconsistent store ff. Finally, (3.9d) corresponds to the case when A suspends.

(3.9e)–(3.9g) consider the cases when c is not entailed by the current store. In this situation, the agent now must behave instantaneously as B, and the definition follows the same reasoning as for (3.9a), (3.9b) and (3.9d). The main difference is that instead of adding c to the positive condition in the first conditional state, we add $\{c\}$ to the negative condition.



$(y \geq 0, \varnothing) \rightarrowtail y \geq 0$
$stutt(\{y \geq 0\})$
$(y \geq 0, \varnothing) \rightarrowtail y \geq 0 \wedge z \leq 0$
$(y \geq 0, \varnothing) \rightarrowtail y \geq 0$
$stutt(\{y \geq 0\})$
$\boxtimes$
$(y \geq 0, \varnothing) \rightarrowtail y \geq 0 \wedge z \leq 0$
$\boxtimes$

Figure 2: Tree representation of $\mathcal{A}[\![A_4]\!]_{\mathcal{I}}$ of Example 3.17



$(y \geq 0, \varnothing) \rightarrowtail y \geq 0$
$stutt(\{y \geq 0\})$
$(y \geq 0, \varnothing) \rightarrowtail y \geq 0 \wedge z \leq 0$
$\boxtimes$

Figure 3: Graph representation of $\mathcal{A}[\![A_4]\!]_{\mathcal{I}}$ of Example 3.17

Example 3.17

In this example we compute the semantics for the *tccp* agent $A_1 := A_2 \parallel A_3$ where

$$\begin{aligned}
A_2 &:= \mathsf{tell}(y = 2) \parallel \mathsf{tell}(x = y)\\
A_3 &:= \mathsf{ask}(true) \rightarrow \mathsf{now}\ (x = 0)\ \mathsf{then}\ \mathsf{tell}(z > 0)\ \mathsf{else}\ A_4\\
A_4 &:= \mathsf{ask}(y \geq 0) \rightarrow \mathsf{tell}(z \leq 0)
\end{aligned}$$

Since there are no process calls, the interpretation $\mathcal{I}$ is irrelevant for the result. We start by computing the semantics for A_4. Let $r := (y \geq 0, \varnothing) \rightarrowtail y \geq 0 \cdot (y \geq 0, \varnothing) \rightarrowtail y \geq 0 \wedge z \leq 0 \cdot \boxtimes$. Then

$$\begin{aligned}
\mathcal{A}[\![A_4]\!]_{\mathcal{I}} &= lfp_{\mathbb{M}}(\lambda R.\, \{r\} \sqcup stutt(\{y \geq 0\}) \cdot R)\\
&= \{r,\ stutt(\{y \geq 0\}) \cdot r,\ stutt(\{y \geq 0\}) \cdot stutt(\{y \geq 0\}) \cdot r,\ \ldots\}\\
&= \left\{\left(stutt(\{y \geq 0\})\right)^n \cdot r \,\middle|\, n \in \mathbb{N}\right\} \sqcup \{stutt(\{y \geq 0\}) \cdots stutt(\{y \geq 0\}) \cdots\}
\end{aligned}$$

Figure 2 graphically represents $\mathcal{A}[\![A_4]\!]_{\mathcal{I}}$, which consists of a trace for the case in which the guard is satisfied, and a set of traces for the case in which it suspends. As it can be observed, the tree in Figure 2 consists of an infinite replication of the same pattern. We can depict such infinite trees as finite graphs, as in Figure 3. The back-loop arc is just a graphical shortcut which represents the (infinite) tree that is obtained by unrolling the loop. It is important to note that nodes reached by a path of length 2 (via the back-loop arc) have to be considered as a single arc, thus corresponding just to a 1 time instant delay. With the semantics of A_4, we compute $\mathcal{A}[\![A_3]\!]_{\mathcal{I}} = \{r_1, r_2\} \cup R$ where

$$r_1 := \{(true, \varnothing) \rightarrowtail true \cdot (x = 0, \varnothing) \rightarrowtail x = 0 \wedge z > 0 \cdot \boxtimes$$

$$r_2 := (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdot (y \geq 0, \{x = 0\}) \rightarrowtail y \geq 0 \cdot (y \geq 0, \varnothing) \rightarrowtail y \geq 0 \wedge z \leq 0 \cdot \boxtimes$$
$$R := (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdot (\mathit{true}, \{y \geq 0,\ x = 0\}) \rightarrowtail \mathit{true} \cdot \mathcal{A}[\![A_4]\!]_{\mathcal{I}}$$

All the traces of $\mathcal{A}[\![A_3]\!]_{\mathcal{I}}$ start with the conditional store $(\mathit{true}, \varnothing) \rightarrowtail \mathit{true}$ corresponding to the ask agent with guard true. The trace r_1 corresponds to the case when (in the current time instant) the guard $x = 0$ is satisfied; the trace r_2 corresponds to $x = 0$ not satisfied and $y \geq 0$ satisfied; while we have R when none is satisfied and A_4 is executed.

Now we can compute the semantics for A_1 by parallel composition of $\mathcal{A}[\![A_3]\!]_{\mathcal{I}}$ with $\mathcal{A}[\![A_2]\!]_{\mathcal{I}} = \{(\mathit{true}, \varnothing) \rightarrowtail (y = 2 \wedge x = y) \cdot \boxtimes\}$. We recall that when the parallel operator composes two conditional traces, it checks their compatibility, thus if two traces have inconsistent stores or conditions then nothing is produced.

$$\begin{aligned}\mathcal{A}[\![A_1]\!]_{\mathcal{I}} = \{&(\mathit{true}, \varnothing) \rightarrowtail (y = 2 \wedge x = y) \cdot (y = 2 \wedge x = y, \{x = 0\}) \rightarrowtail (y = 2 \wedge x = y) \cdot \\ &(y = 2 \wedge x = y, \varnothing) \rightarrowtail (y = 2 \wedge x = y \wedge z \leq 0) \cdot \boxtimes\}\end{aligned}$$

In detail, the combination of the first conditional trace in $\mathcal{A}[\![A_3]\!]_{\mathcal{I}}$ (let us call it r_1) and the conditional trace in $\mathcal{A}[\![A_2]\!]_{\mathcal{I}}$ does not produce contributes since the constraint $y = 2$, when propagated to the second component of r_1 is in contradiction with the positive part of the condition ($y = 2 \wedge x = y \wedge x = 0 \equiv \mathit{false}$). Indeed, $(\mathit{true}, \varnothing) \rightarrowtail (y = 2 \wedge x = y) \cdot ((x = 0, \varnothing) \rightarrowtail x = 0 \wedge z > 0 \cdot \boxtimes)\downarrow_{(y=2\wedge x=y)} = (\mathit{true}, \varnothing) \rightarrowtail (y = 2 \wedge x = y) \cdot (\mathit{false}, \varnothing) \rightarrowtail \mathit{false}$ is not a trace since $(\mathit{false}, \varnothing)$ is not a valid condition.

The combination of the set of traces corresponding to the suspension of the agent (third trace in the semantics of A_3) and the tell agent, also produces no trace. The definition Definition 3.16 prescribes to compute $(\mathit{true}, \varnothing) \rightarrowtail y = 2 \wedge x = y \cdot \bigsqcup\{((\mathit{true}, \{y \geq 0,\ x = 0\}) \rightarrowtail \mathit{true} \cdot r')\downarrow_{(y=2\wedge x=y)} \mid r' \in \mathcal{A}[\![A_4]\!]_{\mathcal{I}}\}$, which is empty, since $y = 2 \wedge x = y \not\gg (\mathit{true}, \{y \geq 0,\ x = 0\})$ because $y = 2 \wedge x = y \Rightarrow y \geq 0$. These traces would correspond to the suspension of the agent ask, and this can happen only when $y \geq 0$ is not satisfied, but the first component of the parallel agent tells $y = 2$, thus $y \geq 0$ is satisfied. Therefore, only the combination of the trace r_2 in $\mathcal{A}[\![A_3]\!]_{\mathcal{I}}$ and the trace of $\mathcal{A}[\![A_2]\!]_{\mathcal{I}}$ produces a conditional trace.

Example 3.18

Consider the agent $A := \exists x\, A_1$, where A_1 is defined in Example 3.17.

In order to apply the definition of the evaluation function for the hiding agent, let us first check that the trace $r \in \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$ is x-self-sufficient. To this end, we hide all the information that is not concerned with x and the resulting conditional trace

$$(\mathit{true}, \varnothing) \rightarrowtail x = 2 \cdot (x = 2, \{x = 0\}) \rightarrowtail x = 2 \cdot (x = 2, \varnothing) \rightarrowtail x = 2 \cdot \boxtimes$$

is indeed x-self-sufficient. Therefore, as Definition 3.16 states, the trace $\bar{\exists}_x\, r$ belongs to the semantics of A. Namely

$$\mathcal{A}[\![A]\!]_{\mathcal{I}} = \{(\mathit{true}, \varnothing) \rightarrowtail y = 2 \cdot (y = 2, \varnothing) \rightarrowtail y = 2 \cdot (y = 2, \varnothing) \rightarrowtail (y = 2 \wedge z \leq 0) \cdot \boxtimes\}$$

Let us now consider the agent $A' := \mathsf{tell}(x \leq 0) \parallel A$. We have

$$\begin{aligned}\mathcal{A}[\![A']\!]_{\mathcal{I}} = \{&(\mathit{true}, \varnothing) \rightarrowtail y = 2 \wedge x \leq 0 \cdot \\ &(y = 2 \wedge x \leq 0, \varnothing) \rightarrowtail y = 2 \wedge x \leq 0 \cdot \\ &(y = 2 \wedge x \leq 0, \varnothing) \rightarrowtail (y = 2 \wedge x \leq 0 \wedge z \leq 0) \cdot \boxtimes\}\end{aligned}$$

It is easy to see that the information on the variable x added by the tell agent does not affect the *internal* execution of the agent A, as expected.

There are some technical decisions that ensure the correctness of the defined semantics. Due to the partial nature of the constraint system, the combination of the hiding operator with non-determinism can make the language behavior non-monotonic (i.e., non-condensing). As already mentioned, this is the reason because for all non-monotonic languages of the *ccp* paradigm, the compositional semantics that have been written either avoid non-monotonic constructs or restrict their use. Let us show now that we are able to handle the following example, which is an adaptation to *tccp* of the one used in [9, 20] to illustrate the non-monotonicity problem.

Example 3.19

Consider the non-monotonic agent

$$A \coloneqq \mathsf{ask}(x = 1) \to \mathsf{tell}(\mathit{true}) + \mathsf{ask}(\mathit{true}) \to \mathsf{tell}(y = 2).$$

It is easy to see that for the initial store *true* just the second branch can be taken, whereas for the (greater) initial store $x = 1$, the two branches can be executed. Since there are no process calls, for any interpretation $\mathcal{I}$, $\mathcal{A}[\![A]\!]_{\mathcal{I}} = \{r_1, r_2\}$, where

$$r_1 \coloneqq (x = 1, \varnothing) \rightarrowtail x = 1 \cdot (x = 1, \varnothing) \rightarrowtail x = 1 \cdot \boxtimes$$
$$r_2 \coloneqq (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdot (\mathit{true}, \varnothing) \rightarrowtail y = 2 \cdot \boxtimes$$

We have two possible traces depending on whether the initial store is strong enough to entail $x = 1$ or not.

Now, let us consider $A' \coloneqq \mathsf{tell}(x = 1) \parallel \exists x\, A$. Since

$$\bar{\exists}_{\mathit{Var} \smallsetminus \{x\}}\, r_1 = (x = 1, \varnothing) \rightarrowtail x = 1 \cdot (x = 1, \varnothing) \rightarrowtail x = 1 \cdot \boxtimes$$
$$\bar{\exists}_{\mathit{Var} \smallsetminus \{x\}}\, r_2 = (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdot (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdot \boxtimes$$

only r_2 is x-self-sufficient and, by Definition 3.16,

$$\mathcal{A}[\![\exists x\, A]\!]_{\mathcal{I}} = \{(\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdot (\mathit{true}, \varnothing) \rightarrowtail y = 2 \cdot \boxtimes\}.$$

By composing we have

$$\mathcal{A}[\![A']\!]_{\mathcal{I}} = \{(\mathit{true}, \varnothing) \rightarrowtail x = 1 \cdot (x = 1, \varnothing) \rightarrowtail y = 2 \land x = 1 \cdot \boxtimes\}.$$

It is easy to see that the information on the variable x added by the tell agent does not affect the *internal* execution of the agent A, as expected.

In the definition of the propagation operator (Definition 3.7), the propagated information is added not only to the store of the state, but also to the (positive part of the) condition. This means that the positive part of the conditions in a trace contains the information that had to be satisfied up to that computation step, but also the constraints that have been added during computation in the previous time instants. From the computations in the examples above, it may seem that the propagation of the accumulated information in the conditions of the states could be redundant. However, it is necessary in order to have full abstraction w.r.t. the behavior, otherwise we would distinguish agents whose behavior is actually the same as shown in the following example.

Example 3.20

Consider the following two (very similar) agents:

$$A_1 := \mathsf{ask}(x > 2) \to \mathsf{tell}(y = 1) \qquad A_2 := \mathsf{ask}(x > 4) \to \mathsf{tell}(y = 1)$$

We have similar but different semantics. Namely,

$$\begin{aligned}
\mathcal{A}[\![A_1]\!]_{\mathcal{I}} &= \{\big(\mathit{stutt}(\{x > 2\})\big)^n \cdot r_1 \mid n \in \mathbb{N}\} \sqcup \{\mathit{stutt}(\{x > 2\}) \cdots \mathit{stutt}(\{x > 2\}) \cdots\} \\
&\quad r_1 = (x > 2, \varnothing) \rightarrowtail \mathit{true} \cdot (x > 2, \varnothing) \rightarrowtail y = 1 \cdot \boxtimes \\
\mathcal{A}[\![A_2]\!]_{\mathcal{I}} &= \{\big(\mathit{stutt}(\{x > 4\})\big)^n \cdot r_2 \mid n \in \mathbb{N}\} \sqcup \{\mathit{stutt}(\{x > 4\}) \cdots \mathit{stutt}(\{x > 4\}) \cdots\} \\
&\quad r_2 = (x > 4, \varnothing) \rightarrowtail \mathit{true} \cdot (x > 4, \varnothing) \rightarrowtail y = 1 \cdot \boxtimes
\end{aligned}$$

However, consider now the following two agents, which embed A_1 and A_2 in the same context:

$$A_1' := \mathsf{tell}(x = 7) \parallel \mathsf{ask}(\mathit{true}) \to A_1 \qquad A_2' := \mathsf{tell}(x = 7) \parallel \mathsf{ask}(\mathit{true}) \to A_2$$

Then, the two traces corresponding to the satisfaction of the guards are, respectively:

$$r_3 = (\mathit{true}, \varnothing) \rightarrowtail x = 7 \cdot r_1{\downarrow_{(x=7)}} \qquad r_4 = (\mathit{true}, \varnothing) \rightarrowtail x = 7 \cdot r_2{\downarrow_{(x=7)}}$$

Since the propagated constraint is stronger than the guards in both the agents, the resulting compositions are the same. In fact, thanks to the accumulation of the store in the condition, we do not distinguish them:

$$\begin{aligned}
r_1{\downarrow_{(x=7)}} = r_2{\downarrow_{(x=7)}} = (\mathit{true}, \varnothing) \rightarrowtail x = 7 \cdot \\
(x = 7, \varnothing) \rightarrowtail x = 7 \cdot (x = 7, \varnothing) \rightarrowtail x = 7 \wedge y = 1 \cdot \boxtimes
\end{aligned}$$

If the constraint $x = 7$ is not added to the condition, but only to the store of the state, then we have two different conditional traces for these two agents.

3.2.3 Fixpoint Denotations of Process Declarations

Now we can finally define the semantics for a set of process declarations D.

Definition 3.21 (Fixpoint semantics) *Given $D \in \mathbb{D}_{\mathbf{C}}^{\Pi}$, we define $\mathcal{D}[\![D]\!] : \mathbb{I} \to \mathbb{I}$ as*

$$\mathcal{D}[\![D]\!]_{\mathcal{I}}(p(x)) := \bigsqcup \{\mathcal{A}[\![A]\!]_{\mathcal{I}} \mid p(\vec{x}) \coloneq A \in D\}.$$

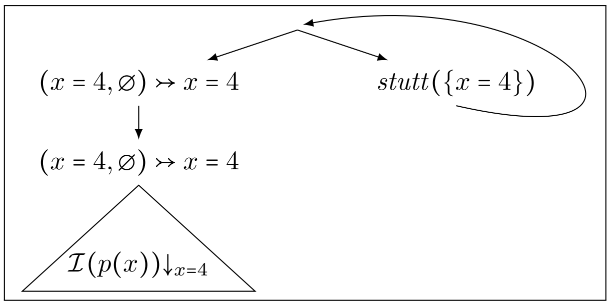
The fixpoint denotation *of D is $\mathcal{F}[\![D]\!] := \mathit{lfp}(\mathcal{D}[\![D]\!]) = \mathcal{D}[\![D]\!]{\uparrow}\omega$.*

We denote with $\approx_{\mathcal{F}}$ the equivalence relation on $\mathbb{D}_{\mathbf{C}}^{\Pi}$ induced by $\mathcal{F}$. Namely, $D_1 \approx_{\mathcal{F}} D_2 \Leftrightarrow \mathcal{F}[\![D_1]\!] = \mathcal{F}[\![D_2]\!]$.

The semantics of a tccp *program $D \,.\, A$ is $\mathcal{P}[\![D \,.\, A]\!] := \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$.*

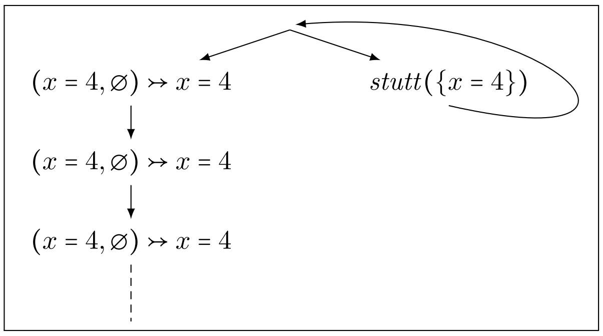
$\mathcal{F}[\![D]\!]$ is well defined since $\mathcal{D}[\![D]\!]$ is continuous (as stated formally in Lemma A.5).

Let us show how the semantics for a set of process declarations is computed by means of some examples.



$(x = 4, \varnothing) \rightarrowtail x = 4$
$stutt(\{x = 4\})$
$(x = 4, \varnothing) \rightarrowtail x = 4$
$\mathcal{I}(p(x))\downarrow_{x=4}$

Figure 4: Graph representation for $\mathcal{A}[\![A]\!]_{\mathcal{I}}$ of Example 3.22



$(x = 4, \varnothing) \rightarrowtail x = 4$
$stutt(\{x = 4\})$
$(x = 4, \varnothing) \rightarrowtail x = 4$
$(x = 4, \varnothing) \rightarrowtail x = 4$

Figure 5: Graph representation of the fixpoint $\mathcal{F}[\![D]\!]$ for the Example 3.22

Example 3.22

Let $D := \{p(x) :- A\}$ where $A := \mathsf{ask}(x = 4) \to p(x)$. First we need to compute, for each $\mathcal{I} \in \mathbb{I}$, the evaluation of the body of the process declaration. Namely,

$$\begin{aligned}\mathcal{A}[\![A]\!]_{\mathcal{I}} = &\Big\{\big(\, stutt(\{x = 4\})\big)^n \cdot \bar{r} \cdot s \,\Big|\, n \in \mathbb{N},\, s \in \mathcal{I}(p(x))\Big\} \sqcup \\ &\{stutt(\{x = 4\}) \cdots stutt(\{x = 4\}) \cdots\}\end{aligned}$$

where $\bar{r} := (x = 4, \varnothing) \rightarrowtail x = 4 \cdot (x = 4, \varnothing) \rightarrowtail x = 4$. It is worth noticing that the second conditional state of $\bar{r}$ corresponds to the delay that is introduced each time that a process call is run. $\mathcal{A}[\![A]\!]_{\mathcal{I}}$ is graphically represented in Figure 4.

The iterates of $\mathcal{D}[\![D]\!]$ are

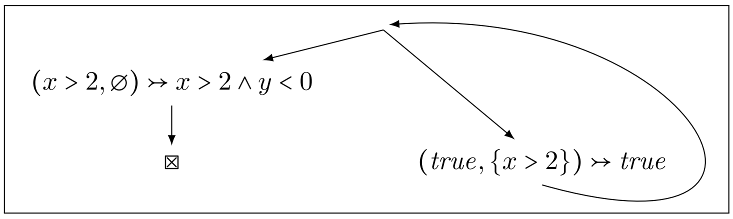
$$\mathcal{D}[\![D]\!]{\uparrow}1 = \begin{cases} p(x) \mapsto & \{(stutt(\{x = 4\}))^n \cdot \bar{r} \,|\, n \in \mathbb{N}\} \sqcup \\ & \{stutt(\{x = 4\}) \cdots stutt(\{x = 4\}) \cdots\} \end{cases}$$

$$\mathcal{D}[\![D]\!]{\uparrow}2 = \begin{cases} p(x) \mapsto & \{(stutt(\{x = 4\}))^n \cdot \bar{r} \cdot \bar{r} \,|\, n \in \mathbb{N}\} \sqcup \\ & \{stutt(\{x = 4\}) \cdots stutt(\{x = 4\}) \cdots\} \end{cases}$$

$$\vdots$$

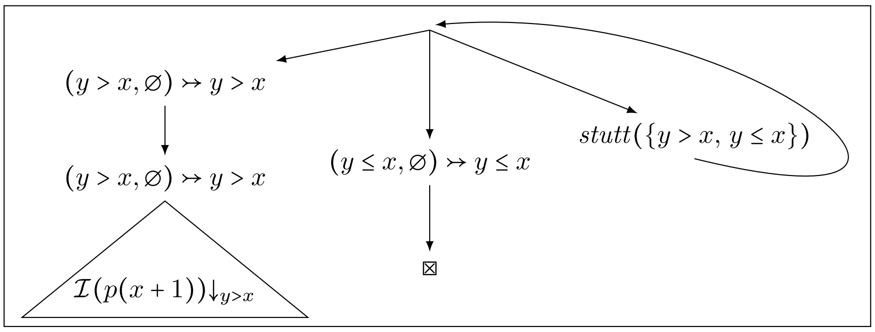
$$\mathcal{D}[\![D]\!]{\uparrow}\omega = \begin{cases} p(x) \mapsto & \{(stutt(\{x = 4\}))^n \cdot \bar{r} \cdots \bar{r} \cdots \,|\, n \in \mathbb{N}\} \sqcup \\ & \{stutt(\{x = 4\}) \cdots stutt(\{x = 4\}) \cdots\} \end{cases}$$

The limit $\mathcal{F}[\![D]\!](p(x)) = (\mathcal{D}[\![D]\!]{\uparrow}\omega)(p(x))$ is graphically represented in Figure 5. Note that the application of the propagation operator to the previous iterates removes all the stuttering sequences, and this is the reason because just the first sequence remains.



$(x > 2, \varnothing) \rightarrowtail x > 2 \wedge y < 0$
$\boxtimes$
$(true, \{x > 2\}) \rightarrowtail true$

Figure 6: Graph representation of the fixpoint $\mathcal{F}[\![D]\!]$ of Example 3.23



$(y > x, \varnothing) \rightarrowtail y > x$
$(y > x, \varnothing) \rightarrowtail y > x$
$\mathcal{I}(p(x+1))\!\downarrow_{y>x}$
$(y \le x, \varnothing) \rightarrowtail y \le x$
$\boxtimes$
$stutt(\{y > x,\ y \le x\})$

Figure 7: Graph representation for $\mathcal{A}[\![A]\!]_{\mathcal{I}}$ in Example 3.24

Example 3.23

Let $D := \{q(x,y) :\!- A\}$ where

$$A := \mathsf{now}\ (x > 2)\ \mathsf{then}\ \mathsf{tell}(y < 0)\ \mathsf{else}\ q(x,y)$$

Then

$$\mathcal{D}[\![D]\!]\!\uparrow\!1 = \Big\{q(x,y) \mapsto \{\bar{r},\ (true, \{x > 2\}) \rightarrowtail true\}$$

$$\mathcal{D}[\![D]\!]\!\uparrow\!2 = \begin{cases} q(x,y) \mapsto & \{\bar{r},\ (true, \{x > 2\}) \rightarrowtail true \cdot \bar{r}, \\ & (true, \{x > 2\}) \rightarrowtail true \cdot (true, \{x > 2\}) \rightarrowtail true\} \end{cases}$$

$$\vdots$$

$$\mathcal{F}[\![D]\!] = \begin{cases} q(x,y) \mapsto & \Big\{\big((true, \{x > 2\}) \rightarrowtail true\big)^n \cdot \bar{r} \,\Big|\, n \in \mathbb{N}\Big\} \\ & \sqcup\{(true, \{x > 2\}) \rightarrowtail true \cdots (true, \{x > 2\}) \rightarrowtail true \cdots\} \end{cases}$$

where $\bar{r} := (x > 2, \varnothing) \rightarrowtail x > 2 \wedge y < 0 \cdot \boxtimes$. $\mathcal{F}[\![D]\!](q(x,y))$ is graphically represented in Figure 6.

Example 3.24

Let $D := \{p(x,y) :\!- A\}$ where

$$A := \mathsf{ask}(y > x) \to p(x+1, y) + \mathsf{ask}(y \le x) \to \mathsf{skip}$$

As usually done in *tccp* community, we assume that we can use expressions of the form $x + 1$ directly in the arguments of a process call. We can simulate this behavior by writing $\exists x'\,(\mathsf{tell}(x' = x + 1) \parallel p(x', y))$ instead of $p(x+1, y)$ (but introducing a delay of one time unit). We have

$$\mathcal{A}[\![A]\!]_{\mathcal{I}} = \{(y > x, \varnothing) \rightarrowtail y > x \cdot (y > x, \varnothing) \rightarrowtail y > x \cdot r \ \mid\ r \in \mathcal{I}(p(x+1, y))\} \sqcup$$

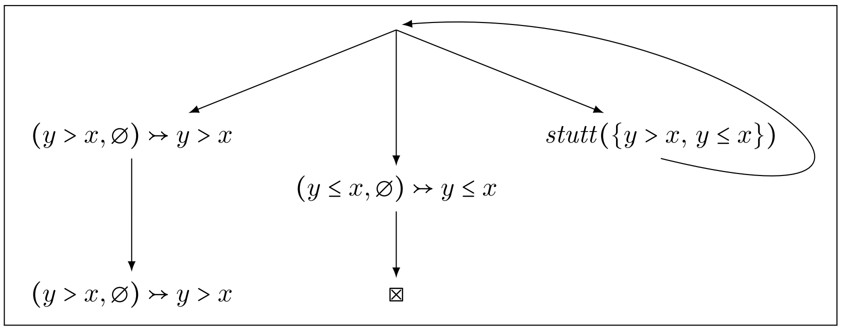

$(y > x, \varnothing) \rightarrowtail y > x$
$stutt(\{y > x,\ y \le x\})$
$(y \le x, \varnothing) \rightarrowtail y \le x$
$(y > x, \varnothing) \rightarrowtail y > x$
$\boxtimes$

Figure 8: Graph representation of $\mathcal{D}[\![D]\!]{\uparrow}1$ in Example 3.24

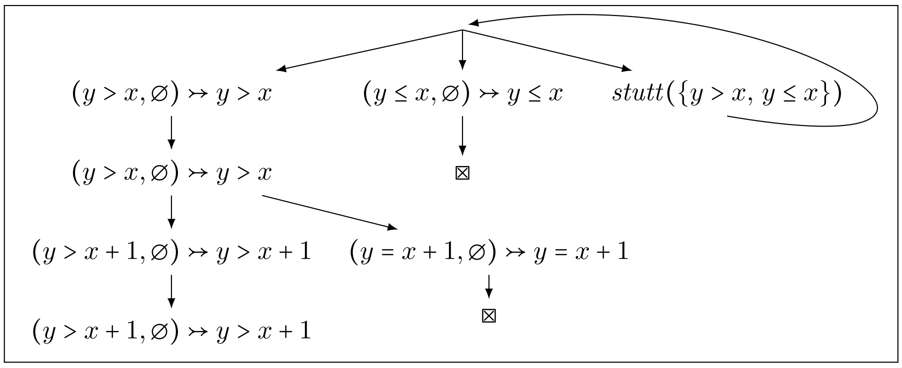

$(y > x, \varnothing) \rightarrowtail y > x$
$(y \le x, \varnothing) \rightarrowtail y \le x$
$stutt(\{y > x,\ y \le x\})$
$(y > x, \varnothing) \rightarrowtail y > x$
$\boxtimes$
$(y > x+1, \varnothing) \rightarrowtail y > x+1$
$(y = x+1, \varnothing) \rightarrowtail y = x+1$
$(y > x+1, \varnothing) \rightarrowtail y > x+1$
$\boxtimes$

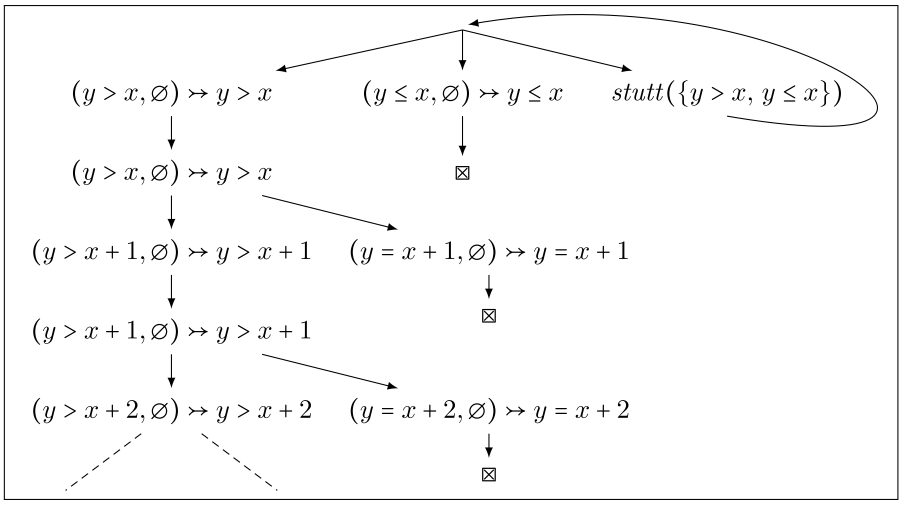
Figure 9: Graph representation of f $\mathcal{D}[\![D]\!]{\uparrow}2$ in Example 3.24

$$\begin{aligned}
&\{(y \le x, \varnothing) \rightarrowtail y \le x \cdot \boxtimes\} \sqcup \\
&\{(stutt(\{y > x,\ y \le x\}))^n \cdot (y > x, \varnothing) \rightarrowtail y > x \cdot \\
&\quad (y > x, \varnothing) \rightarrowtail y > x \cdot r \mid n \in \mathbb{N},\ r \in \mathcal{I}(p(x+1, y))\} \sqcup \\
&\{(stutt(\{y > x,\ y \le x\}))^n \cdot (y \le x, \varnothing) \rightarrowtail y \le x \cdot \boxtimes \ \mid\ n \in \mathbb{N}\} \sqcup \\
&\{stutt(\{y > x,\ y \le x\}) \cdots stutt(\{y > x,\ y \le x\}) \cdots\}
\end{aligned}$$

which is graphically shown in Figure 7. For this agent, we have three branches, one for each condition of the choice and one corresponding to the stuttering possibility. The first iteration of $\mathcal{D}[\![D]\!]$ is

$$\mathcal{D}[\![D]\!]{\uparrow}1 = \begin{cases}
p(x,y) \mapsto & \{(y > x, \varnothing) \rightarrowtail y > x \cdot (y > x, \varnothing) \rightarrowtail y > x\} \sqcup \\
& \{(y \le x, \varnothing) \rightarrowtail y \le x \cdot \boxtimes\} \sqcup \\
& \{(stutt(\{y > x,\ y \le x\}))^n \cdot (y > x, \varnothing) \rightarrowtail y > x \cdot \\
& \quad (y > x, \varnothing) \rightarrowtail y > x \ \mid\ n \in \mathbb{N}\} \sqcup \\
& \{(stutt(\{y > x,\ y \le x\}))^n \cdot (y \le x, \varnothing) \rightarrowtail y \le x \cdot \boxtimes \mid n \in \mathbb{N}\} \sqcup \\
& \{stutt(\{y > x,\ y \le x\}) \cdots stutt(\{y > x,\ y \le x\}) \cdots\}
\end{cases}$$

which is graphically represented in Figure 8. Figure 9 represents the second iteration, whereas Figure 10 is the limit of the fixpoint computation. In the semantics, it can be easily observed that the process stops in one time instant when $y \le x$ and in $1 + 2(y - x)$ time instants otherwise. This process can be combined with other processes to be used as a kind of *timer* since it forces the time passing during a given time interval.



$(y > x, \varnothing) \rightarrowtail y > x$
$(y \le x, \varnothing) \rightarrowtail y \le x$
$stutt(\{y > x,\, y \le x\})$
$(y > x, \varnothing) \rightarrowtail y > x$
⊠
$(y > x+1, \varnothing) \rightarrowtail y > x+1$
$(y = x+1, \varnothing) \rightarrowtail y = x+1$
⊠
$(y > x+1, \varnothing) \rightarrowtail y > x+1$
$(y > x+2, \varnothing) \rightarrowtail y > x+2$
$(y = x+2, \varnothing) \rightarrowtail y = x+2$
⊠

Figure 10: Graph representation of $\mathcal{F}[\![D]\!]$ in Example 3.24.

Example 3.25

Consider the following process declaration, presented in [17], which models a subsystem of a microwave controller. The underlying constraint system is the well-known Herbrand constraint system [8].

$$\begin{aligned}
&\mathit{microwave}(\mathit{Door}, \mathit{Button}, \mathit{Error}) \coloneq \exists D\, \exists B\, \exists E \\
&\quad \big(\, \mathsf{tell}(\mathit{Error} = [\_ \mid E]) \parallel \mathsf{tell}(\mathit{Door} = [\_ \mid D]) \parallel \mathsf{tell}(\mathit{Button} = [\_ \mid B]) \\
&\quad\;\; \parallel \big(\mathsf{now}(\mathit{Door} = [\mathit{open} \mid D] \wedge \mathit{Button} = [\mathit{on} \mid B]) \\
&\quad\;\;\;\; \mathsf{then}\ \exists E1\, \mathsf{tell}(E = [1 \mid E1]) \parallel \exists B1\, \mathsf{tell}(B = [\mathit{off} \mid B1]) \\
&\quad\;\;\;\; \mathsf{else}\ \exists E1\, \mathsf{tell}(E = [0 \mid E1])\big) \\
&\quad\;\; \parallel \mathit{microwave}(D, B, E)\big)
\end{aligned}$$

This process declaration detects if the door is open while the microwave is turned on. In that case, it forces that in the next time instant the microwave is turned-off and it emits an error signal (1); otherwise, the agent emits a signal of no error (0). Due to the monotonicity of the store, streams are used to model *imperative-style* variables [10]. In the example, the streams *Error*, *Door* and *Button* store the values that these *variables* get along the computation. The first three tell agents link the future values of the streams with the *future streams* E, D and B. Then, when it is detected a possible *risk* (characterized by the guard of the now agent), the microwave is turned off and an *error* signal is emitted (by the then branch of the conditional agent). The final recursive call restarts the same control at the next time instant.

The fixpoint semantics $\mathcal{F}(\mathit{microwave}(D, B, E))$ is graphically represented in Figure 11, where:

$$\mathit{risk}_k \coloneq \exists_D\, \exists_B (\mathit{Door} = [\underbrace{\mathit{open} \mid \dots}_{k \text{ times}} \mid D] \wedge \mathit{Button} = [\mathit{on} \mid \underbrace{\mathit{off} \mid \mathit{on} \mid \dots}_{k-1 \text{ times}} \mid B])$$

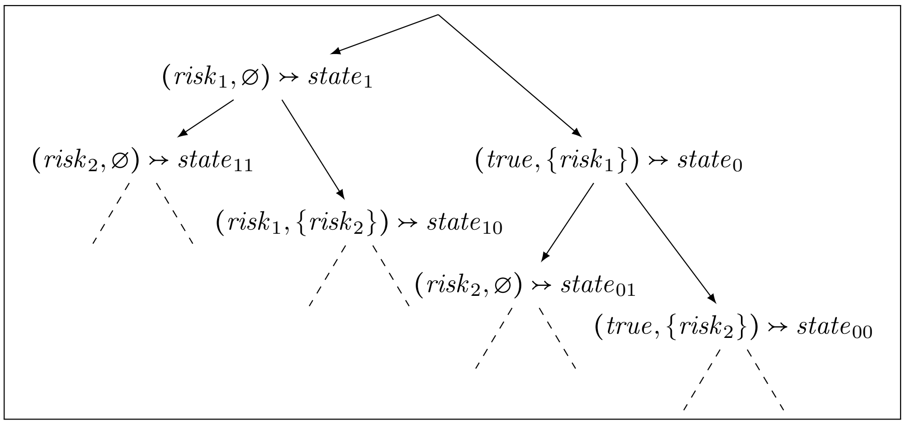

$(risk_1, \varnothing) \rightarrowtail state_1$
$(risk_2, \varnothing) \rightarrowtail state_{11}$
$(risk_1, \{risk_2\}) \rightarrowtail state_{10}$
$(true, \{risk_1\}) \rightarrowtail state_0$
$(risk_2, \varnothing) \rightarrowtail state_{01}$
$(true, \{risk_2\}) \rightarrowtail state_{00}$

Figure 11: Tree representation of $\mathcal{F}[\![D]\!]$ in Example 3.25.

$$\begin{aligned} state_{b_1 \dots b_n} := \exists_{E1}\, \exists_{D}\, \exists_{B1}(&Error = [\_ \mid b_1 \mid \dots \mid b_n \mid E1] \wedge Door = [\_ \mid D] \wedge \\ &Button = [\_ \mid \underbrace{on \mid off \mid \dots}_{\Sigma_{i=1}^{n} b_i \text{ times}} \mid B1]) \end{aligned}$$

We have coded the indices of stores in the conditional states with a binary number in order to make the figure readable. It is worth noticing that the stores labeled with $state_b$ where the last digit of b is 1 correspond to states where an error is emitted.

All the conditional sequences in the semantics of this process are infinite sequences. This is consistent with the fact that we are modeling a process that is intended to be active forever, checking whether the risky situation holds. It is worth noticing that this kind of processes can be handled only if the semantics is able to capture infinite computations, which is one of the main features of our proposal.

3.2.4 Full abstraction of $\mathcal{F}$ semantics

Our semantics $\mathcal{F}$ is fully abstract w.r.t. the operational behavior. This subsection is dedicated to prove this fact. First, we need to define an auxiliary operator which, taken a "real" initial store c, instantiates the "hypothetical" states of a conditional trace r producing the corresponding "real" behavioral timed trace. Intuitively, this operator works by consistently adding to each conditional state the information given by the initial store c, discarding those sequences which falsify conditions.

Definition 3.26 (Instantiation operator) *The* instantiation *operator* $\Downarrow : \mathbf{M} \times \mathbf{C} \to \mathbf{C}^*$ *is a partial function defined by structural induction as:* $\epsilon\Downarrow_c := \epsilon$*; otherwise* $r\Downarrow_{ff} := ff$*; otherwise* $\boxtimes\Downarrow_c := c$*,*

$$\begin{aligned} (stutt(\eta^-) \cdot r')\Downarrow_c &:= c && \text{if } \forall c^- \in \eta^- .\, c \nvdash c^- \\ (\eta \rightarrowtail d \cdot r')\Downarrow_c &:= c \cdot (r'\Downarrow_{c \otimes d}) && \text{if } c \Vdash \eta \text{ and } (c \otimes d) \neq ff \end{aligned}$$

We abuse notation by denoting with $R\Downarrow_c$ *the extension of* $\Downarrow_c$ *to* $\mathbb{M}$*:* $R\Downarrow_c := \{r\Downarrow_c \mid r \in R \text{ and } r\Downarrow_c \text{ is defined}\}$*.*

The instantiation operator is consistent w.r.t. the propagation operator (Definition 3.7), in the sense that, for any c' that entails c, $r{\Downarrow_c} = (r{\downarrow_{c'}}){\Downarrow_c}$ (as stated formally in Lemma A.6). Moreover, the instantiation operator $\Downarrow$ "distributes" over the parallel composition operator $\bar{\|}$ (Definition 3.10) (as stated formally in Lemma A.8).

The key result to prove correctness of $\mathcal{F}$ w.r.t. $\approx_{ss}$ is the following theorem which shows that the behavior of a program P can be determined by instantiation of the semantics $\mathcal{P}[\![P]\!]$.

Theorem 3.27 *For each program P and each $c \in \mathbf{C}$, prefix$(\mathcal{P}[\![P]\!]{\Downarrow_c}) = \mathcal{B}^{ss}[\![P]\!]_c$.*

The following theorem is the key result to prove full abstraction of $\mathcal{F}$ w.r.t. $\approx_{ss}$.

Theorem 3.28 *Let P_1, P_2 be two programs. Then $\mathcal{P}[\![P_1]\!] = \mathcal{P}[\![P_2]\!] \Leftrightarrow \forall c \in \mathbf{C}.\, \mathcal{P}[\![P_1]\!]{\Downarrow_c} = \mathcal{P}[\![P_2]\!]{\Downarrow_c}$.*

Full abstraction follows directly from Theorems 3.27 and 3.28 and Proposition 3.29.

Proposition 3.29 *Let D_1, $D_2 \in \mathbb{D}^{\Pi}_{\mathbf{C}}$. Then $D_1 \approx_{\mathcal{F}} D_2 \Leftrightarrow \forall A \in \mathbb{A}^{\Pi}_{\mathbf{C}}.\, \mathcal{P}[\![D_1 \,.\, A]\!] = \mathcal{P}[\![D_2 \,.\, A]\!]$.*

Corollary 3.30 (Correctness and full abstraction of $\mathcal{F}$) *Let D_1, $D_2 \in \mathbb{D}^{\Pi}_{\mathbf{C}}$. Then $D_1 \approx_{ss} D_2$ if and only if $D_1 \approx_{\mathcal{F}} D_2$.*

4 Big-step semantics

In the literature, many authors (like [10]) call *observables* all the abstractions of the behavioral timed traces (including behavioral timed traces themselves as the degenerate identity abstraction). Moreover, they typically use this same name for the collection of all observables of a program.

Many other authors use the term *observable (property)* for an abstraction function ϕ which, when applied to the set of traces of a program, delivers the observations of interest and the observation of program P is just the application of ϕ to the traces of P.

We prefer to use the latter nomenclature and, in the sequel, we call $\phi(\mathcal{B}^{ss}[\![Q]\!])$ *observable behavior of a program Q w.r.t. observable ϕ* (or simply *ϕ-observable behavior of Q*) and we denote it by $\mathcal{B}^{\phi}[\![Q]\!]$.

The observable property which is usually considered in papers dealing with semantics of *ccp* languages (e.g. see [13]) is the one that collects the input/output pairs of terminating computations, including deadlocked ones. Indeed, using the (original version of the) transition system of Definition 2.1, [10] defines the notion of *input-output observables* as $\mathcal{O}^{io}(A) := \{\langle c_0, c_n\rangle \mid \langle A_0, c_0\rangle \to^* \langle A_n, c_n\rangle \not\to\}$. In this definition, there is an implicit reference to a set of declarations D. Since in the sequel we need to state some formal results for two (different) sets of declarations, we use the explicit notation $\mathcal{O}^{io}[\![D \,.\, A]\!]$ instead of $\mathcal{O}^{io}(A)$.

As we already mentioned, in *tccp* also infinite computations *must* be considered, for example when we are modeling reactive systems. Thus, we do not restrict only to terminating computations. However, we nevertheless want to be able to distinguish if an input-output pair refers to a finite or infinite computation. Thus, we will use *input-output pairs with associated termination mode* of the form $\langle c_0, \mathit{mode}(c_n)\rangle$, where $c_0 \in \mathbf{C}$ is the input store of the computation, $c_n \in \mathbf{C}$ is the output store (which is the *lub* of the stores of the computation) and *mode* is either *fin* or *inf* for finite or infinite computations, respectively.

Definition 4.1 (Input-output behavior of programs) *Given $c, c' \in \mathbf{C}$ such that $c \vdash c'$, an* input-output pair with termination mode *is either $\langle c, \mathit{fin}(c')\rangle$ or $\langle c, \mathit{inf}(c')\rangle$.*

We denote by **IO** *the set of input-output pairs with termination mode and by $\mathbb{IO}$ the domain $\wp(\mathbf{IO})$, ordered by set inclusion.*

Let $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $A_0 \in \mathbb{A}^{\Pi}_{\mathbf{C}}$. The input-output behavior *of the program $D \,.\, A_0$ is defined as*

$$\begin{aligned}\mathcal{B}^{io}[\![D \,.\, A_0]\!] :=& \big\{\langle c_0, \mathit{fin}(c_n)\rangle \,\big|\, c_0 \in \mathbf{C}, \langle A_0, c_0\rangle \to^* \langle A_n, c_n\rangle \not\to\big\} \cup \\ & \big\{\langle c_0, \mathit{inf}(\otimes_{i\geq 0} c_i)\rangle \,\big|\, c_0 \in \mathbf{C}, \langle A_0, c_0\rangle \to \cdots \to \langle A_i, c_i\rangle \to \cdots\big\}\end{aligned}$$

where $\to$ is the transition relation of Figure 1.

We denote by $\approx_{io}$ the equivalence relation between process declarations induced by $\mathcal{B}^{io}$, namely $D_1 \approx_{io} D_2 \Leftrightarrow \forall A \in \mathbb{A}^{\Pi}_{\mathbf{C}}.\ \mathcal{B}^{io}[\![D_1 \,.\, A]\!] = \mathcal{B}^{io}[\![D_2 \,.\, A]\!]$.

We denote by $\pi^{\mathbb{IO}}_F$ the projection which selects just the pairs whose mode is fin and by $\mathbb{IO}_F$ we denote $\pi^{\mathbb{IO}}_F(\mathbb{IO})$.

Moreover we denote by $\mathcal{B}^{io}_F[\![D \,.\, A]\!]$ the finite fragment of $\mathcal{B}^{io}[\![D \,.\, A]\!]$ i.e., $\pi^{\mathbb{IO}}_F(\mathcal{B}^{io}[\![D \,.\, A]\!])$.

Clearly, $(\mathbb{IO}, \subseteq, \bigcup, \bigcap, \mathbf{IO}, \varnothing)$ is a complete lattice.

In the sequel, we define an abstract interpretation [7] of the small-step semantics $\mathcal{P}[\![D \,.\, A]\!]$ (Section 3.2.3) which gives $\mathcal{B}^{io}[\![D \,.\, A]\!]$. Then we prove that the finite fragment of this abstraction (i.e., $\mathcal{B}^{io}_F[\![D \,.\, A]\!]$) is isomorphic to $\mathcal{O}^{io}[\![D \,.\, A]\!]$ (modulo the changes in the definition of the small-step semantics).

As suggested by the abstract interpretation approach, we proceed as follows. First, we define a Galois Insertion $(\mathbb{M}, \sqsubseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{IO}, \subseteq)$ and then we lift it over interpretations $\mathbb{I} \underset{\dot\alpha}{\overset{\dot\gamma}{\leftrightarrows}} [\mathbb{MGC} \to \mathbb{IO}]$ by function composition as $\dot\alpha(f) = \alpha \circ f$. The optimal abstract version of the semantics $\mathcal{D}[\![D]\!]$ is simply obtained as $\mathcal{D}^{\alpha}[\![D]\!] := \dot\alpha \circ \mathcal{D}[\![D]\!] \circ \dot\gamma$. Abstract interpretation theory assures that $\mathcal{F}^{\alpha}[\![D]\!] := \mathit{lfp}(\mathcal{D}^{\alpha}[\![D]\!])$ is the best correct approximation of $\mathcal{F}[\![D]\!]$. Correct because $\alpha(\mathcal{F}[\![D]\!]) \subseteq \mathcal{F}^{\alpha}[\![D]\!]$ and best because it is the minimum (w.r.t. $\subseteq$) of all correct approximations.

4.1 Input-output semantics with infinite outcomes

Now we formally define the Galois Insertion which abstracts conditional traces to input-output pairs with infinite outcomes.

Definition 4.2 (Input-Output abstraction) *Given any $R \in \mathbb{M}$, we define*

$$\begin{aligned}\alpha_{io}(R) :=& \{\langle c_0, \mathit{fin}(c_n)\rangle \mid c_0 \in \mathbf{C}, r \in R, \mathit{last}(r\Downarrow_{c_0}) = c_n\} \cup & (4.1)\\ & \{\langle c_0, \mathit{inf}(\otimes_{i\geq 0} c_i)\rangle \mid c_0 \in \mathbf{C}, r \in R, r\Downarrow_{c_0} = c_0 \ldots c_i \ldots\} \\ \gamma_{io}(P) :=& \bigsqcup\{r \in \mathbf{CT} \mid \langle c_0, \mathit{fin}(c_n)\rangle \in P, \mathit{last}(r\Downarrow_{c_0}) = c_n\} \sqcup & (4.2)\\ & \bigsqcup\{r \in \mathbf{CT} \mid \langle c_0, \mathit{inf}(c)\rangle \in P, r\Downarrow_{c_0} = c_0 \ldots c_i \ldots, c = \otimes_{i\geq 0} c_i\}\end{aligned}$$

We abuse notation and denote with the same symbols the lifting to interpretations ($\alpha_{io}(\mathcal{I}) := \alpha_{io} \circ \mathcal{I}$, $\gamma_{io}(\mathcal{I}^{\alpha}) := \gamma_{io} \circ \mathcal{I}^{\alpha}$).

$(\mathbb{M}, \sqsubseteq, \bigsqcup, \bigsqcap, \mathbf{M}, \{\epsilon\}) \xrightleftharpoons[\alpha_{io}]{\gamma_{io}} (\mathbb{IO}, \subseteq, \bigcup, \bigcap, \mathbf{IO}, \varnothing)$ is a Galois insertion (as stated formally in Lemma A.9).

The input-output behavior of a program is indeed obtainable by abstraction of its (concrete) semantics.

Theorem 4.3 *Let $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$. Then $\alpha_{io}(\mathcal{P}[\![D\,.\,A]\!]) = \mathcal{B}^{io}[\![D\,.\,A]\!]$.*

Now (as anticipated), following the abstract interpretation theory, we define the optimal abstract version of $\mathcal{D}$ as $\mathcal{D}^{io} := \alpha_{io} \circ \mathcal{D} \circ \gamma_{io}$,[5] and thus the best correct approximation w.r.t. α_{io} of the semantic function $\mathcal{F}$ is the least fixpoint of $\mathcal{D}^{io}$, i.e., $\mathcal{F}^{io}[\![D]\!] := \mathit{lfp}(\mathcal{D}^{io}[\![D]\!])$. Unfortunately, $\mathcal{F}^{io}[\![D]\!]$ turns out to be very imprecise, mainly because the information contained in the input-output pairs is not enough to keep the synchronization between parallel processes. This means that sets of declarations with the same behavior $\mathcal{B}^{io}$ could have different $\mathcal{F}^{io}$ semantics, as shown by the following example.

Example 4.4

Consider the two sets of declarations $D_1 := \{d_1, d_2\}$ and $D_2 := \{d_1, d_3\}$ where

$$\begin{aligned}
d_1 &:= p(x,\, y) \coloneq q(x) \parallel \mathsf{ask}(\mathit{true}) \to \mathsf{now}\ x = 2\ \mathsf{then}\ \mathsf{tell}(y = 0)\ \mathsf{else}\ \mathsf{tell}(y = 1)\\
d_2 &:= q(x) \coloneq \mathsf{tell}(x = 2)\\
d_3 &:= q(x) \coloneq \mathsf{ask}(\mathit{true}) \to \mathsf{tell}(x = 2)
\end{aligned}$$

Clearly, D_2 differs from D_1 just because of the delay in adding the constraint $x = 2$ to the store. This difference shows up in the input-output behavior of $p(x,\, y)$. Indeed,

$$\begin{aligned}
\alpha_{io}(\mathcal{P}[\![D_1\,.\,p(x,\, y)]\!]) &= \{\langle c,\, \mathit{fin}(c \wedge x = 2 \wedge y = 0)\rangle \mid c \in \mathbf{L}\}\\
\alpha_{io}(\mathcal{P}[\![D_2\,.\,p(x,\, y)]\!]) &= \{\langle c,\, \mathit{fin}(c \wedge y = 0)\rangle \mid c \in \mathbf{L},\, c \Rightarrow x = 2\} \cup\\
&\quad\ \{\langle c,\, \mathit{fin}(c \wedge x = 2 \wedge y = 1)\rangle \mid c \in \mathbf{L},\, c \not\Rightarrow x = 2\}
\end{aligned}$$

and then (by Theorem 4.3) $D_1 \not\approx_{io} D_2$. However, the abstract fixpoint semantics $\mathcal{F}^{io}$ does not distinguish D_1 from D_2. Indeed,

$$\mathcal{F}^{io}[\![D_1]\!] = \mathcal{F}^{io}[\![D_2]\!] = \begin{cases} q(x) \mapsto \{\langle c,\, \mathit{fin}(c \wedge x = 2)\rangle \mid c \in \mathbf{L}\}\\ p(x,y) \mapsto \{\{\langle c,\, \mathit{fin}(c \wedge y = 0)\rangle \mid c \in \mathbf{L},\, c \Rightarrow x = 2\}\} \cup\\ \qquad\qquad \{\langle c,\, \mathit{fin}(c \wedge x = 2 \wedge y = 1)\rangle \mid c \in \mathbf{L},\, c \not\Rightarrow x = 2\}\end{cases}$$

This example proves that the declarations equivalence induced by fixpoint semantics $\mathcal{F}^{io}$ is not correct w.r.t. $\approx_{io}$ (the input-output behavior declarations equivalence).

Given that $\mathcal{F}^{io}$ is the best possible approximation, this also formally proves that it is not possible to have a correct input-output semantics defined solely on the information provided by the input/output pairs (some more information in denotations is needed). This also formally justifies (a posteriori) why [10] defined $\mathcal{O}^{io}(A)$ as a filter of a more concrete semantics instead of using a direct definition.

[5]A direct (expanded) definition of $\mathcal{D}^{io}$ is not relevant for our present purposes.

4.2 Modeling the input-output semantics of [10]

In this section, we (formally) show that the (original) input-output semantics of *tccp* $\mathcal{O}^{io}[\![D\,.\,A]\!]$ (defined in [10]) is (essentially) isomorphic to $\mathcal{B}^{io}_F[\![D\,.\,A]\!]$ (the finite fragment of the semantics introduced in the previous section).

Theorem 4.5 *Let P_1 and P_2 be two* tccp *programs such that no trace in $\mathcal{P}[\![P_1]\!] \sqcup \mathcal{P}[\![P_2]\!]$ is a failed conditional trace. Then $\mathcal{O}^{io}[\![P_1]\!] = \mathcal{O}^{io}[\![P_2]\!] \iff \pi^{\mathbb{IO}}_F(\alpha_{io}(\mathcal{P}[\![P_1]\!])) = \pi^{\mathbb{IO}}_F(\alpha_{io}(\mathcal{P}[\![P_2]\!]))$.*

This theorem does not hold for *any* pair of *tccp* programs because of the difference in the definition of small-step transition relation $\rightarrow$ (as shown by the following example). Namely, we do not have the same input-output pairs (except for the tag *fin*) only when a program reaches *ff* (along some execution path). This explains why we use the adjective “essentially” when mentioning the isomorphism between $\mathcal{O}^{io}[\![D\,.\,A]\!]$ and $\mathcal{B}^{io}_F[\![D\,.\,A]\!]$.

Example 4.6

Let $P_1 := D\,.\,loop$ and $P_2 := D\,.\,\mathsf{tell}(false)$, where $D := \{loop \coloneq \mathsf{tell}(false) \parallel loop\}$. We have that $\mathcal{B}^{io}_F[\![P_1]\!] = \mathcal{B}^{io}_F[\![P_2]\!] = \{\langle c, fin(false)\rangle\}$ while $\mathcal{O}^{io}[\![P_1]\!] = \varnothing \neq \mathcal{O}^{io}[\![P_2]\!] = \{\langle c, fin(false)\rangle\}$.

The difference is due to the change we made in the definition of the small-step operational semantics. More specifically, in the operational semantics that we use (Definition 2.1), when the store *ff* is reached, the transition relation $\rightarrow$ is not defined. We devised $\rightarrow$ in this way to be conform with the original rationale of the *ccp* paradigm. As a consequence, when a sequence computes *ff*, it is considered as a failure computation with output *ff*. In contrast, in the operational semantics of [10], the transition relation $\rightarrow$ does not consider the *ff* store as a special case and then it is possible to execute an agent on the *ff* store. This means that for finite computations that reach the *ff* store both notions coincide, but for infinite computations which are definitively *ff*, in the operational semantics of [10], the two notions are different.

By considering the special case for *ff*, it is straighforward to modify Definition 4.2 to compute exactly $\mathcal{O}^{io}[\![P]\!]$.

To conclude, it is interesting to note that $\mathcal{B}^{io}_F[\![P]\!]$ can be equivalently obtained by *first* appropriately filtering the conditional traces and *after* applying the abstraction α_{io}. Formally, given $M \in \mathbb{M}$, let $\pi^{\mathbb{IO}}_F(M) := \{r \in M \mid r$ ends with $\boxtimes$ or it contains a stuttering$\}$ and let $\mathbb{M}_F := \pi^{\mathbb{IO}}_F(\mathbb{M})$. Note that this domain correspond to sequences such that the application of the $\Downarrow$ operator produces only finite sequences of stores. It can be proved that the following diagram commutes

$$\begin{tikzcd}
(\mathbb{M}, \sqsubseteq) \arrow[r, "\pi^{\mathbb{M}}_F"] \arrow[d, shift left, two heads, "\alpha_{io}"] & (\mathbb{M}_F, \sqsubseteq) \arrow[d, shift left, two heads, "\alpha_{io}"] \\
(\mathbb{IO}, \sqsubseteq) \arrow[r, "\pi^{\mathbb{IO}}_F"] \arrow[u, shift left, "\gamma_{io}"] & (\mathbb{IO}_F, \sqsubseteq) \arrow[u, shift left, "\gamma_{io}"]
\end{tikzcd}$$

5 Conclusions

In this work, we have presented a small-step semantics that is fully abstract w.r.t. the *tccp* language behavior and that is suitable to be used as the basis of analysis techniques such as abstract diagnosis. The task of defining a compositional fully-abstract semantics for the language has shown to be difficult due to the non-monotonic nature of the language, which is a characteristic shared with other concurrent languages of the *ccp* family.

However, by defining a more elaborated semantic domain (that uses conditions to model hypothetical computations) and a suitable interpretation of the agents' behavior, we have encompassed these difficulties.

To our knowledge, this is the first denotational semantics for a language in the *ccp* family that is compositional and fully abstract and that covers the whole language (including the non monotonic behavior of the languages).

We have also defined a big-step semantics for the language as an abstraction of the small-step one. This semantics collects the *limit* store of (finite and infinite) computations. We proved that its fragment for finite computations is precise enough to recover the original input-output semantics of the language. Moreover, we also proved that it is not possible to have a correct input-output semantics which is defined *solely* on the information provided by the input/output pairs.

A Proofs

In the sequel, to avoid a proliferation of parenthesis, we assume that $\downarrow_c$ and $\Downarrow_c$ have priority over $\cdot$ and $\bar{\|}$.

A.1 Proofs of Section 3

By construction, we can see that the conditional traces computed by $\mathcal{A}$ always satisfy that the store in a given time instant entails the positive condition. Formally,

Property A.1 *Let $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$, $\mathcal{I} \in \mathbb{I}_{\Pi}$ and $r \in \mathcal{A}[\![A]\!]_{\mathcal{I}}$. For each conditional tuple $(\eta^+, \eta^-) \rightarrowtail a$ occurring in r, $a \vdash \eta^+$.*

Proof.
This property is directly verified by (3.8) and (3.9) of Definition 3.16: when a guard is added to the positive condition, it is also added to the correspondent store, and propagated to the subsequent trace.

There exists a relation between the propagation operator $\downarrow$ and the merge $\otimes$ of the constraint system: the consecutive propagation of two constraints $(r\downarrow_c)\downarrow_{c'}$ is equivalent to $r\downarrow_{(c\otimes c')}$.

Lemma A.2 *Let $c, c' \in \mathbf{C}$ and $r \in \mathbf{M}$ such that $(r\downarrow_{c'})\downarrow_c$ is defined. Then $r\downarrow_{(c\otimes c')}$ is defined and $(r\downarrow_{c'})\downarrow_c = r\downarrow_{(c\otimes c')}$.*

Proof.
We proceed by the structural induction on r.

$\underline{r = \epsilon \text{ and } r = \boxtimes}$ Straightforward.

$\boldsymbol{r = (\eta^+, \eta^-) \rightarrowtail d \cdot r'}$ By hypothesis, $(r{\downarrow_{c'}}){\downarrow_c}$ is defined, thus, $c \gg (\eta^+ \otimes c', \eta^-)$ and $(r'{\downarrow_{c'}}){\downarrow_c}$ is defined. It follows directly that $c \otimes c' \gg (\eta^+, \eta^-)$ and, by inductive hypothesis, $(r'{\downarrow_{c \otimes c'}})$ is defined. Thus, $(r{\downarrow_{c\otimes c'}})$ is defined too.

$$\begin{aligned}
(r{\downarrow_{c'}}){\downarrow_c} =& (((\eta^+, \eta^-) \rightarrowtail d \cdot r'){\downarrow_{c'}}){\downarrow_c} \\
& \qquad [\text{by Definition } 3.7] \\
=& ((c' \otimes \eta^+, \eta^-) \rightarrowtail c' \otimes d \cdot r'{\downarrow_{c'}}){\downarrow_c} \\
& \qquad [\text{by Definition } 3.7] \\
=& (c \otimes c' \otimes \eta^+, \eta^-) \rightarrowtail c \otimes c' \otimes d \cdot (r'{\downarrow_{c'}}){\downarrow_c} \\
& \qquad [\text{by Inductive Hypothesis}] \\
=& (c \otimes c' \otimes \eta^+, \eta^-) \rightarrowtail c \otimes c' \otimes d \cdot r'{\downarrow_{c\otimes c'}} \\
=& r{\downarrow_{c \otimes c'}}
\end{aligned}$$

$\boldsymbol{r = \mathit{stutt}(\eta^-) \cdot r'}$ By hypothesis, $(r{\downarrow_{c'}}){\downarrow_c}$ is defined, thus, for all $c^- \in \eta^-$ $c \nvdash c^-$ and $c' \nvdash c^-$. Furthermore, $(r'{\downarrow_{c'}}){\downarrow_c}$ is defined. It follows directly that for all $c^- \in \eta^-$ $c \otimes c' \nvdash c^-$, moreover, by inductive hypothesis, $(r'{\downarrow_{c\otimes c'}})$ is defined. Thus, $(r{\downarrow_{c\otimes c'}})$ is defined too.

$$\begin{aligned}
(r{\downarrow_{c'}}){\downarrow_c} =& ((\mathit{stutt}(\eta^-) \cdot r'){\downarrow_{c'}}){\downarrow_c} \\
& \qquad [\text{by Definition } 3.7] \\
=& (\mathit{stutt}(\eta^-) \cdot r'{\downarrow_{c'}}){\downarrow_c} \\
& \qquad [\text{by Definition } 3.7] \\
=& \mathit{stutt}(\eta^-) \cdot (r'{\downarrow_{c'}}){\downarrow_c} \\
& \qquad [\text{by Inductive Hypothesis}] \\
=& \mathit{stutt}(\eta^-) \cdot r'{\downarrow_{c\otimes c'}} \\
=& r{\downarrow_{c\otimes c'}}
\end{aligned}$$

There exists a relation between the parallel composition and the operator of propagation as stated by the following lemma.

Lemma A.3 *Let $r_1, r_2 \in \mathbf{M}$ and $c \in \mathbf{C}$ such that $r_1{\downarrow_c}$ and $r_2{\downarrow_c}$ are defined. Then $r_1{\downarrow_c} \mathbin{\overline{\|}} r_2{\downarrow_c}$ is defined and $r_1{\downarrow_c} \mathbin{\overline{\|}} r_2{\downarrow_c} = (r_1 \mathbin{\overline{\|}} r_2){\downarrow_c}$.*

Proof.

$\boldsymbol{r_1 = \epsilon}$ **(or $\boldsymbol{r_1 = \boxtimes}$) and any $\boldsymbol{r_2}$** The statement follows directly from Definitions 3.7 and 3.10.

$\boldsymbol{r_1 = (\eta_1^+, \eta_1^-) \rightarrowtail d_1 \cdot r_1'}$ **and** $\boldsymbol{r_2 = (\eta_2^+, \eta_2^-) \rightarrowtail d_2 \cdot r_2'}$ Since $r_1{\downarrow_c}$ and $r_2{\downarrow_c}$ are defined, it follows that $\eta_1 = (\eta_1^+, \eta_1^-)$ and $\eta_2 = (\eta_2^+, \eta_2^-)$ are c-compatible. We have to distinguish two cases.

$\boldsymbol{c \otimes d_1 \neq \mathit{ff}}$ **and** $\boldsymbol{c \otimes d_2 \neq \mathit{ff}}$ By hypothesis, since $r_1{\downarrow_c}$ and $r_2{\downarrow_c}$ are defined, we have that c is consistent with both η_1 and η_2, and therefore with their conjunction $(\eta_1 \otimes \eta_2)$. Furthermore, $r_1'{\downarrow_c}$ and $r_2'{\downarrow_c}$ are defined as well. By inductive hypothesis $(r_1' \mathbin{\overline{\|}} r_2'){\downarrow_c}$ is defined too, thus, we can conclude that also $(r_1 \mathbin{\overline{\|}} r_2){\downarrow_c}$ is defined.

$$\begin{aligned}
r_1{\downarrow_c} \mathbin{\overline{\|}} r_2{\downarrow_c} =& ((\eta_1^+, \eta_1^-) \rightarrowtail d_1 \cdot r_1'){\downarrow_c} \mathbin{\overline{\|}} ((\eta_2^+, \eta_2^-) \rightarrowtail d_2 \cdot r_2'){\downarrow_c} \\
& \qquad [\text{by Definition } 3.7]
\end{aligned}$$

$$\begin{aligned}
&= ((\eta_1^+ \otimes c, \eta_1^-) \rightarrowtail d_1 \otimes c \cdot r_1'{\downarrow_c}) \mathbin{\overline{\parallel}} ((\eta_2^+ \otimes c, \eta_2^-) \rightarrowtail d_2 \otimes c \cdot r_2'{\downarrow_c}) \\
&\qquad [\,\text{by Definition 3.10}\,] \\
&= (\eta_1^+ \otimes \eta_2^+ \otimes c, \eta_1^- \cup \eta_2^-) \rightarrowtail d_1 \otimes d_2 \otimes c \cdot (r_1'{\downarrow_c} \mathbin{\overline{\parallel}} r_2'{\downarrow_c}) \\
&\qquad [\,\text{by Inductive Hypothesis}\,] \\
&= (\eta_1^+ \otimes \eta_2^+ \otimes c, \eta_1^- \cup \eta_2^-) \rightarrowtail d_1 \otimes d_2 \otimes c \cdot (r_1' \mathbin{\overline{\parallel}} r_2'){\downarrow_c} \\
&\qquad [\,\text{by Definition 3.7}\,] \\
&= (r_1 \mathbin{\overline{\parallel}} r_2){\downarrow_c}
\end{aligned}$$

$\underline{\boldsymbol{c \otimes d_1 = \mathit{ff}} \textbf{ or } \boldsymbol{c \otimes d_2 = \mathit{ff}}}$ In this case $r_1{\downarrow_c} \mathbin{\overline{\parallel}} r_2{\downarrow_c}$ reaches the store ff in one step, as also occurs when we compute $(r_1 \mathbin{\overline{\parallel}} r_2){\downarrow_c}$:

$$\begin{aligned}
r_1{\downarrow_c} \mathbin{\overline{\parallel}} r_2{\downarrow_c} &= ((\eta_1^+, \eta_1^-) \rightarrowtail d_1 \cdot r_1'){\downarrow_c} \mathbin{\overline{\parallel}} ((\eta_2^+, \eta_2^-) \rightarrowtail d_2 \cdot r_2'){\downarrow_c} \\
&\qquad [\,\text{by Definition 3.7 and Definition 3.10}\,] \\
&= (\eta_1^+ \otimes \eta_2^+ \otimes c, \eta_1^- \cup \eta_2^-) \rightarrowtail \mathit{ff} \cdot \boxtimes \\
&\qquad [\,\text{by Definition 3.7 and Definition 3.10}\,] \\
&= (r_1 \mathbin{\overline{\parallel}} r_2){\downarrow_c}
\end{aligned}$$

$\underline{\boldsymbol{r_1 = (\eta_1^+, \eta_1^-) \rightarrowtail d_1 \cdot r_1'} \textbf{ and } \boldsymbol{r_2 = \mathit{stutt}(\eta_2^-) \cdot r_2'}}$ By hypothesis, since $r_1{\downarrow_c}$ and $r_2{\downarrow_c}$ are defined, we have that c is consistent with η_1 and does not entail any constraint belonging to η_2^-, and therefore it is consistent with $(\eta_1^+, \eta_1^- \cup \eta_2^-)$. Furthermore, $r_1'{\downarrow_c}$ and $r_2'{\downarrow_c}$ are defined as well. By inductive hypothesis $(r_1' \mathbin{\overline{\parallel}} r_2'){\downarrow_c}$ is defined too, thus, we can conclude that also $(r_1 \mathbin{\overline{\parallel}} r_2){\downarrow_c}$ is defined.

$$\begin{aligned}
r_1{\downarrow_c} \mathbin{\overline{\parallel}} r_2{\downarrow_c} &= ((\eta_1^+, \eta_1^-) \rightarrowtail d_1 \cdot r_1'){\downarrow_c} \mathbin{\overline{\parallel}} (\mathit{stutt}(\eta_2^-) \cdot r_2'){\downarrow_c} \\
&\qquad [\,\text{by Definition 3.7}\,] \\
&= ((\eta_1^+ \otimes c, \eta_1^-) \rightarrowtail d_1 \otimes c \cdot r_1'{\downarrow_c}) \mathbin{\overline{\parallel}} (\mathit{stutt}(\eta_2^-) \cdot r_2'{\downarrow_c}) \\
&\qquad [\,\text{by Definition 3.10}\,] \\
&= (\eta_1^+ \otimes c, \eta_1^- \cup \eta_2^-) \rightarrowtail d_1 \otimes c \cdot (r_1'{\downarrow_c} \mathbin{\overline{\parallel}} r_2'{\downarrow_c}) \\
&\qquad [\,\text{by Inductive Hypothesis}\,] \\
&= (\eta_1^+ \otimes c, \eta_1^- \cup \eta_2^-) \rightarrowtail d_1 \otimes c \cdot (r_1' \mathbin{\overline{\parallel}} r_2'){\downarrow_c} \\
&\qquad [\,\text{by Definition 3.7}\,] \\
&= (r_1 \mathbin{\overline{\parallel}} r_2){\downarrow_c}
\end{aligned}$$

$\underline{\boldsymbol{r_1 = \mathit{stutt}(\eta_1^-) \cdot r_1'} \textbf{ and } \boldsymbol{r_2 = \mathit{stutt}(\eta_2^-) \cdot r_2'}}$ By hypothesis, since $r_1{\downarrow_c}$ and $r_2{\downarrow_c}$ are defined, we have that c does not entail any constraint in $\eta_1^- \cup \eta_2^-$, furthermore $r_1'{\downarrow_c}$ and $r_2'{\downarrow_c}$ are defined as well. By inductive hypothesis $(r_1' \mathbin{\overline{\parallel}} r_2'){\downarrow_c}$ is defined too, thus, we can conclude that also $(r_1 \mathbin{\overline{\parallel}} r_2){\downarrow_c}$ is defined.

$$\begin{aligned}
r_1{\downarrow_c} \mathbin{\overline{\parallel}} r_2{\downarrow_c} &= (\mathit{stutt}(\eta_1^-) \cdot r_1'){\downarrow_c} \mathbin{\overline{\parallel}} (\mathit{stutt}(\eta_2^-) \cdot r_2'){\downarrow_c} \\
&\qquad [\,\text{by Definition 3.7}\,] \\
&= (\mathit{stutt}(\eta_1^-) \cdot r_1'{\downarrow_c}) \mathbin{\overline{\parallel}} (\mathit{stutt}(\eta_2^-) \cdot r_2'{\downarrow_c}) \\
&\qquad [\,\text{by Definition 3.10}\,]
\end{aligned}$$

$$
\begin{aligned}
&= \mathit{stutt}(\eta_1^- \cup \eta_2^-) \cdot (r_1'{\downarrow_c} \mathbin{\overline{\|}} r_2'{\downarrow_c}) \\
&\qquad [\text{by Inductive Hypothesis}] \\
&= \mathit{stutt}(\eta_1^- \cup \eta_2^-) \cdot (r_1' \mathbin{\overline{\|}} r_2'){\downarrow_c} \\
&\qquad [\text{by Definition 3.7}] \\
&= (r_1 \mathbin{\overline{\|}} r_2){\downarrow_c}
\end{aligned}
$$

An important technical result states that the evaluation function for agents $\mathcal{A}$ is closed under context embedding.

Lemma A.4 *Let $A_1, A_2 \in \mathbb{A}_{\mathbf{C}}^{\Pi}$ and $\mathcal{I} \in \mathbb{I}$. Then $\mathcal{A}[\![A_1]\!]_{\mathcal{I}} = \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$ if and only if, for all context $C[\,]$, $\mathcal{A}[\![C[A_1]]\!]_{\mathcal{I}} = \mathcal{A}[\![C[A_2]]\!]_{\mathcal{I}}$.*

Proof.

$\Leftarrow$ Directly holds.

$\Rightarrow$ This implication follows from Definition 3.16. The evaluation function $\mathcal{A}$ is defined by composition of the semantics of its subagents. In particular, the semantics of both, $C[A_1]$ and $C[A_2]$, is computed from the semantics of A_1 and A_2, respectively. Since A_1 and A_2 are equivalent, then also the semantics of $C[A_1]$ and $C[A_2]$ coincide.

Lemma A.5 *For each $A \in \mathbb{A}_{\mathbf{C}}^{\Pi}$ and each $D \in \mathbb{D}_{\mathbf{C}}^{\Pi}$, $\mathcal{A}[\![A]\!]$ and $\mathcal{D}[\![D]\!]$ are continuos.*

Proof.

Consider $A \in \mathbb{A}_{\mathbf{C}}^{\Pi}$ and $D \in \mathbb{D}_{\mathbf{C}}^{\Pi}$, to prove the continuity of $\mathcal{A}[\![A]\!]$, we have to verify two properties: monotonicity and finitarity. The continuity of $\mathcal{D}[\![D]\!]$ follows directly from the continuity of $\mathcal{A}[\![A]\!]$ and from Definition 3.21.

Monotonicity. It is sufficient to show that for each $\mathcal{I}_1, \mathcal{I}_2 \in \mathbb{I}$ and and for each $A \in \mathbb{A}_{\mathbf{C}}^{\Pi}$, $\mathcal{I}_1 \sqsubseteq \mathcal{I}_2 \Rightarrow \mathcal{A}[\![A]\!]_{\mathcal{I}_1} \sqsubseteq \mathcal{A}[\![A]\!]_{\mathcal{I}_2}$. Observe that the only case in which $\mathcal{A}$ depends on the interpretation is the case of the process call.

By definition of $\sqsubseteq$, $\mathcal{I}_1(p(x)) \sqsubseteq \mathcal{I}_2(p(x))$, thus:

$$
\begin{aligned}
\mathcal{A}[\![p(x)]\!]_{\mathcal{I}_1} &= \bigsqcup\{(tt, \varnothing) \rightarrowtail tt \cdot r \mid r \in \mathcal{I}_1(p(x))\} \\
&\sqsubseteq \bigsqcup\{(tt, \varnothing) \rightarrowtail tt \cdot r \mid r \in \mathcal{I}_2(p(x))\} = \mathcal{A}[\![p(x)]\!]_{\mathcal{I}_2}
\end{aligned}
$$

Finitarity. Again, it is sufficient to consider the evaluation function $\mathcal{A}$ for the case of the process call. $\mathcal{A}[\![A]\!]_{\mathcal{I}}$ depends on a finitary subset of $\mathcal{I}$, in particular on the subset regarding $p(x)$ which is a finitary set of conditional traces closed by prefix.

Lemma A.6 *Let $r \in \mathbf{M}$ and $c, c' \in \mathbf{C}$ such that $c \vdash c'$ and $r{\Downarrow_c}$ is defined. Then $(r{\downarrow_{c'}}){\Downarrow_c}$ is defined and $r{\Downarrow_c} = (r{\downarrow_{c'}}){\Downarrow_c}$.*

Proof.

By hypothesis, $r{\Downarrow_c}$ is defined, thus c is compatible with all the conditions occurring in r. Since $c \vdash c'$, it is easy to notice that also c' is compatible with all the conditions occurring in r, thus $r{\downarrow_{c'}}$ is defined. Then, $(r{\downarrow_{c'}}){\Downarrow_c}$ is defined as well. If $c = \mathit{ff}$, by Definition 3.26, $r{\Downarrow_{\mathit{ff}}} = \mathit{ff} = (r{\downarrow_{c'}}){\Downarrow_{\mathit{ff}}}$. Otherwise, if $c \neq \mathit{ff}$, we proceed by induction on the structure of r.

$\boxed{r = \epsilon \text{ and } r = \boxtimes}$ The statement follows directly from Definitions 3.7 and 3.26.

$\boxed{r = (\eta^+, \eta^-) \rightarrowtail d \cdot r'}$ We distinguish three subcases:

1. If $d \otimes c \neq \mathit{ff}$, it follows that $d \otimes c' \neq \mathit{ff}$, thus:

$$\begin{aligned}
(r{\downarrow_{c'}}){\Downarrow_c} &= (((\eta^+, \eta^-) \rightarrowtail d \cdot r'){\downarrow_{c'}}){\Downarrow_c} \\
&\quad [\text{by Definition 3.7}] \\
&= ((\eta^+ \otimes c', \eta^-) \rightarrowtail d \otimes c' \cdot r'{\downarrow_{c'}}){\Downarrow_c} \\
&\quad [\text{by Definition 3.26}] \\
&= c \cdot (r'{\downarrow_{c'}}){\Downarrow_{c \otimes d \otimes c'}} \\
&\quad [\text{by Inductive Hypothesis}] \\
&= c \cdot r'{\Downarrow_{c \otimes d \otimes c'}} \\
&\quad [\text{since } c \vdash c'] \\
&= c \cdot r'{\Downarrow_{c \otimes d}}
\end{aligned}$$

By Definition 3.26, $r{\Downarrow_c} = (\eta^+, \eta^-) \rightarrowtail d \cdot r'{\Downarrow_c} = c \cdot r'{\Downarrow_{c \otimes d}}$, thus $r{\Downarrow_c} = (r{\downarrow_{c'}}){\Downarrow_c}$.

2. If $d \otimes c = \mathit{ff}$ and $d \otimes c' \neq \mathit{ff}$:

$$\begin{aligned}
(r{\downarrow_{c'}}){\Downarrow_c} &= ((\eta^+, \eta^-) \rightarrowtail d \cdot r'{\downarrow_{c'}}){\Downarrow_c} \\
&\quad [\text{by Definition 3.7}] \\
&= ((\eta^+ \otimes c', \eta^-) \rightarrowtail d \otimes c' \cdot r'{\downarrow_{c'}}){\Downarrow_c} \\
&\quad [\text{by Definition 3.26}] \\
&= c \cdot \mathit{ff}
\end{aligned}$$

By Definition 3.26, $r{\Downarrow_c} = ((\eta^+, \eta^-) \rightarrowtail d \cdot r'){\Downarrow_c} = c \cdot \mathit{ff}$, thus $r{\Downarrow_c} = (r{\downarrow_{c'}}){\Downarrow_c}$.

3. If $d \otimes c' = \mathit{ff}$, it follows that $d \otimes c = \mathit{ff}$:

$$\begin{aligned}
(r{\downarrow_{c'}}){\Downarrow_c} &= (((\eta^+, \eta^-) \rightarrowtail d \cdot r'){\downarrow_{c'}}){\Downarrow_c} \\
&\quad [\text{by Definition 3.7}] \\
&= ((\eta^+ \otimes c', \eta^-) \rightarrowtail \mathit{ff} \cdot \boxtimes){\Downarrow_c} \\
&\quad [\text{by Definition 3.26}] \\
&= c \cdot \mathit{ff}
\end{aligned}$$

By Definition 3.26, it follows that $r{\Downarrow_c} = c \cdot \mathit{ff} = (r{\downarrow_{c'}}){\Downarrow_c}$.

$\boxed{r = \mathit{stutt}(\eta^-) \cdot r'}$ By Definition 3.26, it follows that:

$$\begin{aligned}
(r{\downarrow_{c'}}){\Downarrow_c} &= ((\mathit{stutt}(\eta^-) \cdot r'){\downarrow_{c'}}){\Downarrow_c} \\
&\quad [\text{by Definition 3.7}] \\
&= (\mathit{stutt}(\eta^-) \cdot r'{\downarrow_{c'}}){\Downarrow_c} \\
&\quad [\text{by Definition 3.26}] \\
&= c
\end{aligned}$$

By Definition 3.26, $r{\Downarrow_c} = (\mathit{stutt}(\eta^-) \cdot r'){\Downarrow_c} = c$, thus $r{\Downarrow_c} = (r{\downarrow_{c'}}){\Downarrow_c}$.

In order to formulate the following Lemma A.8, we need to introduce the counterpart of $\breve{\|}$ on behavioral timed traces.

Definition A.7 *Let $s, s_1, s_2 \in \mathbf{C}^*$. $\breve{\|} : \mathbf{C}^* \times \mathbf{C}^* \to \mathbf{C}^*$ is defined by structural induction as:*

$$s \mathbin{\breve{\|}} \epsilon := s \qquad \epsilon \mathbin{\breve{\|}} s := s \tag{A.1a}$$

$$(c_1 \cdot s_1) \mathbin{\breve{\|}} (c_2 \cdot s_2) := \begin{cases} (c_1 \otimes c_2) \cdot (c_2 \otimes s_1 \mathbin{\breve{\|}} c_1 \otimes s_2) & \text{if } c_1 \otimes c_2 \neq \mathit{ff} \\ \mathit{ff} & \text{if } c_1 \otimes c_2 = \mathit{ff} \end{cases} \tag{A.1b}$$

where, by abusing notation, $c \otimes (c_1 \cdots c_n)$ denotes $(c \otimes c_1) \cdots (c \otimes c_n)$.

Lemma A.8 *Let $c \in \mathbf{C}$; $A_1, A_2 \in \mathbb{A}^{\Pi}_{\mathbf{C}}$; $\mathcal{I} \in \mathbb{I}$; $r_1 \in \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$ and $r_2 \in \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$ such that $r_1 \mathbin{\bar{\|}} r_2$, $r_1 \Downarrow_c$ and $r_2 \Downarrow_c$ are defined. Then, $(r_1 \mathbin{\bar{\|}} r_2) \Downarrow_c$ is defined and $r_1 \Downarrow_c \mathbin{\breve{\|}} r_2 \Downarrow_c = (r_1 \mathbin{\bar{\|}} r_2) \Downarrow_c$.*

Proof.
Since both $r_1 \Downarrow_c$ and $r_2 \Downarrow_c$ are defined, c satisfies all the conditions in r_1 and r_2. It is easy to notice from Definition 3.10 that c satisfies also the conditions of $r_1 \mathbin{\bar{\|}} r_2$, thus, $(r_1 \mathbin{\bar{\|}} r_2) \Downarrow_c$ is defined as well.

We proceed to prove that $r_1 \Downarrow_c \mathbin{\breve{\|}} r_2 \Downarrow_c = (r_1 \mathbin{\bar{\|}} r_2) \Downarrow_c$ by induction on the structure of r_1.

$r_1 = \epsilon$ and any r_2 By Definition 3.10, $(r_1 \mathbin{\bar{\|}} r_2) \Downarrow_c = (\epsilon \mathbin{\bar{\|}} r_2) \Downarrow_c = r_2 \Downarrow_c$. By Definition 3.26 and by Equation (A.1a), we obtain: $r_1 \Downarrow_c \mathbin{\breve{\|}} r_2 \Downarrow_c = \epsilon \mathbin{\breve{\|}} r_2 \Downarrow_c = r_2 \Downarrow_c$. Thus, $r_1 \Downarrow_c \mathbin{\breve{\|}} r_2 \Downarrow_c = (r_1 \mathbin{\bar{\|}} r_2) \Downarrow_c$.

$r_1 = \boxtimes$ and any r_2 By Definition 3.10, $(r_1 \mathbin{\bar{\|}} r_2) \Downarrow_c = (\boxtimes \mathbin{\bar{\|}} r_2) \Downarrow_c = r_2 \Downarrow_c$. By Definition 3.26 and by Equation (A.1b), $r_1 \Downarrow_c \mathbin{\breve{\|}} r_2 \Downarrow_c = c \mathbin{\breve{\|}} r_2 \Downarrow_c = r_2 \Downarrow_c$. Thus, $r_1 \Downarrow_c \mathbin{\breve{\|}} r_2 \Downarrow_c = (r_1 \mathbin{\bar{\|}} r_2) \Downarrow_c$.

$r_1 = \eta_1 \rightarrowtail d_1 \cdot r_1'$ and $r_2 = \eta_2 \rightarrowtail d_2 \cdot r_2'$

$d_1 \otimes d_2 \neq \mathit{ff}$

$$\begin{aligned}
(r_1 \mathbin{\bar{\|}} r_2) \Downarrow_c &= ((\eta_1 \rightarrowtail d_1 \cdot r_1') \mathbin{\bar{\|}} (\eta_2 \rightarrowtail d_2 \cdot r_2')) \Downarrow_c \\
&\qquad [\text{by Definition 3.10}] \\
&= (\eta_1 \otimes \eta_2 \rightarrowtail d_1 \otimes d_2 \cdot (r_1' \downarrow_{d_2} \mathbin{\bar{\|}} r_2' \downarrow_{d_1})) \Downarrow_c \\
&\qquad [\text{by Definition 3.26}] \\
&= c \cdot (r_1' \downarrow_{d_2} \mathbin{\bar{\|}} r_2' \downarrow_{d_1}) \Downarrow_{c \otimes d_1 \otimes d_2} \\
&\qquad [\text{by Inductive Hypothesis}] \\
&= c \cdot ((r_1' \downarrow_{d_2}) \Downarrow_{c \otimes d_1 \otimes d_2} \mathbin{\breve{\|}} (r_2' \downarrow_{d_1}) \Downarrow_{c \otimes d_1 \otimes d_2}) \\
&\qquad [\text{by Lemma A.6}] \\
&= c \cdot (r_1' \Downarrow_{c \otimes d_1 \otimes d_2} \mathbin{\breve{\|}} r_2' \Downarrow_{c \otimes d_1 \otimes d_2}) \\
&\qquad [d_1 \text{ (resp. } d_2\text{) is entailed by the stores in } r_1' \text{ (resp. } r_2'\text{)}]
\end{aligned}$$

$$
\begin{aligned}
&= c \cdot (r'_1{\Downarrow}_{c\otimes d_1} \mathbin{\breve{\parallel}} r'_2{\Downarrow}_{c\otimes d_2}) \\
&\qquad [\,\text{by Equation (A.1b)}\,] \\
&= (c \cdot r'_1{\Downarrow}_{c\otimes d_1}) \mathbin{\breve{\parallel}} (c \cdot r'_2{\Downarrow}_{c\otimes d_2})
\end{aligned}
$$

By Definition 3.26, $r_1{\Downarrow}_c \mathbin{\breve{\parallel}} r_2{\Downarrow}_c = (c \cdot r'_1{\Downarrow}_{c\otimes d_1}) \mathbin{\breve{\parallel}} (c \cdot r'_2{\Downarrow}_{c\otimes d_2})$; therefore, we conclude $r_1{\Downarrow}_c \mathbin{\breve{\parallel}} r_2{\Downarrow}_c = (r_1 \mathbin{\bar{\parallel}} r_2){\Downarrow}_c$.

$d_1 \otimes d_2 = \mathit{ff}$

$$
\begin{aligned}
(r_1 \mathbin{\bar{\parallel}} r_2){\Downarrow}_c &= ((\eta_1 \rightarrowtail d_1 \cdot r'_1) \mathbin{\bar{\parallel}} (\eta_2 \rightarrowtail d_2 \cdot r'_2)){\Downarrow}_c \\
&\qquad [\,\text{by Definition 3.10}\,] \\
&= (\eta_1 \otimes \eta_2 \rightarrowtail \mathit{ff} \cdot \boxtimes){\Downarrow}_c \\
&\qquad [\,\text{by Definition 3.26}\,] \\
&= c \cdot \mathit{ff}
\end{aligned}
$$

By Definition 3.26 and by Equation (A.1b), $r_1{\Downarrow}_c \mathbin{\breve{\parallel}} r_2{\Downarrow}_c = c \cdot \mathit{ff}$, thus $r_1{\Downarrow}_c \mathbin{\breve{\parallel}} r_2{\Downarrow}_c = (r_1 \mathbin{\bar{\parallel}} r_2){\Downarrow}_c$.

$r_1 = \eta_1 \rightarrowtail d_1 \cdot r'_1$ and $r_2 = \mathit{stutt}(\eta_2^-) \cdot r'_2$

$$
\begin{aligned}
(r_1 \mathbin{\bar{\parallel}} r_2){\Downarrow}_c &= ((\eta_1 \rightarrowtail d_1 \cdot r'_1) \mathbin{\bar{\parallel}} (\mathit{stutt}(\eta_2^-) \cdot r'_2)){\Downarrow}_c \\
&\qquad [\,\text{by Definition 3.10}\,] \\
&= ((\eta_1^+, \eta_1^- \cup \eta_2^-) \rightarrowtail d_1 \cdot (r'_1 \mathbin{\bar{\parallel}} r'_2{\downarrow}_{d_1})){\Downarrow}_c \\
&\qquad [\,\text{by Definition 3.26}\,] \\
&= c \cdot (r'_1 \mathbin{\bar{\parallel}} r'_2{\downarrow}_{d_1}){\Downarrow}_{c\otimes d_1} \\
&\qquad [\,\text{by Inductive Hypothesis}\,] \\
&= c \cdot (r'_1{\Downarrow}_{c\otimes d_1} \mathbin{\breve{\parallel}} (r'_2{\downarrow}_{d_1}){\Downarrow}_{c\otimes d_1}) \\
&\qquad [\,\text{by Lemma A.6}\,] \\
&= c \cdot (r'_1{\Downarrow}_{c\otimes d_1} \mathbin{\breve{\parallel}} r'_2{\Downarrow}_{c\otimes d_1}) \\
&\qquad [\,\text{by Equation (A.1b)}\,] \\
&= (c \cdot r'_1{\Downarrow}_{c\otimes d_1}) \mathbin{\breve{\parallel}} (c \cdot r'_2{\Downarrow}_c)
\end{aligned}
$$

By Definition 3.26, $r_1{\Downarrow}_c \mathbin{\breve{\parallel}} r_2{\Downarrow}_c = (c \cdot r'_1{\Downarrow}_{c\otimes d_1}) \mathbin{\breve{\parallel}} (c \cdot r'_2{\Downarrow}_c)$, thus $r_1{\Downarrow}_c \mathbin{\breve{\parallel}} r_2{\Downarrow}_c = (r_1 \mathbin{\bar{\parallel}} r_2){\Downarrow}_c$.

$r_1 = \mathit{stutt}(\eta_1^-) \cdot r'_1$ and $r_2 = \mathit{stutt}(\eta_2^-) \cdot r'_2$

$$
\begin{aligned}
(r_1 \mathbin{\bar{\parallel}} r_2){\Downarrow}_c &= ((\mathit{stutt}(\eta_1^-) \cdot r'_1) \mathbin{\bar{\parallel}} (\mathit{stutt}(\eta_2^-) \cdot r'_2)){\Downarrow}_c \\
&\qquad [\,\text{by Definition 3.10}\,] \\
&= (\mathit{stutt}(\eta_1^- \cup \eta_2^-) \cdot (r'_1 \mathbin{\bar{\parallel}} r'_2)){\Downarrow}_c \\
&\qquad [\,\text{by Definition 3.26}\,] \\
&= c \cdot (r'_1 \mathbin{\bar{\parallel}} r'_2){\Downarrow}_c \\
&\qquad [\,\text{by Inductive Hypothesis}\,] \\
&= c \cdot (r'_1{\Downarrow}_c \mathbin{\breve{\parallel}} r'_2{\Downarrow}_c) \\
&\qquad [\,\text{by Equation (A.1b)}\,]
\end{aligned}
$$

$$= (c \cdot r'_1 \Downarrow_c) \,\breve{\|}\, (c \cdot r'_2 \Downarrow_c)$$

By Definition 3.26, $r_1 \Downarrow_c \,\breve{\|}\, r_2 \Downarrow_c = (c \cdot r'_1 \Downarrow_c) \,\breve{\|}\, (c \cdot r'_2 \Downarrow_c)$, thus $r_1 \Downarrow_c \,\breve{\|}\, r_2 \Downarrow_c = (r_1 \,\bar{\|}\, r_2) \Downarrow_c$.

Proof of Theorem 3.27.

Let $d \in \mathbf{C}$ and $P = D.A$ with $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$, we proceed by structural induction on A.

skip The proof in this case is straightforward.

$$\mathit{prefix}(\mathcal{A}[\![\mathsf{skip}]\!]_{\mathcal{F}[\![D]\!]}) \Downarrow_d = \mathit{prefix}(\{\boxtimes\}) \Downarrow_d = \{\epsilon,\ d\} = \mathcal{B}^{ss}[\![D \,.\, \mathsf{skip}]\!]_d$$

tell(c)

$$\begin{aligned}
\mathit{prefix}((\mathcal{A}[\![\mathsf{tell}(c)]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d) &= \mathit{prefix}((tt, \varnothing) \rightarrowtail c \cdot \boxtimes) \Downarrow_d) \\
&= \mathit{prefix}(d \cdot (d \otimes c)) \\
&= \mathcal{B}^{ss}[\![D \,.\, \mathsf{tell}(c)]\!]_d
\end{aligned}$$

A$=\sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i$ We prove the two directions independently.

$\subseteq$ We show that, given a conditional trace $r \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$, it holds that $\forall d \in \mathbf{C}.\, \mathit{prefix}(r \Downarrow_d) \subseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d$. We have to distinguish two cases.

1. Let $r = (c_j, \varnothing) \rightarrowtail c_j \cdot r_j \downarrow_{c_j}$ with $1 \leq j \leq n$ and $r_j \in \mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]}$ that is c_j-compatible. Consider $d \in \mathbf{C}$ such that $r \Downarrow_d$ is defined, thus $d \vdash c_j$ and $(r_j \downarrow_{c_j}) \Downarrow_{d \otimes c_j}$ is defined too; assume $d \neq \mathit{ff}$, then

$$\begin{aligned}
\mathit{prefix}(r \Downarrow_d) &= \mathit{prefix}(\{((c_j, \varnothing) \rightarrowtail c_j \cdot r_j \downarrow_{c_j}) \Downarrow_d \mid 1 \leq j \leq n,\ r_j \in \mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]}\}) \\
&\qquad [\text{by Definition 3.26}] \\
&= \mathit{prefix}(\{d \cdot (r_j \downarrow_{c_j}) \Downarrow_{d \otimes c_j} \mid 1 \leq j \leq n,\ r_j \in \mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]}\}) \\
&\qquad [\text{by Lemma A.6 and since } d \vdash c_j] \\
&= \mathit{prefix}(\{d \cdot r_j \Downarrow_d \mid 1 \leq j \leq n,\ r_j \in \mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]}\}) \\
&\qquad [\text{by Equation (3.1)}] \\
&= \{\epsilon,\ d\} \cup \{d \cdot s \mid 1 \leq j \leq n,\ s \in \mathit{prefix}(\mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d)\} \\
&\qquad [\text{by Inductive Hypothesis}] \\
&\subseteq \{\epsilon,\ d\} \cup \{d \cdot s \mid 1 \leq j \leq n,\ s \in \mathcal{B}^{ss}[\![D \,.\, A_j]\!]_d\}
\end{aligned}$$

The element ϵ directly belongs to $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d$. Since $d \vdash c_j$, also d belongs to $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d$ (at least one step is performed in the computation). Finally, the set $\{d \cdot s \mid 1 \leq j \leq n,\ \in \mathcal{B}^{ss}[\![D \,.\, A_j]\!]_d\}$ is also contained in $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d$. In particular, following Rule **R2**, the agent $\sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i$ (executed with a store d that entails one of the guards, e.g. c_j) behaves, in the next time instant, as the corresponding agent A_j over the store (which is not modified in that step). If $d = \mathit{ff}$, by definition of $\Downarrow$ (Definition 3.26), we have that $\mathit{prefix}(r \Downarrow_{\mathit{ff}}) = \{\epsilon, \mathit{ff}\}$ which corresponds to the set $\mathcal{B}^{ss}[\![D \,.\, A]\!]_{\mathit{ff}}$ since the transition relation $\to$ is not defined for the configuration $\langle A,\ \mathit{ff}\rangle$. For the case $d \nvdash c_j$, the operator $\Downarrow_d$ computes no trace from r, thus $\varnothing \subseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d$ holds.

2. Let $r = \mathit{stutt}(\{c_1, \ldots, c_n\}) \cdot r'$ such that $r' \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$ and for all $1 \leq j \leq n$, $c_j \neq tt$. Consider $d \in \mathbf{C}$ such that for all $1 \leq j \leq n$, $d \nvdash c_j$, then $\mathit{prefix}(r{\Downarrow_d}) = \{\epsilon,\ d\} \subseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d$. Otherwise, if it exists $1 \leq j \leq n$ such that $d \vdash c_j$, then the operator $\Downarrow_d$ is not defined on r, thus $\mathit{prefix}(r{\Downarrow_d}) = \varnothing \subseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d$.

$\boxed{\supseteq}$ For each $d \in \mathbf{C}$, it exists a conditional trace $r \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$ such that $\mathit{prefix}(r{\Downarrow_d}) \supseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d$. There are three cases to be considered:

1. when the store d does not satisfy any guard (in this case, the agent does not run any computation step),
2. when there exists a guard c_j such that $d \vdash c_j$ and $d \neq \mathit{ff}$, and finally
3. when $d = \mathit{ff}$.

Let us prove them.

1. Suppose that for all $1 \leq j \leq n$, $d \nvdash c_j$; then, the small-step behavior is $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d = \{\epsilon,\ d\}$. Thus, it exists a conditional trace $r \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$ such that $r = \mathit{stutt}(\{c_1, \ldots, c_n\}) \cdot r'$ with $r' \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. Moreover, by Definition 3.26 and Definition 3.1, it follows that $\mathit{prefix}(r{\Downarrow_d}) = \{\epsilon,\ d\} \supseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d$.
2. Suppose that there exists $1 \leq j \leq n$ such that $d \vdash c_j$ and $d \neq \mathit{ff}$. One of the conditional traces computed by the semantics evaluation function $\mathcal{A}$ is $r = (c_j, \varnothing) \rightarrowtail c_j \cdot r_j{\downarrow_{c_j}}$ with $r_j \in \mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]}$. Then, we have:

$$\begin{aligned}
\mathit{prefix}(r{\Downarrow_d}) &= \mathit{prefix}(((c_j, \varnothing) \rightarrowtail c_j \cdot r_j{\downarrow_{c_j}}){\Downarrow_d}) \\
&\qquad [\text{by Definition } 3.26] \\
&= \mathit{prefix}(\{d \cdot (r_j{\downarrow_{c_j}}){\Downarrow_{d \otimes c_j}} \mid r_j \in \mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]}\}) \\
&\qquad [\text{by Lemma A.6 and since } d \vdash c_j] \\
&= \mathit{prefix}(\{d \cdot r_j{\Downarrow_d} \mid r_j{\Downarrow_d} \in \mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]}{\Downarrow_d}\}) \\
&\qquad [\text{by Equation } (3.1)] \\
&= \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathit{prefix}(\mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]}{\Downarrow_d})\} \\
&\qquad [\text{by Inductive Hypothesis}] \\
&\supseteq \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathcal{B}^{ss}[\![D \,.\, A_j]\!]_d\} \\
&\qquad [\text{by Rule } \mathbf{R2}] \\
&\supseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d
\end{aligned}$$

3. Suppose that $d = \mathit{ff}$. As seen before, $r = (c_j, \varnothing) \rightarrowtail c_j \cdot r_j{\downarrow_{c_j}}$ with $r_j \in \mathcal{A}[\![A_j]\!]_{\mathcal{F}[\![D]\!]}$ and $1 \leq j \leq n$ belongs to $\mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. Then, we have:

$$\begin{aligned}
\mathit{prefix}(r{\Downarrow_{\mathit{ff}}}) &= \mathit{prefix}(((c_j, \varnothing) \rightarrowtail c_j \cdot r_j{\downarrow_{c_j}}){\Downarrow_{\mathit{ff}}}) \\
&\qquad [\text{by Definition } 3.26] \\
&= \{\epsilon,\ \mathit{ff}\} \\
&\qquad [\text{by Definition } 3.1] \\
&\supseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_{\mathit{ff}}
\end{aligned}$$

Therefore, we can conclude that $\mathit{prefix}(\mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}{\Downarrow_d}) = \mathcal{B}^{ss}[\![D \,.\, A]\!]_d$.

$\underline{\textsf{now } c \textsf{ then } A_1 \textsf{ else } A_2}\rfloor$ We prove the two directions independently. We abbreviate the conditional agent and call it A ($A \coloneqq \textsf{now } c \textsf{ then } A_1 \textsf{ else } A_2$).

$\subseteq\rfloor$ There are seven possible cases, one for each type of trace r in (3.9).

1. Consider $r = (c, \varnothing) \rightarrowtail c \cdot \boxtimes$, and assume that $\boxtimes \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$, which means, by Definition 3.16, that $A_1 = \textsf{skip}$. We consider now the three possible cases:
   (a) If $d \vdash c$ and $d \neq f\!f$, then it is straightforward that $\mathit{prefix}(r \Downarrow_d) = \mathit{prefix}(d \cdot d) = \{\epsilon,\ d,\ d \cdot d\}$. On the behavioral part, we know from Rule **R4** that the observable of A is the set of all prefixes of $d \cdot d$, so we can conclude $\mathit{prefix}(r \Downarrow_d) \subseteq \mathcal{B}^{ss}[\![D\,.\,A]\!]_d$.
   (b) If $d = f\!f$, the small-step behavior is $\mathcal{B}^{ss}[\![D\,.\,A]\!]_{f\!f} = \{\epsilon, f\!f\}$. Since $f\!f \vdash c$ it is straightforward that $\mathit{prefix}(r \Downarrow_{f\!f}) = \{\epsilon, f\!f\} = \mathcal{B}^{ss}[\![D\,.\,A]\!]_{f\!f}$.
   (c) If $d \nvdash c$, then the application of $\Downarrow_d$ to the agent semantics does not compute any behavioral timed trace. Therefore, $\mathit{prefix}(r \Downarrow_d) = \varnothing \subseteq \mathcal{B}^{ss}[\![D\,.\,A]\!]_d$.
2. Consider $r = (\eta^+ \otimes c, \eta^-) \rightarrowtail a \otimes c \cdot r' \downarrow_c$ with $(\eta^+, \eta^-) \rightarrowtail a \cdot r' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$, r' being d-compatible, $a \otimes c \neq f\!f$ and $\forall h^- \in \eta^-.\, \eta^+ \otimes c \nvdash h^-$. Let $d \in \mathbf{C}$ such that $d \Vdash (\eta^+ \otimes c, \eta^-)$. This implies that $d \vdash c$ since c belongs to the positive condition. Under these conditions, we have:

$$
\begin{aligned}
&\mathit{prefix}(r \Downarrow_d) = \\
&\quad = \mathit{prefix}(\{((\eta^+ \otimes c, \eta^-) \rightarrowtail a \otimes c \cdot r' \downarrow_c) \Downarrow_d \mid (\eta^+, \eta^-) \rightarrowtail a \cdot r' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}\}) \\
&\qquad\quad [\text{by Definition 3.26}] \\
&\quad = \mathit{prefix}(\{d \cdot (r' \downarrow_c) \Downarrow_{d \otimes a \otimes c} \mid d \cdot r' \Downarrow_{d \otimes a} \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d\}) \\
&\qquad\quad [\text{by Lemma A.6 since } d \vdash c] \\
&\quad = \mathit{prefix}(\{d \cdot r' \Downarrow_{d \otimes a} \mid d \cdot r' \Downarrow_{d \otimes a} \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d\}) \\
&\quad = \mathit{prefix}(\mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d) \\
&\qquad\quad [\text{by Inductive Hypothesis}] \\
&\quad \subseteq \mathcal{B}^{ss}[\![D\,.\,A_1]\!]_d \\
&\qquad\quad [\text{by Rule } \mathbf{R3}] \\
&\quad \subseteq \mathcal{B}^{ss}[\![D\,.\,A]\!]_d
\end{aligned}
$$

   For the case when $d \not\Vdash (\eta^+ \otimes c, \eta^-)$ (and also when r' is not d-compatible), $\mathit{prefix}(r \Downarrow_d) = \varnothing$ which is directly included in $\mathcal{B}^{ss}[\![D\,.\,A]\!]_d$.
3. Consider the conditional trace $r = (\eta^+ \otimes c, \eta^-) \rightarrowtail f\!f \cdot \boxtimes$. Let d be a store such that $d \Vdash (\eta^+ \otimes c, \eta^-)$. This implies that $d \vdash c$. Under these conditions, we get:

$$
\begin{aligned}
\mathit{prefix}(r \Downarrow_d) &= \mathit{prefix}(((\eta^+ \otimes c, \eta^-) \rightarrowtail f\!f \cdot \boxtimes) \Downarrow_d) \\
&\qquad [\text{by Definition 3.26}] \\
&= \{\epsilon, f\!f\} \\
&\qquad [\text{by Definition 3.1}]
\end{aligned}
$$

$$\subseteq \mathcal{B}^{ss}[\![D\,.\,A]\!]_d$$

For the case when $d \not\Vdash (\eta^+ \otimes c, \eta^-)$, $\mathit{prefix}(r\Downarrow_d) = \varnothing$ which is directly included in $\mathcal{B}^{ss}[\![D\,.\,A]\!]_d$.

4. Let be $r = (c, \eta^-) \rightarrowtail c \cdot r'$ with $\mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$ and r' d-compatible. Assume that $d \Vdash (c, \eta^-)$ and $d \neq \mathit{ff}$. Then, we have:

$$\begin{aligned}
\mathit{prefix}(r\Downarrow_d) &= \mathit{prefix}(\{((c,\eta^-) \rightarrowtail c \cdot r')\Downarrow_d \mid \mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}\})\\
&\qquad [\text{by Definition 3.26}]\\
&\mathit{prefix}(\{d \cdot r'\Downarrow_{d\otimes c} \mid \mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}\})\\
&\qquad [\text{since } d \vdash c]\\
&= \mathit{prefix}(\{d \cdot r'\Downarrow_d \mid \mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}\})\\
&\qquad [\text{by Definition 3.16}]\\
&= \mathit{prefix}(\{d \cdot r'\Downarrow_d \mid r' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}\})\\
&\qquad [\text{by Equation (3.1)}]\\
&= \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathit{prefix}(\mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}\Downarrow_d)\}\\
&\qquad [\text{by Inductive Hypothesis}]\\
&\subseteq \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathcal{B}^{ss}[\![D\,.\,A_1]\!]_d\}\\
&\qquad [\text{by Rule } \mathbf{R4}]\\
&\subseteq \mathcal{B}^{ss}[\![D\,.\,A]\!]_d
\end{aligned}$$

The third step follows from the definition of the semantics $\mathcal{A}$ (Definition 3.16). The construct *stutt* is introduced only by an ask agent. Thus, we know that A_1 is an ask agent. The Equation (3.8), states that $\mathit{stutt}(\eta^-)$ is always followed by a conditional trace which belongs to the semantics of the ask, which can be reduced to say that r' belongs to $\mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$.
If $d = \mathit{ff}$ we have that $\mathit{prefix}(r\Downarrow_{\mathit{ff}}) = \{\epsilon, \mathit{ff}\}$ which corresponds to the behavior $\mathcal{B}^{ss}[\![D\,.\,A]\!]_{\mathit{ff}}$ since the transition relation $\rightarrow$ is not defined for the agent A starting with store ff.
If $d \not\Vdash (c, \eta^-)$ (and also when r' is not d-compatible) we have that $\mathit{prefix}(r\Downarrow_d) = \varnothing \subseteq \mathcal{B}^{ss}[\![D\,.\,A]\!]_d$.

5. Let be $r = (\mathit{tt}, \{c\}) \rightarrowtail \mathit{tt} \cdot \boxtimes$ with $\boxtimes \in \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$. By Definition 3.16, it follows that A_2 is a skip agent. If $d \nvdash c$, it is straightforward that $\mathit{prefix}(r\Downarrow_d) = \mathit{prefix}(d \cdot d) = \{\epsilon,\ d,\ d \cdot d\}$. From Rule **R6**, we know that the observable of the agent A consists of the set of all prefixes of $d \cdot d$. Therefore, $\mathit{prefix}(r\Downarrow_d) \subseteq \mathcal{B}^{ss}[\![D\,.\,A]\!]_d$. On the contrary, if $d \vdash c$, $r\Downarrow_d$ does not compute any trace because d does not satisfy the condition, thus $\mathit{prefix}(r\Downarrow_d) = \varnothing \subseteq \mathcal{B}^{ss}[\![D\,.\,A]\!]_d$.

6. Consider now $r = (\eta^+, \eta^- \cup \{c\}) \rightarrowtail c' \cdot r'$ such that $(\eta^+, \eta^-) \rightarrowtail c' \cdot r' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$ and $c' \nvdash c$. If $d \Vdash (\eta^+, \eta^- \cup \{c\})$, we know also that $d \nvdash c$. Under these conditions, we have:

$$\begin{aligned}
\mathit{prefix}(\{r\Downarrow_d) &= \mathit{prefix}(\{((\eta^+, \eta^- \cup \{c\}) \rightarrowtail c' \cdot r')\Downarrow_d \mid (\eta^+, \eta^-) \rightarrowtail c' \cdot r' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\})\\
&\qquad [\text{by Definition 3.26}]\\
&= \mathit{prefix}(\{d \cdot r'\Downarrow_{d\otimes c'} \mid d \cdot r'\Downarrow_{d\otimes c'} \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\Downarrow_d\})
\end{aligned}$$

$$
\begin{aligned}
&= \mathit{prefix}(\mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d) \\
&\qquad [\,\text{by Inductive Hypothesis}\,] \\
&\subseteq \mathcal{B}^{ss}[\![D \,.\, A_2]\!]_d \\
&\qquad [\,\text{by Rule } \mathbf{R5}\,] \\
&\subseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d
\end{aligned}
$$

Otherwise, if $d \Vdash\!\!\!\!/\; (\eta^+, \eta^- \cup \{c\})$, then $\mathit{prefix}(r\Downarrow_d) = \varnothing$, which is directly contained in $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d$.

7. Finally, let us now consider the case when $r = (tt, \eta^- \cup \{c\}) \rightarrowtail tt \cdot r'$ with $\mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$. Assume that $d \Vdash (tt, \eta^- \cup \{c\})$. Then, we have:

$$
\begin{aligned}
\mathit{prefix}(r\Downarrow_d) &= \mathit{prefix}(\{((tt, \eta^- \cup \{c\}) \rightarrowtail tt \cdot r')\Downarrow_d \mid \mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\}) \\
&\qquad [\,\text{by Definition 3.26}\,] \\
&\quad \mathit{prefix}(\{d \cdot r'\Downarrow_d \mid \mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\}) \\
&\qquad [\,\text{by Definition 3.16}\,] \\
&= \mathit{prefix}(\{d \cdot r'\Downarrow_d \mid r' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\}) \\
&\qquad [\,\text{by Equation (3.1)}\,] \\
&= \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathit{prefix}(\mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d)\} \\
&\qquad [\,\text{by Inductive Hypothesis}\,] \\
&\subseteq \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathcal{B}^{ss}[\![D \,.\, A_2]\!]_d\} \\
&\qquad [\,\text{by Rule } \mathbf{R6}\,] \\
&\subseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d
\end{aligned}
$$

The third step, similarly to case Point 4, can be done since each construct $\mathit{stutt}(\eta^-)$ is introduced by a choice agent, and Equation (3.8) states that it is always followed by a conditional trace r' belonging recursively to the semantics of A_2.
If the initial assumption does not hold (e.g. if $d \Vdash\!\!\!\!/\; (tt, \eta^- \cup \{c\})$), we have that $\mathit{prefix}(r\Downarrow_d) = \varnothing \subseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d$.

$\boxed{\supseteq}$ We have four cases, one for each rule defining the operational semantics for the conditional agent in Figure 1.

Rule R3 Let us recall the conditions to apply Rule **R3**: it must occur $\langle A_1,\, d\rangle \to \langle A_1',\, d'\rangle$ and $d \vdash c$. In this case, we have that $\mathcal{B}^{ss}[\![D.A]\!]_d = \mathcal{B}^{ss}[\![D.A_1]\!]_d$. By inductive hypothesis, we know that $\mathit{prefix}(\mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d) \supseteq \mathcal{B}^{ss}[\![D.A_1]\!]_d$, thus also $\mathit{prefix}(\mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d) \supseteq \mathcal{B}^{ss}[\![D \,.\, A]\!]_d$. Next, we prove the inclusion $\mathit{prefix}(\mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d) \supseteq \mathit{prefix}(\mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]})\Downarrow_d$. We proceed by induction on the structure of a generic $r_1 \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$ in order to find $r \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$ such that $\mathit{prefix}(r_1\Downarrow_d) \subseteq \mathit{prefix}(r\Downarrow_d)$.

1. If $r_1 = \boxtimes$, then by Equation (3.9) the conditional trace $r = (c, \varnothing) \rightarrowtail c \cdot \boxtimes \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. We know that $\boxtimes\Downarrow_d = d$ and $r\Downarrow_d = d \cdot (d \otimes c) = d \cdot d$, since $d \vdash c$. It is easy to see that the prefixes of d are all included in the prefixes of $d \cdot d$.
2. If $r_1 = (\eta^+, \eta^-) \rightarrowtail c' \cdot r'$, then, by definition, the conditional trace $r = (\eta^+ \otimes c, \eta^-) \rightarrowtail c' \otimes c \cdot r'\downarrow_c \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. Let us

now assume that $d \Vdash (\eta^+, \eta^-)$. Then, $r_1 \Downarrow_d = d \cdot r' \Downarrow_{d \otimes c'}$ and, since $d \vdash c$ by the initial assumptions, $r \Downarrow_d = d \cdot r' \Downarrow_{d \otimes c' \otimes c} = d \cdot r' \Downarrow_{d \otimes c'} = r_1 \Downarrow_d$, thus the inclusion of the prefixes directly holds. Otherwise, if $d \nVdash (\eta^+, \eta^-)$, then the operator $\Downarrow_d$ is undefined in both cases.

3. Finally, if $r_1 = stutt(\eta^-) \cdot r'$, then, by definition, $r = (c, \eta^-) \rightarrowtail c \cdot r' \downarrow_c \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. Assume that for all $h^- \in \eta^-$, $d \nvdash h^-$. Then, $r_1 \Downarrow_d = d$ and it holds that its prefixes are all included in the prefixes of $r \Downarrow_d = d \cdot r \Downarrow_{d \otimes c}$. Otherwise, if it exists $h^- \in \eta^-$ such that $d \vdash h^-$, then the $\Downarrow_d$ operator is undefined in both cases.

Rule R4 The conditions to apply this rule are $\langle A_1, d\rangle \not\rightarrow$, $d \vdash c$ and $d \neq \mathit{ff}$, in which case the small-step behavior is defined as $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d = prefix(d \cdot d)$. There are two cases in which it may happen that $\langle A_1, d\rangle \not\rightarrow$: (1) when A_1 is a skip agent or (2) if A_1 is a choice agent whose guards are not satisfied by d. Let us discuss them independently.

1. If $A_1 = \mathsf{skip}$, then by Equation (3.2), $\boxtimes \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$ and $r = (c, \varnothing) \rightarrowtail c \cdot \boxtimes \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. We now have that $r \Downarrow_d = d \cdot (d \otimes c) = d \cdot d$, whose prefixes coincide with $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d$.
2. If $A_1 = \sum_{i=1}^{n} \mathsf{ask}(c_i) \rightarrow B_i$ and for all $1 \leq i \leq n \nvdash c_i$, by Equation (3.8) $stutt(\{c_1, \ldots, c_n\}) \cdot r' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$ and, as a consequence, $r = (c, \{c_1, \ldots, c_n\}) \rightarrowtail c \cdot r'$ belongs to $\mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. Now we compute $r \Downarrow_d$ and we get the trace $d \cdot r' \Downarrow_{d \otimes c} = d \cdot r' \Downarrow_d$. By definition of the evaluation function $\mathcal{A}$, r' is different from the empty conditional trace ϵ (by Equation (3.8) a $stutt$ construct is always followed by another conditional state). Therefore, $r' \Downarrow_d = d \cdot d \cdot s$ for some behavioral trace s. As a consequence, the behavior of the agent $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d = d \cdot d$ is included in the set of prefixes of $r \Downarrow_d = d \cdot d \cdot s$.

   In case $d = \mathit{ff}$ we are not allowed to apply any rule in Figure 1, so the small-step behavior is $\mathcal{B}^{ss}[\![D \,.\, A]\!]_{\mathit{ff}} = \{\epsilon, \mathit{ff}\}$. In this case, $A_1 = \mathsf{skip}$ since ff is strong enough to entail any guard of a generic agent $\sum_{i=1}^{n} \mathsf{ask}(c_i) \rightarrow B_i$. As explained above, $\boxtimes \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$ and $r = (c, \varnothing) \rightarrowtail c \cdot \boxtimes \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$, thus $r \Downarrow_{\mathit{ff}} = \mathit{ff}$, and it is easy to note that $\mathcal{B}^{ss}[\![D \,.\, A]\!]_{\mathit{ff}} \in prefix(\mathit{ff})$.

Rule R5 This case is analogous to the case for Rule **R3** but, instead executing the then branch (A_1), the else branch of the conditional agent (A_2) is taken, under the condition that $d \nvdash c$. More specifically, the conditions imposed for the application of the rule are $\langle A_2, d\rangle \rightarrow \langle A_2', d'\rangle$ and $d \nvdash c$, in which case $\mathcal{B}^{ss}[\![D.A]\!]_d = \mathcal{B}^{ss}[\![D.A_2]\!]_d$. By inductive hypothesis, we know that $prefix(\mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d) \supseteq \mathcal{B}^{ss}[\![D.A_1]\!]_d$, thus also $prefix(\mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d) \supseteq \mathcal{B}^{ss}[\![D.A]\!]_d$. In the following, we prove that $prefix(\mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d) \supseteq prefix(\mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d)$ when $d \nvdash c$. We proceed by induction on the structure of a generic $r_2 \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$ in order to find a conditional trace $r \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$ such that $prefix(r_2 \Downarrow_d) \subseteq prefix(r \Downarrow_d)$.

1. If $r_2 = \boxtimes$, then the conditional trace $r = (tt, \{c\}) \rightarrowtail tt \cdot \boxtimes$ belongs to $\mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. We have $\boxtimes \Downarrow_d = d$, whose prefixes are

included in those of $r\Downarrow_d = d \cdot d$.

2. If $r_2 = (\eta^+, \eta^-) \rightarrowtail c' \cdot r'$, then the conditional trace $r = (\eta^+, \eta^- \cup \{c\}) \rightarrowtail c' \cdot r' \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. Let us now assume that $d \Vdash (\eta^+, \eta^-)$; then, $r_2\Downarrow_d = d \cdot r'\Downarrow_{d \otimes c'}$. In addition, since by the initial assumptions $d \nvdash c$, $r\Downarrow_d = d \cdot r'\Downarrow_{d\otimes c'}$, thus the inclusion of the prefixes directly holds. Otherwise, if $d \nVdash (\eta^+, \eta^-)$, then the operator $\Downarrow_d$ is undefined in both cases.
3. Finally, if $r_2 = \mathit{stutt}(\eta^-) \cdot r'$, then by definition the conditional trace $r = (\mathit{tt}, \eta^- \cup \{c\}) \rightarrowtail \mathit{tt} \cdot r' \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. Assume that for all $h^- \in \eta^-$, $d \nvdash h^-$. Then, $r_2\Downarrow_d = d$, and it holds that its prefixes are all included in the prefixes of $r\Downarrow_d = d \cdot r'\Downarrow_d$. Otherwise, if it exists $h^- \in \eta^-$ such that $d \vdash h^-$ the $\Downarrow_d$ operator is undefined in both cases.

Rule R6 This case is analogous to the case for Rule **R4**. Now, the conditions to apply the rule are that $\langle A_2, d\rangle \not\rightarrow$ and $d \nvdash c$. In this case, the small-step behavior is $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d = \mathit{prefix}(d \cdot d)$. There are two cases in which it may happen that $\langle A_2, d\rangle \not\rightarrow$:

1. when A_2 is a skip agent or
2. if A_2 is a choice agent whose guards are not satisfied by d.

Let us discuss them independently.

1. If $A_2 = \mathsf{skip}$, by Equation (3.2) $\boxtimes \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$ and $r = (\mathit{tt}, \{c\}) \rightarrowtail \mathit{tt} \cdot \boxtimes \in \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. Then, since $d \nvdash c$, we have that $r\Downarrow_d = d \cdot d$ which coincides with $\mathcal{B}^{ss}[\![D \,.\, A]\!]_d$.
2. If $A_2 = \sum_{i=1}^{n} \mathsf{ask}(c_i) \rightarrow B_i$ and for all $1 \leq i \leq n \nvdash c_i$, then, by Equation (3.8), $\mathit{stutt}(\{c_1, \ldots, c_n\}) \cdot r' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$ and, as a consequence, $r = (c, \{c_1, \ldots, c_n\}) \rightarrowtail c \cdot r'$ belongs to $\mathcal{A}[\![A]\!]_{\mathcal{F}[\![D]\!]}$. Now, we compute $r\Downarrow_d$ and we get as result the trace $d \cdot r'\Downarrow_d$. Since, by definition of the semantics evaluation function $\mathcal{A}$, a *stutt* is always followed by another conditional tuple, then r' is different from the empty trace. Therefore, $r'\Downarrow_d = d \cdot s$ for some trace s. As a consequence, the behavior of the agent $\mathcal{B}^{ss}[\![D . A]\!]_d = d \cdot d$ is included in the set of prefixes of $r\Downarrow_d = d \cdot d \cdot s$.

$A_1 \parallel A_2$ For this case, we need to use the operator of parallel composition between behavioral timed traces defined in Definition A.7. We extend the parallel composition to sets of behavioral timed traces as: $S_1 \breve{\parallel} S_2 = \{s_1 \breve{\parallel} s_2 \mid s_1 \in S_1 \text{ and } s_2 \in S_2\}$. Furthermore, we abuse of notation and use the symbol $\otimes$ for the merge of a store with the stores of behavioral timed traces. Intuitively, given $c \in \mathbf{C}$ and $s \in \mathbf{C}^*$, $c \otimes s$ means that we add the information c to each store in s. We prove the two directions independently.

$\subseteq$ We distinguish five different cases.

1. Let r_1 be a generic conditional trace and $r_2 = \boxtimes$ (or $r_2 = \epsilon$). By Definition 3.10, $r_1 \bar{\parallel} r_2 = r_1$. In other words, the conditional trace r_2 is associated to an agent that adds no information:

$$\mathit{prefix}((r_1 \bar{\parallel} r_2)\Downarrow_d) = \mathit{prefix}(r_1\Downarrow_d)$$

$$= \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d$$
[by Inductive Hypothesis]
$$\subseteq \mathcal{B}^{ss}[\![D \,.\, A_1]\!]_d$$
$$= \mathcal{B}^{ss}[\![D \,.\, A_1 \parallel A_2]\!]_d$$

Since A_2 does not modify the store, we can conclude that the two behaviors $\mathcal{B}^{ss}[\![D \,.\, A_1]\!]_d$ and $\mathcal{B}^{ss}[\![D \,.\, A_1 \parallel A_2]\!]_d$ coincide.

2. Let be $r = \mathit{stutt}(\eta_1^- \cup \eta_2^-) \cdot r'$ and assume that $d \nVdash h^-$ for all $h^- \in (\eta^- \cup \delta^-)$. Then,

$$\mathit{prefix}(r\Downarrow_d) = \mathit{prefix}(d) = \{\epsilon,\ d\} \subseteq \mathcal{B}^{ss}[\![D \,.\, A_1 \parallel A_2]\!]_d$$

If the initial assumption does not hold, then the set $\mathit{prefix}(r\Downarrow_d)$ is empty and the inclusion directly holds.

3. Let $r = (\eta \otimes \delta) \rightarrowtail c_1 \otimes c_2 \cdot (r_1'\downarrow_{c_2} \bar{\parallel}\, r_2'\downarrow_{c_1})$ be a conditional trace in $\mathcal{A}[\![A_1 \parallel A_2]\!]_{\mathcal{F}[\![D]\!]}$ such that $r_1 = \eta \rightarrowtail c_1 \cdot r_1' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$, $r_2 = \eta \rightarrowtail c_2 \cdot r_2' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$ and $(c_1 \otimes c_2) \neq \mathit{ff}$. Assume that $d \Vdash (\eta \otimes \delta)$ and $d \neq \mathit{ff}$. Moreover, due to the form of r_1 and r_2, we know that there exist two agents A_1' and A_2' such that $\langle A_1, d\rangle \to \langle A_1', d \otimes c_1\rangle$ and $\langle A_2, d\rangle \to \langle A_2', d \otimes c_2\rangle$, respectively. Then,

$$\mathit{prefix}(r\Downarrow_d) =$$
$$= \mathit{prefix}(\{d \cdot (r_1'\downarrow_{c_2} \bar{\parallel}\, r_2'\downarrow_{c_1})\Downarrow_{d\otimes c_1 \otimes c_2} \mid r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]},\ r_2' \in \mathcal{A}[\![A_2']\!]_{\mathcal{F}[\![D]\!]}\})$$
[c_1 and c_2 are already in the stores of r_1 and r_2, respectively]
$$= \mathit{prefix}(\{d \cdot (r_1'\downarrow_{c_1\otimes c_2} \bar{\parallel}\, r_2'\downarrow_{c_1\otimes c_2})\Downarrow_{d\otimes c_1 \otimes c_2} \mid r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]} \text{ and } r_2' \in \mathcal{A}[\![A_2']\!]_{\mathcal{F}[\![D]\!]}\})$$
[by Lemma A.3]
$$= \mathit{prefix}(\{d \cdot (r_1' \bar{\parallel}\, r_2')\downarrow_{c_1\otimes c_2}\Downarrow_{d\otimes c_1 \otimes c_2} \mid r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]} \text{ and } r_2' \in \mathcal{A}[\![A_2']\!]_{\mathcal{F}[\![D]\!]}\})$$
[by Lemma A.6]
$$= \mathit{prefix}(\{d \cdot (r_1' \bar{\parallel}\, r_2')\Downarrow_{d\otimes c_1 \otimes c_2} \mid r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]} \text{ and } r_2' \in \mathcal{A}[\![A_2']\!]_{\mathcal{F}[\![D]\!]}\})$$
[by Lemma A.8]
$$= \mathit{prefix}(\{d \cdot (r_1'\Downarrow_{d\otimes c_1 \otimes c_2} \breve{\parallel}\, r_2'\Downarrow_{d\otimes c_1 \otimes c_2}) \mid r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]} \text{ and } r_2' \in \mathcal{A}[\![A_2']\!]_{\mathcal{F}[\![D]\!]}\})$$
[by Equation (3.1)]
$$\begin{aligned} = \{\epsilon, d\} \cup \{d \cdot (s_1' \breve{\parallel}\, s_2') \mid\ & s_1' \in \mathit{prefix}(\mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]} \Downarrow_{d\otimes c_1 \otimes c_2}), \\ & s_2' \in \mathit{prefix}(\mathcal{A}[\![A_2']\!]_{\mathcal{F}[\![D]\!]} \Downarrow_{d\otimes c_1 \otimes c_2})\} \end{aligned}$$
[by Inductive Hypothesis]
$$\subseteq \{\epsilon, d\} \cup \{d \cdot (s_1' \breve{\parallel}\, s_2') \mid s_1' \in \mathcal{B}^{ss}[\![D \,.\, A_1']\!]_{d\otimes c_1 \otimes c_2},\ s_2' \in \mathcal{B}^{ss}[\![D \,.\, A_2']\!]_{d\otimes c_1 \otimes c_2}\}$$

[by Definition A.7 and by Definition 3.1]
$$\subseteq \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathcal{B}^{ss}[\![D \,.\, A_1' \parallel A_2']\!]_{d\otimes c_1 \otimes c_2}\}$$
[by Rule **R7**]
$$\subseteq \mathcal{B}^{ss}[\![D \,.\, A_1 \parallel A_2]\!]_d$$

If $d = \mathit{ff}$ we have that $\langle A_1, \mathit{ff}\rangle \not\to$ and $\langle A_2, \mathit{ff}\rangle \not\to$, thus $\mathcal{B}^{ss}[\![D . A_1 \parallel A_2]\!]_{\mathit{ff}} = \{\epsilon, \mathit{ff}\}$. Since $d \Vdash (\eta \otimes \delta)$ we have that $\mathit{prefix}(r\Downarrow_{\mathit{ff}}) =$

$\{\epsilon, \mathit{ff}\}$ which corresponds to the small-step behavior $\mathcal{B}^{ss}[\![D . A_1 \parallel A_2]\!]_{\mathit{ff}}$.
If $d \not\Vdash (\eta \otimes \delta)$, then the set $\mathit{prefix}(r\Downarrow_d)$ is empty since $\Downarrow_d$ is not defined under these conditions, thus it is directly included in $\mathcal{B}^{ss}[\![D . A_1 \parallel A_2]\!]_d$.

4. Let us consider now a conditional trace of the form $r = (\eta \otimes \delta) \rightarrowtail \mathit{ff} \cdot \boxtimes$ such that $r_1 = \eta \rightarrowtail c_1 \cdot r_1'$, $r_2 = \delta \rightarrowtail c_2 \cdot r_2'$ and $c_1 \otimes c_2 = \mathit{ff}$. Let us assume that $d \Vdash (\eta \otimes \delta)$ and $d \neq \mathit{ff}$. Then:

$$\begin{aligned}
\mathit{prefix}(r\Downarrow_d) &= \mathit{prefix}(d \cdot c_1 \otimes c_2)\\
&= \mathit{prefix}(d \cdot \mathit{ff})\\
&= \{d \cdot s \mid s \in \mathcal{B}^{ss}[\![D . A_1' \parallel A_2']\!]_{\mathit{ff}}\}\\
&\qquad [\text{by Rule } \mathbf{R7}]\\
&\subseteq \mathcal{B}^{ss}[\![D . A_1 \parallel A_2]\!]_d
\end{aligned}$$

In fact, also the second component of the behavior is the store ff. This case represents the situation in which the contribution of the two conditional traces results in an inconsistent conditional trace.
If $d = \mathit{ff}$ we proceed similarly to the previous case.
If $d \not\Vdash (\eta \otimes \delta)$, then the operator $\Downarrow_d$ is undefined on r, thus we have that $\varnothing \subseteq \mathcal{B}^{ss}[\![D . A_1 \parallel A_2]\!]_d$.

5. Let $r = (\eta^+, \eta^- \cup \delta^-) \rightarrowtail c_1 \cdot (r_1' \mathrel{\bar{\parallel}} r_2'{\downarrow_{c_1}})$ be a conditional trace in $\mathcal{A}[\![A_1 \parallel A_2]\!]_{\mathcal{F}[\![D]\!]}$ such that $r_1 = \eta \rightarrowtail c_1 \cdot r_1' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$, $r_2 = \mathit{stutt}(\delta^-) \cdot r_2' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$ with r_2' that recursively belongs to $\mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$ and for all $h^- \in \delta^-$, $\eta^+ \nvdash h^-$. Let us assume that $d \Vdash (\eta^+, \eta^- \cup \delta^-)$. Then,

$$\begin{aligned}
&\mathit{prefix}(r\Downarrow_d) =\\
&= \mathit{prefix}(\{d \cdot (r_1' \mathrel{\bar{\parallel}} r_2'{\downarrow_{c_1}})\Downarrow_{d\otimes c_1} \mid r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]},\ r_2' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\})\\
&\qquad [\, c_1 \text{ is already contained in the stores of } r_1 \,]\\
&= \mathit{prefix}(\{d \cdot (r_1' \mathrel{\bar{\parallel}} r_2'){\downarrow_{c_1}}\Downarrow_{d\otimes c_1} \mid r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]},\ r_2' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\})\\
&\qquad [\text{by Lemma A.6}]\\
&= \mathit{prefix}(\{d \cdot (r_1' \mathrel{\bar{\parallel}} r_2')\Downarrow_{d\otimes c_1} \mid r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]},\ r_2' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\})\\
&\qquad [\text{by Lemma A.8}]\\
&= \mathit{prefix}(\{d \cdot (r_1'\Downarrow_{d\otimes c_1} \mathrel{\breve{\parallel}} r_2'\Downarrow_{d\otimes c_1}) \mid r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]},\ r_2' \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\})\\
&\qquad [\text{by Equation (3.1)}]\\
&= \{\epsilon, d\} \cup \{d \cdot (s_1' \mathrel{\breve{\parallel}} s_2') \mid s_1' \in \mathit{prefix}(\mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]}\Downarrow_{d\otimes c_1}),\ s_2' \in \mathit{prefix}(\mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}\Downarrow_{d\otimes c_1})\}\\
&\qquad [\text{by Inductive Hypothesis}]\\
&\subseteq \{\epsilon, d\} \cup \{d \cdot (s_1' \mathrel{\breve{\parallel}} s_2') \mid s_1' \in \mathcal{B}^{ss}[\![D . A_1']\!]_{d\otimes c_1},\ s_2' \in \mathcal{B}^{ss}[\![D . A_2]\!]_{d\otimes c_1}\}\\
&\qquad [\text{by Definition A.7 and by Definition 3.1}]\\
&\subseteq \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathcal{B}^{ss}[\![D . A_1' \parallel A_2]\!]_{d\otimes c_1}\}\\
&\qquad [\text{by Rule } \mathbf{R8}]\\
&\subseteq \mathcal{B}^{ss}[\![D . A_1 \parallel A_2]\!]_d
\end{aligned}$$

If $d = \mathit{ff}$, then we proceed as in the first case (Point 3).
If $d \Vdash\!\!\!/\ (\eta^+, \eta^- \cup \delta^-)$, then we have that $\mathit{prefix}(r\Downarrow_d) = \varnothing \subseteq \mathcal{B}^{ss}[\![D \,.\, A_1 \parallel A_2]\!]_d$.

$\sqsupseteq$ In the following, we show that if $s \in \mathcal{B}^{ss}[\![D \,.\, A_1 \parallel A_2]\!]_d$, then $s \in \mathcal{A}[\![A_1 \parallel A_2]\!]_{\mathcal{F}[\![D]\!]}\Downarrow_d$, i.e., we can find a conditional trace $r \in \mathcal{A}[\![A_1 \parallel A_2]\!]_{\mathcal{F}[\![D]\!]}$ such that $s = r\Downarrow_d$. We have four possible cases, depending on the rules defining the operational semantics for the agent.

1. If $\langle A_1,\ d\rangle \rightarrow \langle A_1',\ d_1'\rangle$ and $\langle A_2,\ d\rangle \rightarrow \langle A_2',\ d_2'\rangle$, the behavior of the parallel composition is: $\mathcal{B}^{ss}[\![D \,.\, A_1 \parallel A_2]\!]_d = \{d \cdot s' \mid s' \in \mathcal{B}^{ss}[\![D \,.\, A_1' \parallel A_2']\!]_{d_1' \otimes d_2'}\}$. Let s be an element of that set. By inductive hypothesis we know that there exist the conditional traces $r_1 \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$ and $r_2 \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$ such that $r_1\Downarrow_d = d \cdot s_1'$ and $r_2\Downarrow_d = d \cdot s_2'$, with $s_1' \in \mathcal{B}^{ss}[\![D \,.\, A_1']\!]_d$ and $s_2' \in \mathcal{B}^{ss}[\![D \,.\, A_2']\!]_d$. Now, consider $r = r_1 \mathbin{\overline{\parallel}} r_2$; this conditional trace belongs to $\mathcal{A}[\![A_1 \parallel A_2]\!]_{\mathcal{F}[\![D]\!]}$ whenever r_1 and r_2 are compatible via parallel composition (i.e., $r_1 \mathbin{\overline{\parallel}} r_2$ is a valid conditional trace). We show that $s = (r_1 \mathbin{\overline{\parallel}} r_2)\Downarrow_d$.

   $(r_1 \mathbin{\overline{\parallel}} r_2)\Downarrow_d =$

   [by Lemma A.8]

   $= r_1\Downarrow_d \mathbin{\breve{\parallel}} r_2\Downarrow_d$

   [by Definition 3.26]

   $= (d \cdot s_1') \mathbin{\breve{\parallel}} (d \cdot s_2')$ with $s_1' \in \mathcal{B}^{ss}[\![D \,.\, A_1']\!]_{d_1'}$ and $s_2' \in \mathcal{B}^{ss}[\![D \,.\, A_2']\!]_{d_2'}$

   [by Definition A.7]

   $= d \cdot s_1' \mathbin{\breve{\parallel}} s_2'$ with $s_1' \in \mathcal{B}^{ss}[\![D \,.\, A_1']\!]_{d_1'}$ and $s_2' \in \mathcal{B}^{ss}[\![D \,.\, A_2']\!]_{d_2'}$

   [by Definition A.7 and by Definition 3.1]

   $= d \cdot s'$ with $s' \in \mathcal{B}^{ss}[\![D \,.\, A_1']\!]_{d_1'} \mathbin{\breve{\parallel}} \mathcal{B}^{ss}[\![D \,.\, A_2']\!]_{d_2'}$

   [by Rule **R7** and Equation (A.1)]

   $= d \cdot s'$ with $s' \in \mathcal{B}^{ss}[\![D \,.\, A_1' \parallel A_2']\!]_{d_1' \otimes d_2'}$

   $= s$

2. If $\langle A_1,\ d\rangle \rightarrow \langle A_1',\ d_1'\rangle$ and $\langle A_2,\ d\rangle \not\rightarrow$, then Rule **R8** is applied and we have that $\mathcal{B}^{ss}[\![D \,.\, A_1 \parallel A_2]\!]_d = \{d \cdot s' \mid s' \in \mathcal{B}^{ss}[\![D \,.\, A_1' \parallel A_2]\!]_{d_1'}\}$. Let s be an element of that set. By inductive hypothesis, we know that it exists a conditional trace $r_1 \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$ such that $r_1\Downarrow_d = d \cdot s_1'$ with $s_1' \in \mathcal{B}^{ss}[\![D \,.\, A_1']\!]_d$. Moreover, it exists a conditional trace $r_2 \in \mathcal{A}[\![A_2]\!]_{\mathcal{F}[\![D]\!]}$ such that $r_2\Downarrow_d = d$. We distinguish two cases (corresponding to the two agents that can make the agent A_2 not to proceed) in order to prove that $s = (r_1 \mathbin{\overline{\parallel}} r_2)\Downarrow_d$.

   A_2 = skip In this case, the behavior of the parallel composition is that of A_1 since A_2 makes no contribution to the computation. Then, $(r_1 \mathbin{\overline{\parallel}} \boxtimes)\Downarrow_d = d \cdot s'$ with $s' \in \mathcal{B}^{ss}[\![D \,.\, A_1']\!]_d = \mathcal{B}^{ss}[\![D \,.\, A_1' \parallel A_2]\!]_d$, thus $(r_1 \mathbin{\overline{\parallel}} \boxtimes)\Downarrow_d = s$.

   $A_2 = \sum_{i=1}^{n} \mathsf{ask}(c_i) \rightarrow B_i$ Consider $r_2 = \mathit{stutt}(\{c_1, \ldots, c_n\}) \cdot r_2'$ with $r_2' \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$. We can assume that $d \nvdash c_i$ for all c_i, otherwise, the agent A_2 would proceed.

$$
\begin{aligned}
&(\, r_1 \;\overline{\|}\; \mathit{stutt}(c_1, \ldots, c_n) \cdot r'_2) \Downarrow_d = \\
&\quad = d \cdot (r'_1 \;\overline{\|}\; r'_2) \Downarrow_{d'_1} \text{ with } r'_1 \in \mathcal{A}\llbracket A_1 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \text{ and } r'_2 \in \mathcal{A}\llbracket A_2 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \\
&\qquad\quad [\,\text{by Lemma A.8}\,] \\
&\quad = d \cdot (r'_1 \Downarrow_{d'_1}) \;\breve{\|}\; (r'_2 \Downarrow_{d'_1}) \text{ with } r'_1 \in \mathcal{A}\llbracket A_1 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \text{ and } r'_2 \in \mathcal{A}\llbracket A_2 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \\
&\qquad\quad [\,\text{by Inductive Hypothesis}\,] \\
&\quad = d \cdot s'_1 \;\breve{\|}\; s'_2 \text{ with } s'_1 \in \mathcal{B}^{ss}\llbracket D \,.\, A'_1 \rrbracket_{d'_1} \text{ and } s'_2 \in \mathcal{B}^{ss}\llbracket D \,.\, A_2 \rrbracket_{d'_1} \\
&\qquad\quad [\,\text{by Definition A.7 and by Definition 3.1}\,] \\
&\quad = d \cdot s' \text{ with } s' \in \mathcal{B}^{ss}\llbracket D \,.\, A'_1 \rrbracket_{d'_1} \;\breve{\|}\; \mathcal{B}^{ss}\llbracket D \,.\, A_2 \rrbracket_{d'_1} \\
&\qquad\quad [\,\text{by Rule } \mathbf{R7}\text{, Rule } \mathbf{R8} \text{ and Equation (A.1)}\,] \\
&\quad = d \cdot s' \text{ with } s' \in \mathcal{B}^{ss}\llbracket D \,.\, A'_1 \parallel A_2 \rrbracket_{d'_1} \\
&\quad = s
\end{aligned}
$$

3. If $\langle A_1,\ d\rangle \not\to$ and $\langle A_2,\ d\rangle \to \langle A'_2, d'_2\rangle$, then the situation is symmetric to the previous case, so we can conclude that $\mathcal{A}\llbracket A_1 \parallel A_2 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \Downarrow_d \supseteq \mathcal{B}^{ss}\llbracket D \,.\, A_1 \parallel A_2 \rrbracket_d$.
4. Finally, if $\langle A_1,\ d\rangle \not\to$ and $\langle A_2,\ d\rangle \not\to$, then we can reason similarly to Point 2, considering, for both A_1 and A_2, the two cases in which they cannot proceed. We can conclude that $\mathcal{B}^{ss}\llbracket D \,.\, A_1 \parallel A_2 \rrbracket_d = \{\epsilon,\ d\} \subseteq \mathcal{A}\llbracket A_1 \parallel A_2 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \Downarrow_d$.

$\boldsymbol{\exists x\, A_1}$| We prove the two directions independently.

$\subseteq$| We show that: $\mathit{prefix}(\mathcal{A}\llbracket \exists x\, A_1 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \Downarrow_d) \subseteq \mathcal{B}^{ss}\llbracket D \,.\, \exists x\, A_1 \rrbracket_d$. Let $r = \overline{\exists}_x\, r_1$ such that $r_1 \in \mathcal{A}\llbracket A_1 \rrbracket_{\mathcal{F}\llbracket D \rrbracket}$ and r_1 is x-self-sufficient. We show that the prefixes of $(\overline{\exists}_x\, r_1) \Downarrow_d$ are included in the behavior $\mathcal{B}^{ss}\llbracket D \,.\, \exists x\, A_1 \rrbracket_d$, by induction on the length of r_1.

***length*$\boldsymbol{(r_1) = 0}$**| If $r_1 = \epsilon$ the statement directly holds.

***length*$\boldsymbol{(r_1) \geq 1}$**| We distinguish three cases depending on the form of the first state:

1. If $r_1 = \boxtimes$, then $\boxtimes \Downarrow_d = d$ which belongs to the behavior.
2. Consider $r_1 = \eta \rightarrowtail l \cdot r'_1$ with $r'_1 \in \mathcal{A}\llbracket A'_1 \rrbracket_{\mathcal{F}\llbracket D \rrbracket}$ and such that, by inductive hypothesis, there exists a transition $\langle A_1, d\rangle \to \langle A'_1, d'\rangle$. Since r_1 is x-self-sufficient, also r'_1 is x-self-sufficient. Now, assume that $d \Vdash \exists_x\, \eta$ and $d \neq \mathit{ff}$:

$$
\begin{aligned}
&\mathit{prefix}(r \Downarrow_d) = \\
&\qquad = \mathit{prefix}(\{\overline{\exists}_x(\eta \rightarrowtail l \cdot r'_1) \Downarrow_d \mid r'_1 \in \mathcal{A}\llbracket A'_1 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \text{ and } r'_1 \ x\text{-self-sufficient}\}) \\
&\qquad\qquad [\,\text{by Definition 3.26}\,] \\
&\qquad = \mathit{prefix}(\{d \cdot (\overline{\exists}_x\, r'_1) \Downarrow_{d \otimes \exists_x l} \mid r'_1 \in \mathcal{A}\llbracket A'_1 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \text{ and } r'_1 \ x\text{-self-sufficient}\}) \\
&\qquad\qquad [\, r'_1 \in \mathcal{A}\llbracket A'_1 \rrbracket_{\mathcal{F}\llbracket D \rrbracket} \text{ and } r'_1 \ x\text{-self-sufficient}\,] \\
&\qquad = \mathit{prefix}(\{d \cdot s \mid s \in (\mathcal{A}\llbracket \exists x\, A'_1 \rrbracket_{\mathcal{F}\llbracket D \rrbracket}) \Downarrow_{d \otimes \exists_x l}\}) \\
&\qquad\qquad [\,\text{by Equation (3.1)}\,]
\end{aligned}
$$

$$
\begin{aligned}
&= \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathit{prefix}(\mathcal{A}[\![\exists x\, A_1']\!]_{\mathcal{F}[\![D]\!]} \Downarrow_{d \otimes \exists_x l})\} \\
&\qquad [\,\text{by Inductive Hypothesis}\,] \\
&\subseteq \{\epsilon, d\} \cup \{d \cdot s \mid s \in \mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1']\!]_{d \otimes \exists_x l}\} \\
&\qquad [\,\text{by Rule } \mathbf{R9}\,] \\
&\subseteq \mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1]\!]_d
\end{aligned}
$$

If $d = \mathit{ff}$ we have that $\langle \exists x\, A_1, \mathit{ff}\rangle \not\to$, thus $\mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1]\!]_{\mathit{ff}} = \{\epsilon, \mathit{ff}\}$. On the other hand, since $d \Vdash \exists_x \eta$ we have that $\mathit{prefix}(r\Downarrow_{\mathit{ff}}) = \{\epsilon, \mathit{ff}\}$ which corresponds to the small-step behavior $\mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1]\!]_{\mathit{ff}}$.
If $d \not\Vdash \exists_x \eta$, then the operator $\Downarrow_d$ is undefined for the conditional trace, thus $\mathit{prefix}(r\Downarrow_d) = \varnothing \subseteq \mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1]\!]_d$.

3. Consider $r_1 = \mathit{stutt}(\{c_1, \ldots, c_n\}) \cdot r_1'$, with $r_1' \in \mathcal{A}[\![\sum_{i=1}^{i} \mathsf{ask}(n_i) \to B_i]\!]_{\mathcal{F}[\![D]\!]}$. Assume that it exists no index $1 \le j \le n$ such that $d \vdash c_j$. This implies that $d \vdash \exists_x c_j$. Then, we have

$$
\begin{aligned}
\mathit{prefix}(r\Downarrow_d) &= \mathit{prefix}(\bar{\exists}_x(\mathit{stutt}(\{c_1, \ldots, c_n\}) \cdot r_1')\Downarrow_d) \\
&= \mathit{prefix}(\mathit{stutt}(\{\exists_x c_1, \ldots, \exists_x c_n\}) \cdot \bar{\exists}_x r_1' \Downarrow_d) \\
&\qquad [\,\text{by Definition 3.26}\,] \\
&= d \subseteq \mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1]\!]_d
\end{aligned}
$$

If it exists an index j such that $d \vdash c_j$, then $r\Downarrow_d$ is undefined, thus $\mathit{prefix}(r\Downarrow_d) = \varnothing \subseteq \mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1]\!]_d$.

$\supseteq$ From Rule **R9**, we know that, if $d \ne \mathit{ff}$, then $\mathcal{B}^{ss}[\![D\,.\,A_1]\!]_{l \otimes \exists_x d} = l' \cdot \mathcal{B}^{ss}[\![D\,.\,A_1']\!]_d$, where l and l' are local stores. Moreover, $l = \mathit{tt}$ because it is the initial (local) store for A_1. In the following, we show that $d \cdot \mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1']\!]_{d \otimes \exists_x l} \in \mathcal{A}[\![\exists x\, A_1]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_d$, i.e., it exists a trace $r \in \mathcal{A}[\![\exists x\, A_1]\!]_{\mathcal{F}[\![D]\!]}$ such that $r\Downarrow_d = d \cdot s$ with $s \in \mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1']\!]_{d \otimes \exists_x l}$. By inductive hypothesis, we know that $\mathcal{B}^{ss}[\![D . A_1]\!]_{\exists_x d} \subseteq \mathit{prefix}(\mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]} \Downarrow_{\exists_x d})$, and by Rule **R9** it holds that there exists a conditional trace $r_1 \in \mathcal{A}[\![A_1]\!]_{\mathcal{F}[\![D]\!]}$ such that $r_1 \Downarrow_{\exists_x d} = \exists_x d \cdot \mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1']\!]_{l'}$.
Now, r_1 is x-self-sufficient since the only external information is provided by $\exists_x d$, which in fact does not contain information about x. Moreover, r_1 is of the form $\eta \rightarrowtail l' \cdot r_1'$ with $r_1' \in \mathcal{A}[\![A_1']\!]_{\mathcal{F}[\![D]\!]}$. Therefore, it exists a conditional trace $r \in \mathcal{A}[\![\exists x\, A_1]\!]_{\mathcal{F}[\![D]\!]}$ such that $r = \bar{\exists}_x r_1$. Then,

$$
\begin{aligned}
r\Downarrow_d &= (\bar{\exists}_x \eta \rightarrowtail l' \cdot r_1')\Downarrow_d \\
&= (\exists_x \eta \rightarrowtail \exists_x l' \cdot \bar{\exists}_x r_1')\Downarrow_d \\
&\qquad [\,\text{by Definition 3.26}\,] \\
&= d \cdot (\bar{\exists}_x r_1')\Downarrow_{\exists_x l' \otimes d} \\
&\qquad [\,\text{by Definition 3.26}\,] \\
&= d \cdot s \qquad \text{with } s \in \mathcal{A}[\![\exists x\, A_1']\!]_{\mathcal{F}[\![D]\!]} \Downarrow_{\exists_x l' \otimes d} \\
&\qquad [\,\text{by Inductive Hypothesis}\,] \\
&= d \cdot s \qquad \text{with } s \in \mathcal{B}^{ss}[\![D\,.\,\exists x\, A_1']\!]_{\exists_x l' \otimes d}
\end{aligned}
$$

If $d = \mathit{ff}$, then we have that $\mathit{prefix}(r{\Downarrow_{\mathit{ff}}}) = \{\epsilon, \mathit{ff}\}$, which corresponds to the small-step behavior $\mathcal{B}^{ss}[\![D \,.\, \exists x\, A_1]\!]_{\mathit{ff}}$ since the transition relation $\rightarrow$ is not defined for $\langle \exists x\, A_1, \mathit{ff}\rangle$.

<u>***p(x)***</u>| If $d \neq \mathit{ff}$, then

$$\begin{aligned}
\mathcal{A}[\![p(x)]\!]_{\mathcal{F}[\![D]\!]}{\Downarrow_d} &= \{(tt,\varnothing) \rightarrowtail tt \cdot r' \mid r' \in \mathcal{F}[\![D]\!](p(x))\}{\Downarrow_d}\\
&\qquad [\text{since } \mathcal{F}[\![D]\!] = \mathcal{D}[\![D]\!]_{\mathcal{F}[\![D]\!]}\,]\\
&= \{(tt,\varnothing) \rightarrowtail tt \cdot r' \mid r' \in \mathcal{D}[\![D]\!]_{\mathcal{F}[\![D]\!]}(p(x))\}{\Downarrow_d}\\
&\qquad [\text{by Definition 3.21}]\\
&= \{(tt,\varnothing) \rightarrowtail tt \cdot r' \mid r' \in \mathcal{A}[\![B]\!]_{\mathcal{F}[\![D]\!]},\ p(\vec{x}) \coloneq B \in D\}{\Downarrow_d}\\
&= \{d \cdot s' \mid s' \in (\mathcal{A}[\![B]\!]_{\mathcal{F}[\![D]\!]}){\Downarrow_d},\ p(\vec{x}) \coloneq B \in D\}\\
&\qquad [\text{by Inductive Hypothesis}]\\
&= \{d \cdot s' \mid s' \in \mathcal{B}^{ss}[\![D \,.\, B]\!]_d,\ p(\vec{x}) \coloneq B \in D\}\\
&\qquad [\text{by Rule } \mathbf{R10}]\\
&= \mathcal{B}^{ss}[\![D \,.\, p(x)]\!]_d
\end{aligned}$$

Notice that, in the second last equality, the structural induction hypothesis cannot be applied because B can be structurally greater than $p(\vec{x})$. For this reason, we have to introduce a second induction on the number of $p(\vec{x})$ present on B. If B does not contain any process call $p(\vec{x})$, then we can directly apply structural induction. Otherwise, if the agent contains one process call $p(\vec{x})$, it is sufficient to replace the call with the body of the declaration. In this way, B has less process calls $p(\vec{x})$ than A and we can apply the inductive hypothesis.

If $d = \mathit{ff}$, then the transition relation $\rightarrow$ is not defined for the configuration $\langle p(x), \mathit{ff}\rangle$, hence

$$\begin{aligned}
\mathcal{B}^{ss}[\![D \,.\, p(x)]\!]_{\mathit{ff}} &= \{\epsilon, \mathit{ff}\}\\
&= \mathit{prefix}(\{(tt,\varnothing) \rightarrowtail tt \cdot r'{\Downarrow_{\mathit{ff}}} \mid r' \in \mathcal{F}[\![D]\!](p(x))\})\\
&= \mathcal{A}[\![p(x)]\!]_{\mathcal{F}[\![D]\!]}{\Downarrow_{\mathit{ff}}}
\end{aligned}$$

Proof of Theorem 3.28.

$\Rightarrow$| Follows directly from Definition 3.26.

$\Leftarrow$| It is sufficient to show that $\mathcal{P}[\![P_1]\!] \neq \mathcal{P}[\![P_2]\!] \Rightarrow \exists \bar{c} \in \mathbf{C}.\ \mathcal{P}[\![P_1]\!]{\Downarrow_{\bar{c}}} \neq \mathcal{P}[\![P_2]\!]{\Downarrow_{\bar{c}}}$. Without loss of generality, assume that $\mathcal{P}[\![P_1]\!] \supset \mathcal{P}[\![P_2]\!]$, thus, it exists a conditional trace $r_1 \in \mathcal{P}[\![P_1]\!]$ that does not belong to $\mathcal{P}[\![P_2]\!]$. We can distinguish two cases: $\mathcal{P}[\![P_2]\!]$ is empty or $\mathcal{P}[\![P_2]\!]$ contains at least one conditional trace.

If $\mathcal{P}[\![P_2]\!] = \varnothing$, then $\mathcal{P}[\![P_2]\!]{\Downarrow_c}$ is empty for any possible $c \in \mathbf{C}$. Now, if we choose $\bar{c}$ to be the merge ($\otimes$) of all the positive conditions occurring in r_1, then $r_1{\Downarrow_{\bar{c}}}$ is a valid trace. Therefore, $\mathcal{P}[\![P_1]\!]{\Downarrow_{\bar{c}}} \supseteq \{r_1{\Downarrow_{\bar{c}}}\} \neq \varnothing$.

If $\mathcal{P}[\![P_2]\!] \neq \varnothing$, by the initial assumptions, it exists a conditional trace $r_2 \in \mathcal{P}[\![P_2]\!]$ such that $r_1 \neq r_2$. Without loss of generality, assume that $\mathit{length}(r_1) \leq \mathit{length}(r_2)$ and that r_1 differs from r_2 at position k, with

$k \in [1, length(r_1)]$. The index k is guaranteed to exist.[7] We consider the six possible cases, corresponding to the possible forms of the conditional state at position k, in order to prove that there exists a store $\bar{c}$ such that $\mathcal{P}[\![P_1]\!]\Downarrow_{\bar{c}} \neq \mathcal{P}[\![P_2]\!]\Downarrow_{\bar{c}}$. In the following, the stores $\bar{c}_1$ and $\bar{c}_2$ correspond to the merge ($\otimes$) of all the positive conditions occurring in r_1 and r_2, respectively.

1. Let be $(\eta_1^+, \eta_1^-) \rightarrowtail d_1$ and $(\eta_2^+, \eta_2^-) \rightarrowtail d_2$ the k-th conditional tuple in r_1 and r_2, respectively. There are three possible ways in which these two tuples can differ:
   (a) Case $\eta_1^+ \neq \eta_2^+$. Let us assume that $\eta_1^+ \vdash \eta_2^+$ and $\eta_2^+ \nvdash \eta_1^+$. Notice that r_1 has to come from the semantics of an ask or a now construct since they are the only *tccp* agents that can add information to the positive condition (see Definition 3.16). Hence, there exists also a conditional trace $\bar{r}_1 \in \mathcal{P}[\![P_1]\!]$ in which η_1^+ occurs in a negative condition (corresponding to the else branch of a now agent) or in a *stutt* construct (corresponding to the suspension of an ask agent) of the sequence. There are two cases in which $\bar{r}_1$ does not exists, but both are in contradiction with the hypothesis: (1) when $\eta_1^+ = tt$, but this contradicts $\eta_2^+ \nvdash \eta_1^+$ or (2) when a constraint d stronger than η_1^+ ($d \vdash \eta_1^+$) is propagated. In this last case, the trace $\bar{r}_1$ does not exists since the condition is in contradiction with the propagated store. However, since $\eta_1^+ \vdash \eta_2^+$, it follows that d entails also η_2^+ ($d \otimes \eta_1^+ = d \otimes \eta_2^+ = d$). Therefore, the propagation of d makes r_1 and r_2 equal. Since they were supposed to be different only at this point, this is a contradiction with the hypothesis $r_1 \neq r_2$. Therefore, $\bar{r}_1$ exists and belongs to $\mathcal{P}[\![P_1]\!]$. Furthermore, $\bar{r}_1$ differs from any trace in $\mathcal{P}[\![P_2]\!]$ for at least the negative part of a condition or the body of a *stutt*, otherwise, reasoning in a similar way as above, r_1 would also belong to $\mathcal{P}[\![P_2]\!]$, and this is not possible.
   If $\eta_1^+ \nvdash \eta_2^+$ and $\eta_2^+ \vdash \eta_1^+$, we can reason in a symmetric way, thus concluding that it exists $\bar{r}_2 \in \mathcal{P}[\![P_2]\!]$ that differs from any trace in $\mathcal{P}[\![P_1]\!]$ for at least the negative part of a condition or the body of a *stutt*.
   Finally, if $\eta_1^+ \nvdash \eta_2^+$ and $\eta_2^+ \nvdash \eta_1^+$, we can reason as before and deduce that there exist two traces $\bar{r}_1 \in \mathcal{P}[\![P_1]\!]$ and $\bar{r}_2 \in \mathcal{P}[\![P_2]\!]$, which contains respectively η_1^+ and η_2^+ in the negative part of the condition, and such that $\bar{r}_1 \notin \mathcal{P}[\![P_2]\!]$ and $\bar{r}_2 \notin \mathcal{P}[\![P_1]\!]$.
   In case $\bar{r}_1$ (respectively $\bar{r}_2$) comes from an ask agent we remand to the Points 2, 3 and 4 where we deal with the conditional traces containing *stutt* constructs. Otherwise, if r_1 comes from a now agent we can reduce to the following Point 1b where we deal with the negative part of the conditions ($\eta_1^- \neq \eta_2^-$).
   (b) Case $\eta_1^- \neq \eta_2^-$. Let us first assume that $\eta_1^- \subset \eta_2^-$. This means that the store at position k in r_2 has to satisfy a stronger condition than the one in r_1. Let $\bar{c} := \bar{c}_1 \otimes h_2^-$, with $h_2^- \in \eta_2^- \smallsetminus \eta_1^-$. Under these

[7]There are two cases in which k does not exist, but both are in contradiction with the initial hypothesis: (1) $r_1 = r_2$ or (2) one of the traces is a prefix of the other.

conditions, $r_1{\Downarrow_{\bar{c}}}$ computes a behavioral timed trace whereas $r_2{\Downarrow_{\bar{c}}}$ computes no trace since, at position k, $\bar{c}$ entails one of the stores in the negative condition.
For the case in which $\eta_2^- \subset \eta_1^-$ we choose $c = \bar{c}_2 \otimes h_1^-$, with $h_1^- \in \eta_1^- \setminus \eta_2^-$ and reason in an symmetric way.
Finally, if $\eta_1^- \nsubseteq \eta_2^-$ and $\eta_2^- \nsubseteq \eta_1^-$, we can choose indifferently $\bar{c} = \bar{c}_1 \otimes h_2^-$ or $\bar{c} = \bar{c}_2 \otimes h_1^-$ and conclude that $r_1{\Downarrow_{\bar{c}}}$ computes a behavioral timed trace but $r_2{\Downarrow_{\bar{c}}}$ is not defined, or vice-versa.
Thus, we can conclude that $\mathcal{P}[\![P_1]\!]{\Downarrow_{\bar{c}}} \neq \mathcal{P}[\![P_2]\!]{\Downarrow_{\bar{c}}}$.

(c) Case $d_1 \neq d_2$. Consider $\bar{c} = \bar{c}_1 = \bar{c}_2$. There are two possible cases. Assume first that $\bar{c} \nVdash d_1$ and $\bar{c} \nVdash d_2$. Both r_1 and r_2 must be *compatible* with their own conditions, thus, being the store monotonic, it happens that $r_1{\Downarrow_{\bar{c}}}$ and $r_2{\Downarrow_{\bar{c}}}$ are both defined. Moreover, we know that $\eta_1^+ = \eta_2^+$ and from Property A.1 $d_1 \vdash \eta_1^+$ and $d_2 \vdash \eta_1^+$. Since $\bar{c} \nVdash d_1$ and $\bar{c} \nVdash d_2$, we can conclude that in $r_1{\Downarrow_{\bar{c}}}$ at position k we have the store d_1, whereas in $r_2{\Downarrow_{\bar{c}}}$ at the same position we find the store d_2 that is different from d_1 by the initial assumptions. Thus $r_1{\Downarrow_{\bar{c}}} \neq r_2{\Downarrow_{\bar{c}}}$. Assume now that $\bar{c}$ contains more information than the store d_1 (respectively d_2). Then, we know that, at certain point in r_1 (respectively r_2), the positive condition is stronger than d_1 (respectively d_2). Therefore, we can reason as in Point 1a where $\eta_1^+ \neq \eta_2^+$ and r_1 (respectively r_2) are produced by the semantics of an ask or a now agent.

2. Let $\mathit{stutt}(\eta_1^-)$ (respectively $\mathit{stutt}(\eta_2^-)$) be the k-th conditional state in r_1 (respectively r_2). It is sufficient to proceed as in Point 1b to show that there exists a store $\bar{c}$ such that $r_1{\Downarrow_{\bar{c}}}$ is well defined while $r_2{\Downarrow_{\bar{c}}}$ is not. For instance, if $\eta_1^- \subset \eta_2^-$ we set $\bar{c} = \bar{c}_1 \otimes h_2^-$, with $h_2^- \in \eta_2^- \setminus \eta_1^-$. It is easy to notice that $r_1{\Downarrow_{\bar{c}}}$ computes a behavioral timed trace but $r_2{\Downarrow_{\bar{c}}}$ recovers no trace since at position k the constraint h_2^- belongs to the negative part of the condition. Therefore, $\mathcal{P}[\![P_1]\!]{\Downarrow_{\bar{c}}} \neq \mathcal{P}[\![P_2]\!]{\Downarrow_{\bar{c}}}$.

3. Let $\eta_1 \rightarrowtail d_1$ be the k-th conditional tuple in r_1 and $\mathit{stutt}(\eta_2^-)$ the k-th element in r_2. Consider $\bar{c} = \bar{c}_1$. Up to instant k, $r_1{\Downarrow_{\bar{c}}}$ and $r_2{\Downarrow_{\bar{c}}}$ coincide and, as r_1 and r_2 differ only at position k, $\bar{c}$ satisfies all the conditions in r_1 and in r_2 till up that position. The behavioral timed trace $r_2{\Downarrow_{\bar{c}}}$ ends at position k since a *stutt* has been encountered (see Definition 3.26). However, since r_1 is maximal, $r_1{\Downarrow_{\bar{c}}}$ does not end at position k but continues with at least another state, otherwise we would have found an ending symbol $\boxtimes$. In conclusion, $r_2{\Downarrow_{\bar{c}}}$ is at least one store longer than $r_1{\Downarrow_{\bar{c}}}$ and we conclude that $\mathcal{P}[\![P_1]\!]{\Downarrow_{\bar{c}}} \neq \mathcal{P}[\![P_2]\!]{\Downarrow_{\bar{c}}}$.

4. If $\eta_2 \rightarrowtail d_2$ is the k-th element in r_2 and $\mathit{stutt}(\eta_1^-)$ that in r_1, then the proof is symmetric to Point 3.

5. Let $\boxtimes$ and $\eta_2 \rightarrowtail d_2$ be the k-th states of r_1 and r_2, respectively. We can reason similarly to Point 4, choosing $\bar{c} = \bar{c}_2$. By hypothesis, r_1 and r_2 differ only at position k, thus, $r_1{\Downarrow_{\bar{c}}}$ and $r_2{\Downarrow_{\bar{c}}}$ compute the same behavioral timed trace up to position k-th. However, while $r_1{\Downarrow_{\bar{c}}}$ stops at instant k (an ending symbol $\boxtimes$ is found), $r_2{\Downarrow_{\bar{c}}}$ is at least one store longer. Thus, $\mathcal{P}[\![P_1]\!]{\Downarrow_{\bar{c}}} \neq \mathcal{P}[\![P_2]\!]{\Downarrow_{\bar{c}}}$.

6. Let $\boxtimes$ be the k-th element of r_1 and $\mathit{stutt}(\eta_2)$ the conditional state occurring in r_2 at the same position. We set $\bar{c} = \bar{c}_1 \otimes h_2^-$, with $h_2^- \in \eta_2^- \smallsetminus \eta_1^-$. In this way, $r_1 \Downarrow_{\bar{c}}$ is defined but $r_2 \Downarrow_{\bar{c}}$ computes no trace, since, at position k, the constraint h_2^- is required not to be entailed by the current store. Thus, $\mathcal{P}[\![P_1]\!] \Downarrow_{\bar{c}} \neq \mathcal{P}[\![P_2]\!] \Downarrow_{\bar{c}}$.

In conclusion, we can always choose an adequate $\bar{c}$ which differentiates $\mathcal{P}[\![P_1]\!] \Downarrow_{\bar{c}}$ from $\mathcal{P}[\![P_2]\!] \Downarrow_{\bar{c}}$.

Proof of Proposition 3.29.

$\boxed{\Rightarrow}$ Straightforward.

$\boxed{\Leftarrow}$ By Definition 3.21, $\mathcal{P}[\![D_1 \,.\, A]\!] = \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D_1]\!]}$ and $\mathcal{P}[\![D_2 \,.\, A]\!] = \mathcal{A}[\![A]\!]_{\mathcal{F}[\![D_2]\!]}$. We have to check that $\mathcal{F}[\![D_1]\!] = \mathcal{F}[\![D_2]\!]$. The only case depending on the interpretation is when $A = p(x)$; by hypothesis:

$$\begin{aligned}\mathcal{A}[\![p(x)]\!]_{\mathcal{F}[\![D_1]\!]} &= \bigsqcup \{ (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdot r \mid r \in \mathcal{F}[\![D_1]\!](p(x)) \} \\ &= \bigsqcup \{ (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdot r \mid r \in \mathcal{F}[\![D_2]\!](p(x)) \} = \mathcal{A}[\![p(x)]\!]_{\mathcal{F}[\![D_2]\!]}\end{aligned}$$

We have to check that $\mathcal{F}[\![D_1]\!](p(x))$ and $\mathcal{F}[\![D_2]\!](p(x))$ coincide for each $p(x) \in \mathbb{MGC}$. Since $\mathcal{F}[\![D_1]\!]$ (respectively $\mathcal{F}[\![D_2]\!]$) is the least fixpoint of $\mathcal{D}[\![D_1]\!]_\perp$ (respectively $\mathcal{D}[\![D_2]\!]_\perp$), we know that it contains only information regarding the procedure calls in D_1 (respectively D_2). So we can conclude that $\mathcal{F}[\![D_1]\!] = \mathcal{F}[\![D_2]\!]$.

Proof of Corollary 3.30.

Consider D_1, $D_2 \in \mathbb{D}^{\Pi}_{\mathbf{C}}$:

$$\begin{aligned} D_1 \approx_{\mathcal{F}} D_2 &\Leftrightarrow \mathcal{F}[\![D_1]\!] = \mathcal{F}[\![D_2]\!] \\ &\qquad [\text{by Proposition 3.29}] \\ &\Leftrightarrow \forall A \in \mathbb{A}^{\Pi}_{\mathbf{C}}.\, \mathcal{P}[\![D_1 \,.\, A]\!] = \mathcal{P}[\![D_2 \,.\, A]\!] \\ &\qquad [\text{by Theorem 3.28}] \\ &\Leftrightarrow \forall A \in \mathbb{A}^{\Pi}_{\mathbf{C}} \forall c \in \mathbf{C}.\, \mathit{prefix}(\mathcal{P}[\![D_1 \,.\, A]\!] \Downarrow_c) = \mathit{prefix}(\mathcal{P}[\![D_2 \,.\, A]\!] \Downarrow_c) \\ &\qquad [\text{by Theorem 3.27}] \\ &\Leftrightarrow \forall A \in \mathbb{A}^{\Pi}_{\mathbf{C}} \forall c \in \mathbf{C}.\, \mathcal{B}^{ss}[\![D_1 \,.\, A]\!]_c = \mathcal{B}^{ss}[\![D_2 \,.\, A]\!]_c \\ &\Leftrightarrow D_1 \approx_{ss} D_2 \end{aligned}$$

A.2 Proofs of Section 4

Lemma A.9 $(\mathbb{M}, \sqsubseteq, \bigsqcup, \bigsqcap, \mathbf{M}, \{\epsilon\}) \underset{\alpha_{io}}{\overset{\gamma_{io}}{\leftrightarrows}} (\mathbb{IO}, \subseteq, \bigcup, \bigcap, \mathbf{IO}, \varnothing)$

Proof.

1. The function α_{io} is monotonic. Let $R_1, R_2 \in \mathbb{M}$ such that $R_1 \sqsubseteq R_2$, thus, $\alpha_{io}(R_1) \subseteq \alpha_{io}(R_2)$. Otherwise, if there exists an input-output pair belonging to $\alpha_{io}(R_1)$ but not to $\alpha_{io}(R_2)$, this would mean that the associated sequence belongs to R_1 and not to R_2 and this contradicts the hypothesis.

2. The function γ_{io} is monotonic. Let $P_1, P_2 \in \mathbb{IO}$ such that $P_1 \subseteq P_2$. If $\gamma_{io}(P_1) \nsubseteq \gamma_{io}(P_2)$ there would exists $r_1 \in \gamma_{io}(P_1)$ but not a sequence $r_2 \in \gamma_{io}(P_1)$ that extends r_1 (r_1 is a prefix of r_2). It is easy to see that this situation is impossible since, by the definition of γ_{io}, r_1 has to belong also to $\gamma_{io}(P_2)$ (since $P_1 \subseteq P_2$) and r_1 trivially extends itself.

3. $(\gamma_{io} \circ \alpha_{io})$ is extensive, i.e., for all $R \in \mathbb{M}. R \sqsubseteq \gamma_{io}(\alpha_{io}(R))$. We show that $r \in R \Rightarrow r \in \gamma_{io}(\alpha_{io}(R))$, we distinguish three cases:
   (a) If $r = \eta_1 \rightarrowtail c_1 \cdot \dots \cdot \eta_n \rightarrowtail c_n \cdot \boxtimes$ we have that $\alpha_{io}(R) \supseteq \{\langle c_0, \mathit{fin}(c)\rangle \mid c_0 \in \mathbf{C}$ and $\mathit{last}(r{\Downarrow}_{c_0}) = c\}$. Thus, by (4.2) it follows that $r \in \gamma_{io}(\alpha_{io}(r))$.
   (b) If $r = \eta_1 \rightarrowtail c_1 \cdot \dots \cdot \mathit{stutt}(\eta_n^-) \cdot \dots$ we have that $\alpha_{io}(R) \supseteq \{\langle c_0, \mathit{fin}(c)\rangle \mid c_0 \in \mathbf{C}$ and $\mathit{last}(r{\Downarrow}_{c_0}) = c\}$. From (4.2) it follows that $r \in \gamma_{io}(\alpha_{io}(r))$.
   (c) If $r = \eta_1 \rightarrowtail c_1 \dots \eta_n \rightarrowtail c_n \dots$ is an infinite sequence that does not contain any *stutt*, then we have that $\alpha_{io}(R) \supseteq \{\langle c_0, \mathit{inf}(c)\rangle \mid c_0 \in \mathbf{C}$, $r{\Downarrow}_{c_0} = c'_0 \dots c'_i \dots$, and $\otimes_{i \geq 0} c'_i = c\}$. By (4.2) we have that $r \in \gamma_{io}(\alpha_{io}(r))$.

4. $(\alpha_{io} \circ \gamma_{io})$ is the identity for $\mathbb{IO}$, i.e., given $P \in \mathbb{IO}$. $P = \alpha_{io}(\gamma_{io}(P))$. We first show that $p \in P \Rightarrow p \in \alpha_{io}(\gamma_{io}(P))$.
   (a) If $p = \langle c_0, \mathit{fin}(c_n)\rangle$ then $\gamma_{io}(P)$ contains all the conditional traces r such that $\mathit{last}(r{\Downarrow}_{c_0}) = c_n$. By (4.1), $p \in \alpha_{io}(\gamma_{io}(P))$.
   (b) If $p = \langle c_0, \mathit{inf}(c)\rangle$ then $\gamma_{io}(P)$ contains all the conditional state sequences r such that $r{\Downarrow}_{c_0} = c_0 \dots c_i \dots$ and $\otimes_{i \geq 0} = c$. By (4.1), $p \in \alpha_{io}(\gamma_{io}(P))$.

   Now we show the other inclusion i.e., $p \in \alpha_{io}(\gamma_{io}(P)) \Rightarrow p \in P$.
   (a) If $p = \langle c_0, \mathit{fin}(c_n)\rangle$, then it exists $r \in \gamma_{io}(P)$ such that $\mathit{last}(r{\Downarrow}_{c_0}) = c_n$. Obviously, $p \in P$, otherwise r would not belong to $\gamma_{io}(P)$.
   (b) If $p = \langle c_0, \mathit{inf}(c)\rangle$, then it exists $r \in \gamma_{io}(P)$ such that $r{\Downarrow}_{c_0} = c_0 \dots c_i \dots$ and $\otimes_{i \geq 0} = c$. It is easy to notice that $p \in P$, otherwise by using γ_{io} we would not obtain r.

Proof of Theorem 4.3.
Consider $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$, then $\alpha_{io}(\mathcal{P}[\![D \,.\, A]\!]) = \mathcal{B}^{io}[\![D \,.\, A]\!]$. We show the two inclusions independently.

$\boxed{\subseteq}$ Let $r \in \mathcal{P}[\![D \,.\, A]\!]$ and $c_0 \in \mathbf{C}$ such that $r{\Downarrow}_{c_0}$ is defined. In order to show that $\alpha_{io}(\{r\}) \subseteq \mathcal{B}^{io}[\![D \,.\, A]\!]$, we distinguish two cases.

1. In case $r{\Downarrow}_{c_0}$ is finite, let us define $c_n := \mathit{last}(r{\Downarrow}_{c_0})$. By (4.1) $\langle c_0, \mathit{fin}(c_n)\rangle \in \alpha_{io}(\mathcal{P}[\![D \,.\, A]\!])$. By Definitions 3.21, 3.16 and 3.26 it is easy to notice that r can be of three different forms:
   (a) r ends with $\boxtimes$,
   (b) r contains a *stutt* or
   (c) r contains a conditional store $\eta \rightarrowtail d$ such that there is no *stutt* before and $c_0 \otimes d = \mathit{ff}$.

   Now, we show that r takes one of those form when A, with initial store c_0, behaves in the following way: $\langle A, c_0\rangle \to^* \langle A_n, c_n\rangle \not\to$. Looking to the agent semantics $\mathcal{A}$ (Definition 3.16) we observe that:

(a) we obtain a sequence that ends with $\boxtimes$ if a subagent of A is equal to skip or tell, this means that, starting from an initial store c_0 such that $last(r\Downarrow_{c_0})$ is well defined, the operational semantics cannot perform any step from the configuration reached $\langle \mathsf{skip}, c_n\rangle \not\to$;

(b) when A contains an agent $\sum_{i=1}^{n} \mathsf{ask}(g_i) \to A_i$ and $\forall i \in [1,n].\, g_i \neq ff$, then a $stutt(\cup_{i=1}^{n})$ is introduced. Since we assume that $r\Downarrow_{c_0}$ is well defined, it holds that the guards are not entailed by c_0 (merged with the store produced by the sequence up to that position), thus the operational semantics cannot perform any step from the configuration reached $\langle \sum_{i=1}^{n} \mathsf{ask}(g_i) \to A_i, c_n\rangle \not\to$;

(c) when r contains a conditional state $\eta \rightarrowtail d$ (that occurs before any $stutt$) such that $c_0 \otimes d = ff$, we can deduce that, starting from $\langle A, c_0\rangle$, we reach in a finite number of operational steps the state $\langle A_n, ff\rangle \not\to$, from which no further derivation is possible since an inconsistent store has been produced.

Thus, by Definition 4.1 $\langle c_0, fin(c_n)\rangle \in \mathcal{B}^{io}[\![D\,.\,A]\!]$.

2. In case $r\Downarrow_{c_0} = c_0 \dots c_i \dots$ is infinite, let us define $c := \otimes_{i\geq 0} c_i$. By (4.1) $\langle c_0, inf(c)\rangle \in \alpha_{io}(\mathcal{P}[\![D\,.\,A]\!])$. By Theorem 3.27, it is easy to notice that $r\Downarrow_{c_0} \in \mathcal{B}^{ss}[\![D\,.\,A]\!]_{c_0}$, thus A with initial store c_0 behaves in the following way: $\langle A, c_0\rangle \to \dots \to \langle A_i, c_i\rangle \to$. By Definition 4.1 it follows that $\langle c_0, inf(c)\rangle \in \mathcal{B}^{io}[\![D\,.\,A]\!]$.

$\boxed{\supseteq}$ Let $p \in \mathbf{IO}$, we show that $p \in \mathcal{B}^{io}[\![D\,.\,A]\!] \Rightarrow p \in \alpha_{io}(\mathcal{P}[\![D\,.\,A]\!])$. Let us distinguish two cases.

1. In case $p = \langle c_0, fin(c_n)\rangle$, by Definition 4.1 it follows that $\langle A, c_0\rangle \to \dots \to \langle A_n, c_0\rangle \not\to$, and by Definition 3.1, $c_0 \dots c_n \in \mathcal{B}^{ss}[\![D\,.\,A]\!]_{c_0}$. By Theorem 3.27, it exists $r \in \mathcal{P}[\![D\,.\,A]\!]$ such that $c_0\Downarrow_r = c_0 \dots c_n$, and by (4.1) it follows that $\langle c_0, fin(c_n)\rangle \in \alpha_{io}(\mathcal{P}[\![D\,.\,A]\!])$.
2. In case $p = \langle c_0, fin(c)\rangle$, by Definition 4.1 it follows that $\langle A, c_0\rangle \to \dots \to \langle A_i, c_i\rangle \to$, and by Definition 3.1, $c_0 \dots c_i \dots \in \mathcal{B}^{ss}[\![D\,.\,A]\!]_{c_0}$. By Theorem 3.27, it exists $r \in \mathcal{P}[\![D\,.\,A]\!]$ such that $c_0\Downarrow_r = c_0 \dots c_i \dots$, and by (4.1) it follows that $\langle c_0, inf(c)\rangle \in \alpha_{io}(\mathcal{P}[\![D\,.\,A]\!])$.

Proof of Theorem 4.5.

We prove the two directions separately.

$\boxed{\Rightarrow}$ We show the equivalent implication: $\pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_1]\!])) \neq \pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_2]\!])) \Rightarrow \mathcal{O}^{io}[\![P_1]\!] \neq \mathcal{O}^{io}[\![P_2]\!]$. Let us assume, without loss of generality, that $\pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_1]\!])) \subset \pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_2]\!]))$, this means that there exist $r_2 \in \mathcal{P}[\![P_2]\!]$ and $c_0 \in \mathbf{C}$ such that $r_2\Downarrow_{c_0} = c_0 \dots c_n$, but it does not exist $r_1 \in \mathcal{P}[\![P_1]\!]$ such that $r_1\Downarrow_{c_0} = c_0 \dots c_n$. Furthermore, $c_n \neq ff$, since, by hypothesis, r_2 is not a failed conditional trace. By Theorem 3.27, $c_0 \cdots c_n \in \mathcal{B}^{ss}[\![P_2]\!]$ and, by Definition 4.1, $\langle c_0, c_n\rangle \in \mathcal{B}_F^{io}[\![P_2]\!]$. Since $\mathcal{B}_F^{io}$ and $\mathcal{O}^{io}[\![]\!]$ differ only on sequence terminating in ff and $c_n \neq ff$, it follows directly that $\langle c_0, c_n\rangle \in \mathcal{O}^{io}[\![P_2]\!]$. On the other hand, we have that $c_0 \cdots c_n \notin \mathcal{B}^{ss}[\![P_1]\!]$, thus $\langle c_0, c_n\rangle \notin \mathcal{B}_F^{io}[\![P_1]\!]$. Since it is easy to notice that, given a $tccp$ program P, $\mathcal{O}^{io}[\![P]\!] \subseteq \mathcal{B}_F^{io}[\![P]\!]$, we have that $\langle c_0, c_n\rangle \notin \mathcal{O}^{io}[\![P_1]\!]$. Thus, $\langle c_0, c_n\rangle \in \mathcal{O}^{io}[\![P_2]\!] \smallsetminus \mathcal{O}^{io}[\![P_1]\!]$ and we can conclude that $\mathcal{O}^{io}[\![P_1]\!] \neq \mathcal{O}^{io}[\![P_2]\!]$.

$\Leftarrow$ We show the equivalent implication: $\mathcal{O}^{io}[\![P_1]\!] \neq \mathcal{O}^{io}[\![P_2]\!] \Rightarrow \pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_1]\!])) \neq \pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_2]\!]))$. Without loss of generality, assume that $\mathcal{O}^{io}[\![P_1]\!] \subset \mathcal{O}^{io}[\![P_2]\!]$, thus, there exists $\langle c_0, c_n \rangle \in \mathcal{O}^{io}[\![P_2]\!]$ such that $\langle c_0, c_n \rangle \notin \mathcal{O}^{io}[\![P_1]\!]$. Since no trace in $\mathcal{P}[\![P_1]\!] \sqcup \mathcal{P}[\![P_2]\!]$ is failed, we can assume that $c_n \neq \mathit{ff}$. This means that, by using the transition relation defined in [10], we have a derivation of the form $\langle A_2, c_0 \rangle \to \dots \langle A_2', c_n \rangle \not\to$, with $A_2, A_2' \in \mathbb{A}_{\mathbf{C}}^{\Pi}$, $D_2 \in \mathbb{D}_{\mathbf{C}}^{\Pi}$ and $P_2 = D_2 \,.\, A_2$; Furthermore, a derivation, with initial constraint c_0, ending in c_n does not exists for P_1. On the other hand, it can be noticed that by using the transition relation of Figure 1, for P_1 there is no derivation starting with c_0 and ending in c_n. Thus we have that $\langle c_0, c_n \rangle \in \mathcal{B}_F^{io}[\![P_2]\!]$ and $\langle c_0, c_n \rangle \notin \mathcal{B}_F^{io}[\![P_1]\!]$. From Theorem 4.3 it follows that $\langle c_0, c_n \rangle \in \pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_2]\!])) \smallsetminus \pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_1]\!]))$ and we can conclude that $\pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_1]\!])) \neq \pi_F^{\mathbb{IO}}(\alpha_{io}(\mathcal{P}[\![P_2]\!]))$.

References

[1] M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and S. Lucas. Abstract Diagnosis of Functional Programs. In M. Leuschel, editor, *Logic Based Program Synthesis and Transformation – 12th International Workshop, LOPSTR 2002, Revised Selected Papers*, volume 2664 of *Lecture Notes in Computer Science*, pages 1–16, Berlin, 2003. Springer-Verlag.

[2] A. Aristizábal, F. Bonchi, C. Palamidessi, L. F. Pino, and F. D. Valencia. Deriving Labels and Bisimilarity for Concurrent Constraint Programming. In *14th International Conference on Foundations of Software Science and Computational Structures (FOSSACS 2011)*, volume 6604 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2011. 2.1

[3] G. Bacci and M. Comini. Abstract Diagnosis of First Order Functional Logic Programs. In M. Alpuente, editor, *Logic-based Program Synthesis and Transformation, 20th International Symposium*, volume 6564 of *Lecture Notes in Computer Science*, pages 215–233, Berlin, 2011. Springer-Verlag.

[4] M. Comini. *An Abstract Interpretation Framework for Semantics and Diagnosis of Logic Programs.* PhD thesis, Dipartimento di Informatica, Universitá di Pisa, Pisa, Italy, 1998.

[5] M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Abstract Diagnosis. *Journal of Logic Programming*, 39(1-3):43–93, 1999.

[6] M. Comini, L. Titolo, and A. Villanueva. Abstract Diagnosis for Timed Concurrent Constraint programs. *Theory and Practice of Logic Programming*, 11(4-5):487–502, 2011.

[7] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, San Antonio, Texas, January 29–31*, pages 269–282, New York, NY, USA, 1979. ACM Press. 4

[8] F. S. de Boer, A. di Pierro, and C. Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theoretical Computer Science*, 151:37–78, 1995. 1, 2.1, 3, 3.25

[9] F. S. de Boer, M. Gabbrielli, E. Marchiori, and C. Palamidessi. Proving Concurrent Constraint Programs Correct. *ACM Trans. Program. Lang. Syst.*, 19(5):685–725, 1997. 2.1, 3, 3.2.2

[10] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Timed Concurrent Constraint Language. *Information and Computation*, 161(1):45–83, 2000. 1, 1, 2.1, 2.2, (document), 3.25, 4, 4.1, 4.2, 4.2, A.2

[11] F. S. de Boer, J. N. Kok, C. Palamidessi, and J. J. M. M. Rutten. The Failure of Failures in a Paradigm for Asynchronous Communication. In J. C. M. Baeten and J. F. Groote, editors, *Proceedings of the 2nd International Conference on Concurrency Theory (CONCUR'91)*, volume 527, pages 111–126. Springer-Verlag, 1991.

[12] F. S. de Boer, J. N. Kok, C. Palamidessi, and J. J. M. M. Rutten. On Blocks: locality and asynchronous communication (Extended Abstract). In *Proceedings of Sematics: Foundations and Applications, REX Workshop*, volume 666, pages 73–90. Springer-Verlag, 1992.

[13] F. S. de Boer and C. Palamidessi. A Fully Abstract Model for Concurrent Constraint Programming. In S. Abramsky and T. S. E. Maibaum, editors, *Proceedings of TAPSOFT'91*, volume 493 of *Lecture Notes in Computer Science*, pages 296–319, Berlin, 1991. Springer-Verlag. 1, 4

[14] M. Falaschi, M. Gabbrielli, K. Marriott, and C. Palamidessi. Confluence in concurrent constraint programming. *Theoretical Computer Science*, 183:281–315, 1997. 1, 3

[15] M. Falaschi, C. Olarte, and C. Palamidessi. A framework for abstract interpretation of timed concurrent constraint programs. In A. Porto and F. Javier López-Fraguas, editors, *PPDP*, pages 207–218. ACM, 2009. 3

[16] M. Falaschi, C. Olarte, C. Palamidessi, and F. D. Valencia. Declarative Diagnosis of Temporal Concurrent Constraint Programs. In V. Dahl and I. Niemelä, editors, *Logic Programming, 23rd International Conference, ICLP 2007, Proceedings*, volume 4670 of *Lecture Notes in Computer Science*, pages 271–285. Springer-Verlag, 2007. 3

[17] M. Falaschi and A. Villanueva. Automatic verification of timed concurrent constraint programs. *Theory and Practice of Logic Programming*, 6(3):265–300, 2006. 3.25

[18] L. Henkin, J. D. Monk, and A. Tarski. *Cylindric Algebras. Part I and II.* Elsevier Science Publishers, North Holland, 1971.

[19] M. Nielsen, C. Palamidessi, and F. D. Valencia. On the expressive power of temporal concurrent constraint programming languages. In *PPDP*, pages 156–167. ACM, 2002.

[20] M. Nielsen, C. Palamidessi, and F. D. Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. *Nordic Journal of Computing*, 9(1):145–188, 2002. 3, 3.2.2

[21] C. Olarte and F. D. Valencia. Universal concurrent constraint programing: symbolic semantics and applications to security. In R. Wainwright and H. Haddad, editors, *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC08)*, pages 145–150. ACM, 2008. 3

[22] D. Park. Fixpoint Induction and Proofs of Program Properties. *Machine Intelligence*, 5:59–78, 1969.

[23] V. A. Saraswat. *Concurrent Constraint Programming.* The MIT Press, Cambridge, Mass., 1993. 1, 2.1

[24] V. A. 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*, pages 71–80. IEEE Computer Press, 1994. (document), 3

[25] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Timed Default Concurrent Constraint Programming. *Journal on Symbolic Computation*, 11:1–42, 1999. (document), 3

[26] V. A. Saraswat and M. Rinard. Concurrent constraint programming. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 232–245, New York, NY, USA, 1990. ACM. 2, 2.1

[27] V. A. Saraswat, M. Rinard, and P. Panangaden. The Semantic Foundations of Concurrent Constraint Programming. In *Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 333–352, New York, NY, USA, 1991. Acm. 2.1, 2.2, 3