Transformations of CCP Programs

SANDRO ETALLE
University of Twente and CWI
MAURIZIO GABBRIELLI
Università di Bologna
and
MARIA CHIARA MEO
Università di Chieti

We introduce a transformation system for concurrent constraint programming (CCP). We define suitable applicability conditions for the transformations that guarantee the input/output CCP semantics is also preserved when distinguishing deadlocked computations from successful ones and when considering intermediate results of (possibly) nonterminating computations.

The system allows us to optimize CCP programs while preserving their intended meaning: In addition to the usual benefits for sequential declarative languages, the transformation of concurrent programs can also lead to the elimination of communication channels and of synchronization points, to the transformation of nondeterministic computations into deterministic ones, and to the crucial saving of computational space. Furthermore, since the transformation system preserves the deadlock behavior of programs, it can be used for proving deadlock-freeness of a given program with respect to a class of queries. To this aim, it is sometimes sufficient to apply our transformations and to specialize the resulting program with respect to the given queries in such a way that the obtained program is trivially deadlock-free.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*concurrent, distributed, and parallel languages*; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program transformation*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*algebraic approaches to semantics*; D.1.3 [**Programming Techniques**]: Concurrent Programming

General Terms: Languages, Theory

Additional Key Words and Phrases: Concurrent constraint programming, deadlock-freeness, optimization

Author's address: S. Etalle, Distributed and Embedded Systems group, Faculty of Computer Scienze, University of Twente, P.O. Box 217 7500AE Enschede, The Netherlands; M. Gabbrielli, Dipartimento di Scienze dell' Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127, Italy; gabbri@cs.unibo.it; Maria Chiara Meo, Dipartimento di Scienze, Universitá di Chieti, Viale Pimdaro 42, 65127 Pescara, Italy; meo@univaq.it.

1. INTRODUCTION

In the case of logic-based languages, optimization techniques fall into two main categories: On the one hand, there exist methods for compile-time and low-level optimizations such as those for constraint logic programs [Jørgensen et al. 1991], which are usually based on program analysis methodologies (e.g., abstract interpretation). On the other hand, we find source-to-source transformation techniques such as *partial evaluation* [Mogensen and Sestoft 1997] and more general techniques based on the *unfold* and *fold* or on the *replacement* operation.

Unfold/fold transformation techniques were first introduced for functional programs in Burstall and Darlington [1977], and then adapted to logic programming (LP) both for program synthesis [Clark and Sickel 1977; Hogger 1981] and for program specialization and optimization [Komorowski 1982]. Tamaki and Sato [1984] proposed a general framework for the unfold/fold transformation of logic programs, which for years has remained the main historical reference in the field, and has later been extended to constraint logic programming (CLP) in Maher [1993]; Etalle and Gabbrielli [1996]; and Bensaou and Guessarian [1998] (for an overview of the subject, see the survey by Pettorossi and Proietti [1994]). As shown by a number of applications, these techniques provide a powerful methodology for the development and optimization of large programs, and can be regarded as the *basic* transformation techniques, which might be adapted further for partial evaluation.

Despite an extensive literature on declarative sequential languages, unfold/fold transformation sequences have hardly been applied to concurrent languages. Notable exceptions are the papers by Ueda and Fukurawa [1988]; Sahlin [1995]; and de Francesco and Santone [1996] (their relations with this article are discussed in Section 7). When considering partial evaluation, we also find only very few recent attempts [Hosoya et al. 1996; Marinescu and Goldberg 1997; Gengler and Martel 1997] that apply it to the field of concurrent languages.

This situation is partially due to the fact that the nondeterminism and synchronization mechanisms in concurrent languages substantially complicate their semantics, thus also complicating the definition of *correct* transformation systems. Nevertheless, these transformation techniques can be very useful for concurrent languages also, since they allow further optimizations related to the simplification of synchronization and communication mechanisms.

In this article we introduce a transformation system for concurrent constraint programming (CCP) [Saraswat 1989; Saraswat and Rinard 1990; Saraswat et al. 1991]. This paradigm derives from replacing the *store-as-valuation* concept of von Neumann computing by the *store-as-constraint* model—whose computational model is based on a global *store*, which consists of the conjunction of all constraints established until that moment, and expresses some partial information on the values of the variables in the computation. Concurrent processes synchronize and communicate asynchronously via the store by using elementary actions (ask and tell), which can be expressed in a logical form (essentially implication and conjunction [Boer et al. 1997]). On the one hand, CCP enjoys a clean logical semantics, avoiding many of the complications in the concurrent imperative setting. As argued in the position paper by Etalle and Gabbrieli [1998], this aspect is of great help in the development of effective transformation tools. On the other hand, at variance from other theoretical models for concurrency (e.g., the π-calculus), there exist "real" implementations of concurrent constraint languages (notably, the Oz language [Smolka 1995] and the related ongoing Mozart project `http://www.mozart-oz.org/`). Thus, in contrast to other models for concurrency, in this framework, transformation techniques can be readily applied to practical problems.

The transformation system we introduce was originally inspired by the Tamaki and Sato [1984] system. Compared to its predecessors, it offers three improvements: First, we managed to eliminate the limitation that, in a folding operation, the *folding clause* has to be nonrecursive—a limitation present in many other unfold/fold transformation systems—this improvement may lead to the use of new, more sophisticated, transformation strategies. Second, the applicability conditions we propose for the folding operation are now independent of *transformation history*, making the operation much easier to understand and to implement. In fact, following Francesco and Santone [1996], our applicability conditions are based on the notion of "guardedness," and can be checked locally on the program to be folded. Finally, we introduced several new transformation operations. It is also worth mentioning that the declarative nature of CCP allows us to define reasonably simple applicability conditions that ensure the correctness of our system.

We illustrate with a practical example how our transformation system for CCP can be even more useful than its predecessors for sequential logic languages. Indeed, in addition to the usual benefits, in this context the transformations can also lead to the elimination of communication channels and synchronization points, to the transformation of nondeterministic computations into deterministic ones, and to the crucial saving of computational *space*. These improvements were already possible in the context of GHC programs by using the system defined in Ueda and Furukawa [1988].

Our results show that the original and the transformed programs have the same input/output semantics in a rather strong sense—which distinguishes successful, deadlocked, and failed derivations. As a corollary, we see that the original program is deadlock-free iff the transformed one is, and this allows us to employ the transformation system as an effective tool for proving deadlock-freeness: if, after the transformation, we can prove or see that the process we are considering never deadlocks (in some cases the transformation simplifies the program's behavior, so that this can be checked immediately), then we are also sure that the original process does not deadlock either. We also consider nonterminating computation by proving three further correctness results. The first one shows that the intermediate results of (possibly nonterminating) computations are preserved up to logical implication, while the second one ensures full preservation of (traces of) intermediate results, provided we slightly restrict the applicability conditions for our transformations. The third result shows that this restricted transformation system preserves a certain kind of infinite computation (active ones). We discuss the extension of this result to the general case, claiming that our system does not introduce any new infinite computation.

This article is organized as follows: in the next section we present the notation and the necessary preliminary definitions, most of them regarding the CCP paradigm. In Section 3, we define the transformation system, which consists of various different operations (for this reason the section is divided in a number of subsections). We also use a working example to illustrate the application of our methodology. Section 4 states the first main result, concerning the correctness of the transformation system; while Section 5 contains the results for nonterminating computation. Further examples are contained in Section 6. Section 7 compares this article to related work in the literature; and Section 8 concludes. For the sake of readability, we include in this article only proof sketches of several results; the (rather extensive) technical details being deferred to the Appendix.

A preliminary version of this article appeared in Etalle et al. [1998].

2. PRELIMINARIES

The basic idea underlying the CCP paradigm is that computation progresses via monotonic accumulation of information in a global store. The information is produced (in the form of constraints) by the concurrent and asynchronous activity of several agents that can *add* a constraint c to the store by performing the basic action $\mathsf{tell(c)}$. Agents can also *check* whether a constraint c is entailed by the store by using an $\mathsf{ask(c)}$ action. This allows the synchronization of different agents.

Concurrent constraint languages are defined parametrically with respect to the notion of a *constraint system*, which is usually formalized in an abstract way following the guidelines of Scott's treatment of information systems (see Saraswat and Rinard [1090]). Here, we consider a more concrete notion of constraint, based on first-order logic and coinciding with the one used for constraint logic programming (e.g., see Jaffar and Maher [1994]). This allows us to define the transformation operations in a more comprehensible way, while retaining sufficient expressive power. We could equally well define the transformations in terms of the abstract notion of a constraint system in Saraswat and Rinard [1990].[1]

Assume a given signature Σ defining a set of function and predicate symbols and associating an arity with each symbol. A *constraint c* is a first-order Σ-formula built by using symbols of Σ, variables from a given (countable) set V, and the logical connectives and quantifiers ($\wedge, \vee, \neg, \exists$) in the usual way. The interpretation for the symbols in Σ is provided by a Σ-structure $\mathcal{D}$ consisting of a set D and an assignment of functions and relations on D to the symbols in Σ, which respect the arities. So $\mathcal{D}$ defines the computational domain on which constraints are interpreted. In order to model parameter passing, Σ is usually assumed to contain the binary predicate symbol $=$, which is interpreted as the identity in $\mathcal{D}$. We follow this assumption, which allows us to avoid the use

[1]To this aim, we should essentially replace equations of the form $X = Y$ for diagonal elements d_{XY}.

of most general unifiers (indeed, for many computation domains $\mathcal{D}$ the most general unifier of two terms does not exist).

The formula $\mathcal{D} \models \mathsf{c}$ states that c is valid in the interpretation provided by $\mathcal{D}$, i.e., that it is true for every valuation of the free variables of c. The empty conjunction of primitive constraints will be identified with true. We also denote by $\mathit{Var}(\mathsf{e})$ the set of free variables occurring in the expression e.

In the sequel, constraints are considered up to equivalence in the domain $\mathcal{D}$, i.e., we write $\mathsf{c}_1 = \mathsf{c}_2$ in case $\mathcal{D} \models c_1 \leftrightarrow c_2$. Terms are denoted by $\mathsf{t}, \mathsf{s}, \ldots$, variables with $\mathsf{X}, \mathsf{Y}, \mathsf{Z}, \ldots$, further, as a notational convention, $\tilde{\mathsf{t}}$ and $\tilde{\mathsf{X}}$ denote a tuple of terms and a tuple of distinct variables, respectively. $\exists_{-\tilde{\mathsf{X}}}\, \mathsf{c}$ stands for the existential closure of c, except for the variables in $\tilde{\mathsf{X}}$ which remain unquantified. We also assume that the reader is acquainted with the notion of substitution and the most general unifier (see Lloyd [1987]). We denote by $\mathsf{e}\sigma$ the result application of a substitution σ to an expression e. Given a substitution σ, the *domain* of σ, $\mathit{Dom}(\sigma)$, is the finite set of variables $\{\mathsf{X} \mid \mathsf{X}\sigma \neq \mathsf{X}\}$, the *range* of σ is defined as $\mathit{Ran}(\sigma) = \bigcup_{\mathsf{X} \in \mathit{Dom}(\sigma)} \mathit{Var}(\mathsf{X}\sigma)$.

The notation and the semantics of programs and agents are virtually the same as these of Saraswat and Rinard [1990]. In particular, the $\|$ operator allows us to express parallel composition of two agents, and it is usually described in terms of interleaving, while nondeterminism arises by introducing a (global) choice operator $\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c}_i) \rightarrow \mathsf{A}_i$: the agent $\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c}_i) \rightarrow \mathsf{A}_i$ nondeterministically selects one $\mathsf{ask}(\mathsf{c}_i)$, which is enabled in the current store, and then behaves like A_i. Thus, the syntax of CCP *declarations* and *agents* is given by the following grammar:

$$\begin{array}{lll} \textit{Declarations} & \mathsf{D} ::= & \epsilon \mid \mathsf{p}(\tilde{\mathsf{t}}) \leftarrow \mathsf{A} \mid \mathsf{D}, \mathsf{D} \\ \textit{Agents} & \mathsf{A} ::= & \mathsf{stop} \mid \mathsf{tell}(\mathsf{c}) \mid \sum_{i=1}^{n} \mathsf{ask}(\mathsf{c}_i) \rightarrow \mathsf{A}_i \mid \mathsf{A} \parallel \mathsf{A} \mid \mathsf{p}(\tilde{\mathsf{t}}) \\ \textit{Processes} & \mathsf{Proc} ::= & \mathsf{D.A} \end{array}$$

where c and c_i's are constraints. Note that here we allow terms as both formal and actual parameters.

This is not usually the case, since the procedure call $\mathsf{p}(\tilde{\mathsf{t}})$ can be equivalently written as $\mathsf{p}(\tilde{\mathsf{X}}) \parallel \mathsf{tell}(\tilde{\mathsf{X}} = \tilde{\mathsf{t}})$, while the declaration $\mathsf{p}(\tilde{\mathsf{t}}) \leftarrow \mathsf{A}$ is equivalent to $\mathsf{p}(\tilde{\mathsf{X}}) \leftarrow \mathsf{A} \parallel \mathsf{tell}(\tilde{\mathsf{X}} = \tilde{\mathsf{t}})$. We make this assumption only because it simplifies the writing of programs in the examples.

Due to the presence of an explicit choice operator, we assume as usual (without loss of generality) that each predicate symbol is defined by exactly one declaration. A *program* is a set of declarations. In the following examples we assume that the operator $\sum$ binds tighter than $\|$ (so, $\mathsf{ask}(\mathsf{a}) \rightarrow \mathsf{A} \parallel \mathsf{ask}(\mathsf{b}) \rightarrow \mathsf{B} + \mathsf{ask}(\mathsf{d}) \rightarrow \mathsf{C}$ means $(\mathsf{ask}(\mathsf{c}) \rightarrow \mathsf{A}) \parallel (\mathsf{ask}(\mathsf{b}) \rightarrow \mathsf{B} + \mathsf{ask}(\mathsf{d}) \rightarrow \mathsf{C})$). In case some ambiguity arise, we use brackets to indicate the scope of the operators.

An important aspect, for which we depart slightly from the usual formalization of CCP, is the notion of *locality*. In Saraswat and Rinard [1990] locality is obtained by using the operator $\exists$, and the behavior of the agent $\exists_{\mathsf{X}} \mathsf{A}$ is defined like A, with the variable X considered as *local* to it. Here, we do not use such an explicit operator: Analogously to the standard CLP setting, locality is introduced implicitly by assuming that if a process is defined by $\mathsf{p}(\tilde{\mathsf{t}}) \leftarrow \mathsf{A}$ and a variable Y occurs in A but not in $\tilde{\mathsf{t}}$, then Y has to be considered local to A.

Table I. The (Standard) Transition System.

R1	$\langle D.tell(c), d\rangle \to \langle D.stop, c \wedge d\rangle$	
R2	$\langle D. \sum_{i=1}^{n} ask(c_i) \to A_i, d\rangle \to \langle D.A_j, d\rangle$	if $j \in [1, n]$ *and* $\mathcal{D} \models d \to c_j$
R3	$\dfrac{\langle D.A, c\rangle \to \langle D.A', c'\rangle}{\begin{array}{c}\langle D.(A \parallel B), c\rangle \to \langle D.(A' \parallel B), c'\rangle \\ \langle D.(B \parallel A), c\rangle \to \langle D.(B \parallel A'), c'\rangle\end{array}}$	
R4	$\langle D.p(\tilde{t}), c\rangle \to \langle D.A \parallel tell(\tilde{t} = \tilde{s}), c\rangle$	if $p(\tilde{s}) \leftarrow A \in defn_D(p)$

The operational model of CCP is described by a transition system $T = (Conf, \to)$ where configurations (in) $Conf$ are pairs consisting of a process and a constraint (representing the common *store*), while the transition relation $\to \ \subseteq Conf \times Conf$ is described by the (least relation satisfying the) rules **R1**–**R4** of Table 1, which should be self-explanatory. Here and in the following, we assume that we are given a set D of declarations and by $defn_D(p)$ we denote the set of variants[2] of the (unique) declaration in D for the predicate symbol p. Due to the presence of terms as arguments to predicates symbols, which is different from the standard setting in rule **R4**, parameter-passing is performed by a tell action. We also assume the presence of a renaming mechanism that takes care of using fresh variables each time a declaration is considered.[3]

By $\to^*$ we denote the reflexive-transitive closure of the relation $\to$ defined by the transition system; and by $Stop$ we denote any agent that contains only $stop$ and $\parallel$ constructs. A finite derivation (or computation) containing only satisfiable constraints is called *successful* if it is of the form $\langle D.A, c\rangle \to^* \langle D.Stop, d\rangle \not\to$, while it is called *deadlocked* if it is of the form $\langle D.A, c\rangle \to^* \langle D.B, d\rangle \not\to$, with B different from $Stop$ (i.e., B contains at least one suspended agent). A derivation that produces eventually false is called failed. Note that here we consider the so-called "eventual tell" CCP; i.e., when adding constraints to the store (via tell operations) there is no consistency check. Our results could be adapted to the CCP language with consistency check ("atomic tell" CCP) by minor modifications of the transformation operations.

3. THE TRANSFORMATION

In order to illustrate the application of our method, we adopt a working example. We consider an auction problem in which two bidders participate: bidder_a and bidder_b; each bidder takes as input the list of the bids of the other one and produces as output the list of its own bids. When one of the two bidders wants to quit the auction, produces in its own output stream the token quit. This protocol is implemented by the following program, AUCTION. Here and in the following

[2]A variant of a declaration d is obtained by replacing the tuple $\tilde{X}$ of all the variables appearing in d for another tuple $\tilde{Y}$.

[3]For the sake of simplicity, we do not describe this renaming mechanism in the transition system. In Saraswat and Rinard [1990] and Saraswat et al. [1991] the interested reader can find various formal approaches to this problem.

examples, we do not make any assumptions on the specific constraint domain being used, apart from the fact that it should allow us to use lists of elements. This is the case for most existing general-purpose constraint languages, which usually also incorporate some arithmetic domain (see Jaffar and Maher [1994]).

```
auction(LeftBids,RightBids) ← bidder_a([0|RightBids],LeftBids) ∥
        bidder_b(LeftBids,RightBids)

bidder_a(HisList, MyList) ←
        ask(∃_{HisBid,HisList'} HisList = [HisBid|HisList']  ∧ HisBid = quit) → stop
    + ask(∃_{HisBid,HisList'} HisList = [HisBid|HisList']  ∧ HisBid ≠ quit) →
            (tell(HisList = [HisBid|HisList']) ∥
            make_new_bid_a(HisBid,MyBid) ∥
                ask(MyBid = quit) → (tell(MyList = [MyBid|MyList']) ∥ broadcast("a quits"))
            + ask(MyBid ≠ quit) → (tell(MyList = [MyBid|MyList']) ∥
                    tell(MyBid ≠ quit) ∥
                    bidder_a(HisList',MyList')))
```

plus an analogous definition for bidder_b.

Here, the agent make_new_bid_a(HisBid,MyBid) is in charge of producing a new offer, in the presence of the competitor's offer, HisBid; the agent will produce MyBid = quit if it evaluates-that HisBid is too high to be topped, and decides to leave the auction. This agent could be specified further by using arithmetic constraints. In order to avoid deadlock, auction initializes the auction by inserting a fictitious zero bid in the input of bidder a. Notice that in the above program the agent tell(HisList = [HisBid|HisList']) is needed to bind the local variables (HisBid, HisList') to the global one (HisList). In fact, as a result of operational semantics, such a binding is not performed by the ask agent. On the contrary, the agent tell(MyBid ≠ quit) is redundant: we have introduced it in order to slightly simplify the following transformations (the transformations also remain possible without such a tell). The introduction of redundant tells is a transformation operation is formally defined in Section 3.4.

3.1 Introduction of a New Definition

The introduction of a new definition is virtually always the first step in a transformation sequence. Since the new definition is going to be the main target of the transformation operation, this step will actually determine the very direction of the subsequent transformation, and thus the degree of its effectiveness.

Determining which definitions should be introduced is a very difficult task, which falls into the area of *strategies*. To give a simple example, if we wanted to apply *partial evaluation* to our program with respect to a given agent A (i.e., if we wanted to specialize our program so that it would execute the partially instantiated agent A in a more efficient way), then a good starting point would most likely be the introduction of the definition $p(\tilde{X}) \leftarrow A$, where $\tilde{X}$ is an appropriate tuple of variables and p is a new predicate symbol. A different strategy would probably determine the introduction of a different new definition. For a survey of the other possibilities, refer to Pettorossi and Proietti [1994].

In this article we are not concerned with the strategies, but only with the basic transformation operations and their correctness: We aim to define a transformation system that is general enough to be applied in combination with different strategies.

In order to simplify the terminology and the technicalities, we assume that these new declarations are added once and for all to the original program before starting the transformation itself. Note that this is clearly not restrictive. As a notational convention, we call D_0 the program obtained after the introduction of new definitions. In the case of program AUCTION, we assume that the following new declarations are added to the original program.

```
auction_left(LastBid)  ← tell(LastBid ≠ quit)  || bidder_a([LastBid|Bs],As)  || bidder_b(As,Bs).
auction_right(LastBid)  ← tell(LastBid ≠ quit)  || bidder_a(Bs,As)  || bidder_b([LastBid|As],Bs).
```

The agent auction_left(LastBid) engages an auction starting with bid LastBid (which cannot be quit) and expecting bidder "a" to be the next one in the bid. The agent auction_right(LastBid) is symmetric.

3.2 Unfolding

The first transformation we consider is *unfolding*, which consists essentially in the replacement of a procedure call by its definition. The syntax of CCP agents allows us to define it in a very simple way by using the notion of context. A *context*, denoted by C[ ], is simply an agent with a "hole," where the hole can contain any expression of type agent. So, for example, $[\ ] \parallel \mathsf{A}$ and $\mathsf{ask(c)} \rightarrow \mathsf{A} + \mathsf{ask(b)} \rightarrow [\ ]$ are contexts, while $\mathsf{ask(a)} \rightarrow \mathsf{A} + [\ ]$ is not. C[A] denotes the agent obtained by replacing the hole in C[ ] for the agent A in the obvious way.

Definition 3.1 (*Unfolding*). Consider a set of declarations D containing

$$\mathsf{d}\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]$$
$$\mathsf{u}\colon \mathsf{p}(\tilde{\mathsf{s}}) \leftarrow \mathsf{B}$$

Then *unfolding* $\mathsf{p}(\tilde{\mathsf{t}})$ in d consists in replacing d by

$$\mathsf{d}'\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]$$

in D. Here d is the *unfolded* definition, and u is the unfolding one; d and u are assumed to be renamed so that they do not share variables.

After an unfolding, we often need to simplify some of the newly introduced tell's in order to "clean up" the resulting declarations. This is accomplished via a tell elimination. Recall that a most general unifier σ of the terms t and s is called *relevant* if $(Dom(\sigma) \cup Ran(\sigma)) \subseteq Var(\mathsf{t}, \mathsf{s})$.

Definition 3.2 (*Tell Elimination and Tell Introduction*). The declaration

$$\mathsf{d}\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{tell}(\tilde{\mathsf{s}} = \tilde{\mathsf{t}}) \parallel \mathsf{B}]$$

can be transformed via a *tell elimination* into

$$\mathsf{d}'\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{B}\sigma]$$

where σ is a relevant most general unifier of $\tilde{s}$ and $\tilde{t}$, provided that the variables in the domain of σ do not occur either in C[ ] or in H. This operation is applicable either when the computational domain $\mathcal{D}$ admits a most general unifier or when $\tilde{s}$ and $\tilde{t}$ are sequence of distinct variables, in which case σ is simply a renaming. On the other hand, the declaration

$$\text{d: H} \leftarrow \text{C[B}\sigma\text{]}$$

can be transformed via a *tell introduction* into

$$\text{d}'\text{: H} \leftarrow \text{C[tell(}\tilde{X} = \tilde{X}\sigma\text{)} \parallel \text{B]}$$

provided that σ is a substitution such that $\tilde{X} = Dom(\sigma)$ and $Dom(\sigma) \cap (Var(\text{C[ ], H}) \cup Ran(\sigma)) = \emptyset$.

In particular, we can always exchange C[tell(true) ∥ A] with C[A], and vice-versa. The presence of $Ran(\sigma)$ in the above condition is needed to ensure that σ is idempotent: In fact, using substitutions σ in the form $X/f(X)$ is not correct in general. In practice, the constraints on the domain of σ can be weakened by appropriately renaming some local variables; this is also shown in the upcoming example. In fact, if all the occurrences of a local variable in C[ ] are in choice branches different from the one the "hole" lies in, then we can safely rename each one of these occurrences.

In our AUCTION example, we start working on the definition of auction_right, and unfold the agent bidder_b([LastBid|As], Bs) and then perform the subsequent tell eliminations (we eliminate the tells introduced by the unfolding). The result of these operations is the following program:

```
auction_right(LastBid) ← tell(LastBid ≠ quit) ∥
    bidder_a(Bs, As) ∥
       ask(∃HisBid,HisList′ [LastBid|As] = [HisBid|HisList'] ∧ HisBid = quit) → stop
    + ask(∃HisBid,HisList′ [LastBid|As] = [HisBid|HisList'] ∧ HisBid ≠ quit) →
          tell([LastBid|As] = [HisBid|HisList']) ∥
          make_new_bid_b(HisBid,MyBid) ∥
             ask(MyBid = quit) → tell(Bs = [MyBid|Bs']) ∥ broadcast("b quits")
          + ask(MyBid ≠ quit) → tell(Bs = [MyBid|Bs']) ∥
                tell(MyBid ≠ quit) ∥
                bidder_b(HisList',Bs')
```

3.3 Backward Instantiation

The new *backward instantiation* operation is somehow similar to the unfolding one. We begin immediately with its definition.

Definition 3.3 (*Backward Instantiation*). Let D be a set of definitions and

$$\text{d: H} \leftarrow \text{C[p(}\tilde{t}\text{)]}$$
$$\text{b: p(}\tilde{s}\text{)} \leftarrow \text{tell(c)} \parallel \text{B}$$

be two definitions of D. The *backward instantiation* of $p(\tilde{t})$ in d consists in replacing d by d′, which is either

$$d' \colon H \leftarrow C[p(\tilde{t}) \parallel \mathsf{tell}(c) \parallel \mathsf{tell}(\tilde{t} = \tilde{s})]$$

or

$$d' \colon H \leftarrow C[p(\tilde{t}) \parallel \mathsf{tell}(\tilde{t} = \tilde{s})]$$

(it is assumed here that d and b are renamed so that they have no variables in common).

More generally, by considering c to be true, the operation can also be applied when b is not of the form $p(\tilde{s}) \leftarrow \mathsf{tell}(c) \parallel B$.

Intuitively, this operation can be regarded as a "half-unfolding" for the following reason: Performing an unfolding is equivalent to applying a derivation step to the atomic agent under consideration, here we do not quite do it, yet we carry out (part of) the first two phases that the derivation step requires.

In Section 6 we show an application of this operation (Example 6.2).

3.4 Ask and Tell Simplification

A new important operation is one that allows us to modify the ask guards and the tells in a program. Let us call the *produced constraint* of C[ ] the conjunction of all the constraints appearing in ask and tell actions that can be evaluated before [ ] is reached (in the context of C[ ]). Now, if a is the produced constraint of C[ ] and $\mathcal{D} \models a \rightarrow c$, then clearly we can simplify an agent of the form $C[\mathsf{ask}(c) \rightarrow A + \mathsf{ask}(d) \rightarrow B]$ to $C[\mathsf{ask}(\mathsf{true}) \rightarrow A + \mathsf{ask}(d) \rightarrow B]$.[4] Moreover, under the previous hypothesis, we can clearly transform $C[\mathsf{tell}(c) \parallel A]$ to C[A] and, conversely, C[A] to $C[\mathsf{tell}(c) \parallel A]$ (as mentioned previously, the latter transformation, consisting the introduction of a redundant tell, might be needed to prepare a program for the folding operation).

In general, if a is the produced constraint of C[ ] and for some constraint c′ we have that $\mathcal{D} \models \exists_{-\tilde{Z}} (a \wedge c) \leftrightarrow (a \wedge c')$ (where $\tilde{Z} = Var(C, A)$), then we can replace c with c′ in $C[\mathsf{ask}(c) \rightarrow A]$ and in $C[\mathsf{tell}(c)]$. In particular, if we have that $a \wedge c$ is unsatisfiable, then c can be immediately replaced with false (the unsatisfiable constraint). In order to formalize this intuitive idea, we start with the following definition.

Definition 3.4. Given an agent A, the *produced constraint* of A is denoted by pca(A) and is defined by structural induction, as follows:

$$\begin{aligned} pca(\mathsf{tell}(c)) &= c \\ pca(A \parallel B) &= pca(A) \wedge pca(B) \\ pca(A) &= \mathsf{true} \quad \text{for any agent A which is neither of the form tell(c)} \\ & \qquad\qquad \text{nor a parallel composition.} \end{aligned}$$

[4]Note that in general the further simplification to $C[A + \mathsf{ask}(d) \rightarrow B]$ is not correct, although we can transform $C[\mathsf{ask}(\mathsf{true}) \rightarrow A]$ into C[A].

By extending the definition we use for agents to contexts, given a context C[ ], the *produced constraint* of C[ ] is denoted by pc(C[ ]), and is inductively defined as follows:

$$\begin{aligned} \mathsf{pc}([\ ]) &= \mathsf{true} \\ \mathsf{pc}(\mathsf{C}'[\ ] \parallel \mathsf{B}) &= \mathsf{pc}(\mathsf{C}'[\ ]) \wedge \mathsf{pca}(\mathsf{B}) \\ \mathsf{pc}\left(\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c_i}) \rightarrow \mathsf{A_i}\right) &= \mathsf{c_j} \wedge \mathsf{pc}(\mathsf{C}'[\ ]) \text{ where } \mathsf{j} \in [1, \mathsf{n}] \text{ and } \mathsf{A_j} = \mathsf{C}'[\ ] \end{aligned}$$

The following definition allows us to determine when two constraints are *equivalent* within a given context C[ ].

Definition 3.5. Let c, c′ be the constraints, C[ ] be a context, and $\tilde{Z}$ be a set of variables. We say that c is *equivalent to* c′ *within* C[ ] and *w.r.t. the variables in* $\tilde{Z}$ iff $\mathcal{D} \models \exists_{-\tilde{Z}} (\mathsf{pc}(\mathsf{C}[\ ]) \wedge \mathsf{c}) \leftrightarrow \exists_{-\tilde{Z}} (\mathsf{pc}(\mathsf{C}[\ ]) \wedge \mathsf{c}')$.

This definition is employed in the following operation, which allows us to simplify the constraints in the ask and tell guards.

Definition 3.6 (*Ask and Tell Simplification*). Let D be a set of declarations.

(1) Let d: $\mathsf{H} \leftarrow \mathsf{C}[\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c_i}) \rightarrow \mathsf{A_i}]$ be a declaration of D. Suppose that $\mathsf{c}'_1, \ldots, \mathsf{c}'_n$ are constraints such that for $j \in [1, n]$, c'_j is equivalent to $\mathsf{c_j}$ within C[ ] and w.r.t. the variables in *Var*(C, H, $\mathsf{A_j}$).
Then we can replace d with d′: $H \leftarrow C[\sum_{i=1}^{n} ask(c'_i) \rightarrow A_i]$ in D. We call this an *ask simplification* operation.
(2) Let d: H ← C[tell(c)] be a declaration of D. Suppose that the constraint c′ is equivalent to c within C[ ] and w.r.t. the variables in *Var*(C, H).
Then we can replace d with d′: H ← C[tell(c′)] in D. We call this a *tell simplification* operation.

In our AUCTION example, we can consider the produced constraint of tell(LastBid ≠ quit) and modify the subsequent ask constructs, as follows:

```
auction_right(LastBid) ← tell(LastBid ≠ quit) ||
    bidder_a(Bs, As) ||
       ask(∃HisBid,HisList′ [LastBid|As] = [HisBid|HisList’] ∧ LastBid ≠ quit ∧ HisBid = quit) →
          stop
       +
       ask(∃HisBid,HisList′ [LastBid|As] = [HisBid|HisList’]) →
          tell([LastBid|As] = [HisBid|HisList’]) || ...
```

Via the same operation, we can immediately simplify this to

```
auction_right(LastBid) ← tell(LastBid ≠ quit) || bidder_a(Bs, As) ||
       ask(false) → stop
    + ask(true) → tell([LastBid|As] = [HisBid|HisList’]) || ...
```

3.5 Branch Elimination and Conservative Ask Elimination

In the above program we have a guard ask(false), which will of course never be satisfied. Thus, the first important application of the guard simplification operation concerns the elimination of unreachable branches.

Definition 3.7 (*Branch Elimination*). Let D be a set of declarations and let

$$\mathsf{d}:\ \mathsf{H} \leftarrow \mathsf{C}\left[\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c}_i) \rightarrow \mathsf{A}_i\right]$$

be a declaration of D. Assume that $n > 1$ and that for some $j \in [1, n]$, we have that $\mathsf{c}_j = \mathsf{false}$, then we can replace d with

$$\mathsf{d}':\ \mathsf{H} \leftarrow \mathsf{C}\left[\left(\sum_{i=1}^{j-1} \mathsf{ask}(\mathsf{c}_i) \rightarrow \mathsf{A}_i\right) + \left(\sum_{i=j+1}^{n} \mathsf{ask}(\mathsf{c}_i) \rightarrow \mathsf{A}_i\right)\right].$$

The condition that $n > 1$ means that we cannot eliminate all the branches of a choice, and it is needed to ensure the correctness of the system (otherwise we could transform a deadlock into a success. For example, the agent tell(c) $\parallel$ ask(false) $\rightarrow$ stop when evaluated in the empty store produces the constraint c and deadlocks, while the agent tell(c) produces c and succeeds).

By applying this operation to the above example, we can eliminate ask(false) $\rightarrow$ stop, thus obtaining

```
auction_right(LastBid)  ← tell(LastBid ≠ quit)  ||
    bidder_a(Bs, As)  ||
    ask(true)  → tell([LastBid|As] = [HisBid|HisList'])  ||
         . . .
```

Now, we do not see any reason for not eliminating the guard ask(true) altogether. This can indeed be done via the following operation.

Definition 3.8 (*Conservative Ask Elimination*). Consider the declaration

$$\mathsf{d}:\ \mathsf{H} \leftarrow \mathsf{C}[\mathsf{ask}(\mathsf{true}) \rightarrow \mathsf{B}].$$

We can transform d into the declaration

$$\mathsf{d}':\ \mathsf{H} \leftarrow \mathsf{C}[\mathsf{B}].$$

This operation, although trivial, is subject to debate. In fact, Sahlin [1995] defines a similar operation, with the crucial distinction that the choice might still have more than one branch; in other words, in the Sahlin [1995] system we are allowed to simplify the agent C[ask(true) $\rightarrow$ A + ask(b) $\rightarrow$ B] to the agent C[A], even if b is satisfiable. Ultimàtely, we are allowed to replace the agent C[ask(true) $\rightarrow$ A + ask(true) $\rightarrow$ B] either with C[A] or with C[B]. Such an operation is clearly more widely applicable than the one we have presented, but is bound to be *incomplete*, i.e., to lead to the loss of potentially successful branches. Nevertheless, Sahlin argues that an ask elimination, such as the one defined above, is potentially too restrictive for a number of useful optimizations. We only partially agree with this statement—nevertheless, the system we propose could as easily be equipped with ask elimination as the one proposed by Sahlin (which, if employed, would lead to weaker correctness results).

In our example program, the application of these branch eliminations and conservative ask eliminations leads to the following:

```
auction_right(LastBid) ← tell(LastBid ≠ quit) ∥
    bidder_a(Bs, As) ∥
    tell([LastBid|As] = [HisBid|HisList']) ∥
    make_new_bid_b(HisBid,MyBid) ∥
       ask(MyBid = quit) → (tell(Bs = [quit|Bs']) ∥ broadcast("b quits"))
    + ask(MyBid ≠ quit) → (tell(Bs = [MyBid|Bs']) ∥
          tell(MyBid ≠ quit) ∥
          bidder_b(HisList',Bs'))
```

Via a tell elimination of tell([LastBid|As] = [HisBid|HisList']), this simplifies to

```
auction_right(LastBid) ← tell(LastBid ≠ quit) ∥
    bidder_a(Bs, As) ∥
    make_new_bid_b(LastBid,MyBid) ∥
       ask(MyBid = quit) → (tell(Bs = [quit|Bs']) ∥ broadcast("b quits"))
    + ask(MyBid ≠ quit) → (tell(Bs = [MyBid|Bs']) ∥
          tell(MyBid ≠ quit) ∥
          bidder_b(As,Bs'))
```

3.6 Distribution

A crucial operation in our transformation system is the *distribution*, which consists of bringing an agent inside a choice, as follows: from the agent $\mathsf{A} \parallel \sum_{\mathsf{i}} \mathsf{ask}(\mathsf{c_i}) \rightarrow \mathsf{B_i}$, we want to obtain the agent $\sum_{\mathsf{i}} \mathsf{ask}(\mathsf{c_i}) \rightarrow (\mathsf{A} \parallel \mathsf{B_i})$. This operation requires delicate applicability conditions, as it can easily introduce deadlocks. For instance, consider the following contrived program, D.

$$\begin{aligned} &\mathsf{p(Y)} \leftarrow \mathsf{q(X)} \parallel \mathsf{ask(X >= 0)} \rightarrow \mathsf{tell(Y = 0)} \\ &\mathsf{q(0)} \leftarrow \mathsf{stop} \end{aligned}$$

In this program, the process D.p(Y) originates the derivation $\langle \mathsf{D.p(Y), true} \rangle \rightarrow^* \langle \mathsf{D.Stop, Y = 0} \rangle$. Now, if we blindly apply the distribution operation to the first definition, we would change D into

$$\mathsf{p(Y)} \leftarrow \ \mathsf{ask(X >= 0)} \rightarrow (\mathsf{q(X)} \parallel \mathsf{tell(Y = 0)})$$

and now we have that $\langle \mathsf{D.p(Y), true} \rangle$ generates only deadlocking derivations. This situation is avoided by demanding that the agent being distributed will not be able to produce any output, unless it is completely determined which branches of the choices might be entered.

To define the applicability conditions for the distribution operation, we then need the notion of *productive configuration*. Here, and in the following, we say that a derivation $\langle \mathsf{D.A, c} \rangle \rightarrow^* \langle \mathsf{D.A', c'} \rangle$ is *maximal* if $\langle \mathsf{D.A', c'} \rangle \not\rightarrow$.

Definition 3.9 (*Productive*). Given a process D.A and a satisfiable constraint c, we say that $\langle \mathsf{D.A, c} \rangle$ is *productive* iff either it has no (finite) maximal derivations or there exists a derivation $\langle \mathsf{D.A}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D.A}', \mathsf{c}'\rangle$, such that $\mathcal{D} \models \neg(\exists_{-\tilde{Z}}\mathsf{c} \rightarrow \exists_{-\tilde{Z}}\mathsf{c}')$ where $\tilde{\mathsf{Z}} = Var(\mathsf{A})$

So a configuration is productive if its evaluation can (strictly) augment the information contained in the global store. For technical reasons, which will be clear after the next definition; we also call productive those configurations that have no finite maximal derivations.

We can now provide the definition of the distribution operation.

Definition 3.10 (*Distribution*). Let D be a set of declarations and let

$$\mathsf{d}\colon\ \mathsf{H} \leftarrow \mathsf{C}\left[\mathsf{A} \parallel \sum_{i=1}^{n} \mathsf{ask}(\mathsf{c_i}) \rightarrow \mathsf{B_i}\right]$$

be a declaration in D, where $\mathsf{e} = \mathsf{pc}(\mathsf{C}[\ ])$. The *distribution* of A in d yields the definition

$$\mathsf{d}'\colon\ \mathsf{H} \leftarrow \mathsf{C}\left[\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c_i}) \rightarrow (\mathsf{A} \parallel \mathsf{B_i})\right]$$

as a result, provided, that for every constraint c such that $Var(\mathsf{c}) \cap Var(\mathsf{d}) \subseteq Var(\mathsf{H}, \mathsf{C})$. If $\langle \mathsf{D.A}, \mathsf{c} \wedge \mathsf{e}\rangle$ is productive then both the following conditions hold:

(a) there exists at least one $i \in [1, n]$ such that $\mathcal{D} \models (\mathsf{c} \wedge \mathsf{e}) \rightarrow \mathsf{c_i}$;
(b) for each $i \in [1, n]$, either $\mathcal{D} \models (\mathsf{c} \wedge \mathsf{e}) \rightarrow \mathsf{c_i}$ or $\mathcal{D} \models (\mathsf{c} \wedge \mathsf{e}) \rightarrow \neg \mathsf{c_i}$.

Intuitively, the constraint c models the possible ways of "calling" $\mathsf{A} \parallel \sum_{i=1}^{n} \mathsf{ask}(\mathsf{c_i}) \rightarrow \mathsf{B_i}$. Condition (b) basically requires that if the store c is such that A might produce some output (that is, the configuration $\langle \mathsf{D.A}, \mathsf{c} \wedge \mathsf{e}\rangle$ is productive), then for each branch of the choice it is already determined whether we can follow it or not. This guarantees that the constraints possibly added to the store by the evaluation of A cannot influence the choice. Moreover, condition (a) guarantees that we do not apply the operation to a case such as $\mathsf{tell}(\mathsf{X} = \mathsf{a}) \parallel \mathsf{ask}(\mathsf{false}) \rightarrow \mathsf{stop}$, which is clearly wrong. If $\langle \mathsf{D.A}, \mathsf{c} \wedge \mathsf{e}\rangle$ is not productive, then we do not impose any condition, since the evaluation of $\langle \mathsf{D.A}, \mathsf{c} \wedge \mathsf{e}\rangle$ cannot affect the choice. As mentioned previously, we also call productive those configurations that have no finite maximal derivations; that is, those configurations that originate nonterminating computations only (possibly with no output). In fact, in this case we also need conditions (a) and (b), since otherwise bringing A inside the choice might transform a looping program into a deadlocking one.

The above applicability conditions are a strict improvement on the ones we presented in Etalle et al. [1998], in which we used the *required variable* concept. We now give this definition, both for simplifying the explanation for some examples and for comparing the above definition of distribution with the one in Etalle et al. [1998].

Definition 3.11 (*Required Variable*). We say that the process D.A *requires* the variable X iff, for each satisfiable constraint c such that $\mathcal{D} \models \exists_{\mathsf{X}}\mathsf{c} \leftrightarrow \mathsf{c}$, $\langle \mathsf{D.A}, \mathsf{c}\rangle$ is not productive.

In other words, the agent A requires the variable X if, at the moment the global store does not contain any information on X, A cannot produce any information that affects the variables occurring in A and has at least one finite maximal derivation. Even though the above notion is not decidable in general, it is easy to find widely applicable (decidable) sufficient conditions guaranteeing that a certain variable is required. For example, it can be seen immediately that, in our program, bidder_a(Bs, As) requires Bs. In fact, the derivation starting in bidder_a(Bs, As) suspends (without having provided any output) after one step and resumes only when more information for the variable Bs is produced.

The following remark clarifies how the concept of a required variable might be used for ensuring the applicability of the distributive operation. Its proof is straightforward.

Remark 3.12. Referring to Definition 3.10. If A requires a variable that does not occur in H, C[ ], then the distribution operation is applicable.

PROOF. In this case, there exists no constraint c such that $\mathit{Var}(\mathsf{c}) \cap \mathit{Var}(\mathsf{d}) \subseteq \mathit{Var}(\mathsf{H}, \mathsf{C})$, and such that $\langle \mathsf{D.A}, \mathsf{c} \wedge \mathsf{e} \rangle$ is productive. □

In our example, since the agent bidder_a(Bs, As) requires the variable Bs, which occurs only inside the ask guards, we can safely apply the distributive operation. The result is the following program.

```
auction_right(LastBid) ← tell(LastBid ≠ quit) ∥ make_new_bid_b(LastBid,MyBid) ∥
      ask(MyBid = quit) → tell(Bs = [quit|Bs']) ∥ broadcast("b quits") ∥ bidder_a(Bs, As)
   + ask(MyBid ≠ quit) → (tell(Bs = [MyBid|Bs']) ∥
         tell(MyBid ≠ quit) ∥
         bidder_a(Bs, As) ∥
         bidder_b(As, Bs'))
```

In this program we can now eliminate the construct tell(Bs = [MyBid|Bs']). In fact, even though the variable Bs also occurs elsewhere in the definition, we can assume it to be renamed, since it occurs only on choice-branches different from the one on which the considered agent lies. Thus, we obtain

```
auction_right(LastBid) ← tell(LastBid ≠ quit) ∥ make_new_bid_b(LastBid,MyBid) ∥
      ask(MyBid = quit) → tell(Bs = [quit|Bs']) ∥ broadcast("b quits") ∥ bidder_a(Bs, As)
   + ask(MyBid ≠ quit) → (tell(MyBid ≠ quit) ∥
         bidder_a([MyBid|Bs'], As) ∥
         bidder_b(As, Bs'))
```

Before we introduce the fold operation, let us clean up the program a bit further: we can now first apply a tell elimination to tell(Bs = [quit|Bs']), and then properly transform (by unfolding, and simplifying the result) the agent bidder_a([quit|Bs'], As) in the first ask branch. We easily obtain

```
auction_right(LastBid) ← tell(LastBid ≠ quit) ∥ make_new_bid_b(LastBid,MyBid) ∥
      ask(MyBid = quit) → broadcast("b quits") ∥ stop
   + ask(MyBid ≠ quit) → (tell(MyBid ≠ quit) ∥
         bidder_a([MyBid|Bs'], As) ∥
         bidder_b(As, Bs'))
```

The just introduced stop agent can safely be removed (see Proposition 4.2) and we are left with:

```
auction_right(LastBid) ← tell(LastBid ≠ quit) ∥ make_new_bid_b(LastBid,MyBid) ∥
        ask(MyBid = quit) → broadcast("b quits")
    + ask(MyBid ≠ quit) → (tell(MyBid ≠ quit) ∥
            bidder_a([MyBid|Bs'], As) ∥
            bidder_b(As, Bs'))
```

3.7 Folding

In the suite of transformation operations, the folding operation has a special role. This is due to the fact that it allows us to introduce recursion in a definition, often making it independent of the definitions it depended on. As previously mentioned, the applicability conditions that we use here for the folding operation do not depend on transformation history—nevertheless, we require that the declarations used to fold an agent appear in the initial program. Thus, before defining the fold operation, we need the following.

Definition 3.13. A *transformation sequence* is a sequence of programs $D_0, \dots, D_n$, in which D_0 is an *initial program* and each D_{i+1} is obtained from D_i via one of the following transformation operations: unfolding, backward instantiation, tell elimination, tell introduction, ask and tell simplification, branch elimination, conservative ask elimination, distribution, and folding.

Recall that we assume that the new declarations, introduced by using the definition introduction operation, are added once and for all to the original program D_0 before starting the transformation itself. We also need the notion of *guarding context*. Intuitively, a context C[ ] is *guarding* if the "hole" appears in the scope of an ask guard.

Definition 3.14 (*Guarding Context*). We call C[ ] a *guarding context* iff

$$C[\ ] \;=\; C'\left[\sum_{i=1}^{n} \mathrm{ask}(c_i) \rightarrow A_i\right] \quad \text{and } A_j = C''[\ ] \text{ for some } j \in [1, n].$$

So, for example, $\mathrm{ask}(c) \rightarrow (A \parallel [\ ])$ is a guarding context, while $(\mathrm{ask}(c) \rightarrow A) \parallel [\ ]$ is not. We can finally give a definition for folding:

Definition 3.15 (*Folding*). Let $D_0, \dots, D_i$, $i \geq 0$ be a transformation sequence. Consider two definitions:

$$d\colon H \leftarrow C[A] \in D_i$$
$$f\colon B \leftarrow A \in D_0$$

If C[ ] is a *guarding context*, B contains only distinct variables as arguments and $\mathit{Var}(A) \cap \mathit{Var}(C,H) \subseteq \mathit{Var}(B)$, then *folding* A in d consists of replacing d by

$$d'\colon H \leftarrow C[B] \quad \in D_{i+1}$$

(it is assumed here that d and f are suitably renamed, so that the variables they have in common are only the ones occurring in A).

In many situations this operation is also actually applicable in absence of a guarding context, as discussed below.

Remark 3.16. We can also apply the fold operation in the case where C[ ] is not a guarding context (referring to the notation in the previous definition), provided that the definition H ← C[A] was not modified or used during the transformation. In fact, in this case we can simply assume that the original definition of H ← C[A] contained a dummy ask guard, as in

$$\mathsf{H} \leftarrow \mathsf{ask(true)} \rightarrow \mathsf{C[A]},$$

that the folding operation is applied to this definition, and that the guard ask(true) will eventually be removed by an ask elimination operation.

Actually, in many cases this reasoning can also be applied to definitions that *are* used during the transformation. This kind of folding is called *propagation folding* (as opposed to *recursive* folding). It is not employed to introduce recursion, but to *propagate* the efficiency gained (we hope) by the transformation to other contexts. Transformation systems usually provide a special condition for the propagation folding operation. For instance, in Tamaki and Sato [1984], a distinction is made between *new* and *old* predicates. We decided not to do so here. This allows us to have a definition of a folding operation that is particularly simple.

We refer to the end of Example 6.2 for an instance of an application of folding without a guarding context.

The reach of the folding operation is best shown via our example. We can now fold auction_left(MyBid) in the above definition, and obtain

```
auction_right(LastBid) ← tell(LastBid ≠ quit) || make_new_bid_b(LastBid,MyBid) ||
      ask(MyBid = quit) → broadcast("b quits")
    + ask(MyBid ≠ quit) → auction_left(MyBid)
```

Now, by performing an identical optimization on auction_left, we can also obtain

```
auction_left(LastBid) ← tell(LastBid ≠ quit) || make_new_bid_a(LastBid,MyBid) ||
      ask(MyBid = quit) → broadcast("a quits")
    + ask(MyBid ≠ quit) → auction_right(MyBid)
```

This part of the transformation shows in a striking way one of the main benefits of the folding operation: saving synchronization points. Notice that in the initial program the two bidders had to "wait" for each other. In principle, they were working in parallel, but in practice they were always acting sequentially, since we always had to wait for the bid of the competitor. The transformation allowed us to discover this sequentiality and to obtain an equivalent program in which sequentiality is exploited to eliminate all suspension points—which are known to be one of the major overhead sources. Furthermore, the transformation allows a drastic saving of computational *space*. In fact, in the initial definition the parallel composition of the two bidders leads to the construction of two lists containing all the bids done so far. After the transformation, we have a definition that does not build the list any longer, and by exploiting a straightforward optimization can employ only *constant* space.

Concerning the syntax of the operation, in our setting the folding operation reduces to a mere replacement. To people familiar with this operation, this might seem restrictive: We might also wish to apply folding in the case where the definition to be folded contains an *instance* of A, i.e., when d has the form $\mathsf{H} \leftarrow \mathsf{C}[\mathsf{A}\sigma]$ (in this case the folding operation is applicable only if σ satisfies specific conditions described in Tamaki and Sato [1984] for logic programs and in Etalle e Gabbrielli [1996] for CLP). This extended operation would actually correspond to the (most) common definition of folding, as in Tamaki and Sato [1984] Etalle and Gabbrielli [1996]; and in Bensaou and Gùessarian [1998]. In our system such an extended operation is not needed formally, as it can be obtained by combining the folding operation with the tell introduction.

In fact, assume that we would like to fold the definition

$$\mathsf{d}\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{A}\sigma] \quad \in \mathsf{D}_i$$

by using the definition

$$\mathsf{f}\colon \mathsf{B} \leftarrow \mathsf{A} \quad \in \mathsf{D}_0$$

In the first place, via a tell introduction, we can modify definition d as follows:

$$\mathsf{d}^*\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{A} \parallel \mathsf{tell}(\tilde{\mathsf{X}} = \tilde{\mathsf{X}}\sigma)],$$

Clearly, we assume here that $\tilde{\mathsf{X}}$ and σ fulfill the applicability conditions given in Definition 3.2. Then, via a normal folding operation, we obtain

$$\mathsf{d}^{**}\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{X}} = \tilde{\mathsf{X}}\sigma)]$$

(provided that the applicability conditions for folding are satisfied), which is equivalent to the definition

$$\mathsf{d}'\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{B}\sigma].$$

obtained in the case of the folding operation as defined in Tamaki and Sato [1984]; Etalle and Gabbrielli [1996]; and Bensaou and Guessarian [1998]. Actually, in case the constraint domain admits most general unifiers, the definition d' can be obtained from d^{**} by using a tell elimination operation (in this case also we assume that the applicability conditions for the tell elimination are satisfied).

For the sake of simplicity, we do not give the explicit definition of this (derived) extended folding operation and of its applicability conditions. Hence the occurrences of this operation in the last example of Section 6 have to be considered as shorthand for the sequence of operations described above.

4. CORRECTNESS

Any transformation system must be useful (i.e., allow useful transformations and optimization) and, most importantly, *correct*, i.e., it must guarantee that the resulting program is in some sense equivalent to the one we started with.

Having the transition system in Table I at hand, we now provide the intended semantics to be preserved by the transformation system by defining a suitable notion of "observables." We start with the following definition which takes into account terminating and failed computations only. In the next section we also consider nonterminating computations. Here and in the sequel, we say that a constraint c is *satisfiable* iff $\mathcal{D} \models \exists\, \mathsf{c}$.

Definition 4.1 (*Observables*). Let D.A be a CCP process. We define

$$\begin{array}{lll}
\mathcal{O}(\mathsf{D.A}) = & \{\langle \mathsf{c}, \exists_{-Var(\mathsf{A,c})}\mathsf{d}, \mathsf{ss}\rangle & \mid \mathsf{c} \text{ and } \mathsf{d} \text{ are satisfiable, and there exists} \\
& & \quad \text{a derivation } \langle \mathsf{D.A}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D.Stop}, \mathsf{d}\rangle\} \\
& \cup & \\
& \{\langle \mathsf{c}, \exists_{-Var(\mathsf{A,c})}\mathsf{d}, \mathsf{dd}\rangle & \mid \mathsf{c} \text{ and } \mathsf{d} \text{ are satisfiable, and there exists} \\
& & \quad \text{a derivation } \langle \mathsf{D.A}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D.B}, \mathsf{d}\rangle \not\rightarrow, \\
& & \quad \mathsf{B} \neq \mathsf{Stop}\} \\
& \cup & \\
& \{\langle \mathsf{c}, \mathsf{false}, \mathsf{ff}\rangle & \mid \mathsf{c} \text{ is satisfiable, and there exists} \\
& & \quad \text{a derivation } \langle \mathsf{D.A}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D.B}, \mathsf{false}\rangle\}.
\end{array}$$

Thus, what we observe are the results of terminating computations (if consistent), abstracting from the values of the local variables in the results, and distinguishing the successful computations from the deadlocked ones (by using the termination modes ss and dd, respectively). We also observe failed computations, i.e., those computations that produce an inconsistent store.

Having defined a formal semantics for our paradigm, we can now define more precisely the notion of *correctness* for the transformation system: we say that a transformation sequence $\mathsf{D}_0, \ldots, \mathsf{D}_\mathsf{n}$ is *partially correct* iff, for each agent A, we have that

$$\mathcal{O}(\mathsf{D}_0.\mathsf{A}) \supseteq \mathcal{O}(\mathsf{D}_\mathsf{n}.\mathsf{A})$$

holds; that is, nothing is added to the semantics of the initial program. We also say that $\mathsf{D}_0, \ldots, \mathsf{D}_\mathsf{n}$ is *complete* iff, for each agent A, we have that

$$\mathcal{O}(\mathsf{D}_0.\mathsf{A}) \subseteq \mathcal{O}(\mathsf{D}_\mathsf{n}.\mathsf{A})$$

holds; that is, no semantic information is lost during the transformation. Finally a transformation sequence is called *totally correct* iff it is both partially correct and complete.

In the following we prove that the our transformation system is *totally correct*. As previously mentioned, for the sake of readability some proofs are only sketched in, and their full versions can be found in the Appendix.

The proof of this result is originally inspired by Tamaki and Sato [1984] for pure logic programs, and we have retained some of its notation, in particular we also use the notions of weight and of split derivation. Of course, the similarities do not go any further, as demonstrated by the fact that in our transformation system the applicability conditions of folding operation do not depend on transformation history (while allowing the introduction of recursion), and that the folding definitions are allowed to be recursive (the distinction between P_{new} and P_{old} in Tamaki and Sato [1984] is now superfluous).

We start with the following proposition, which allows us to eliminate stop agents in programs.

PROPOSITION 4.2. *For any agent* A *and set of declarations* D, $\mathcal{O}(\mathsf{D.A} \parallel \mathsf{stop}) = \mathcal{O}(\mathsf{D.A})$.

PROOF. The proof follows immediately from the definition of observables by noting that, according to rules **R1–R4**, the agent stop has no transition and $\langle \mathsf{D.A} \parallel \mathsf{stop}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D.B} \parallel \mathsf{stop}, \mathsf{d}\rangle$ iff $\langle \mathsf{D.A}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D.B}, \mathsf{d}\rangle$, where, obviously, $\mathsf{B} \parallel \mathsf{stop}$ is equal to Stop iff $\mathsf{B} = \mathsf{Stop}$ (recall that Stop is the generic agent containing only $\parallel$ and stop). □

The following notion of mode is useful for shortening the notation.

Definition 4.3. Let $\mathsf{D}_0, \ldots, \mathsf{D}_n$ be a transformation sequence, A be an agent, and d a constraint. We define the *mode* $\mathsf{m}(\mathsf{A}, \mathsf{d})$ of the agent A w.r.t. the constraint d, as follows:

$$\mathsf{m}(\mathsf{A}, \mathsf{d}) = \begin{cases} \mathsf{ss} & \text{if } \mathsf{d} \text{ is satisfiable and } \quad \mathsf{A} = \mathsf{Stop} \\ \mathsf{dd} & \text{if } \mathsf{d} \text{ is satisfiable, } \quad \langle \mathsf{D}_0.\mathsf{A}, \mathsf{d}\rangle \not\rightarrow \quad \text{and} \quad \mathsf{A} \neq \mathsf{Stop} \\ \mathsf{ff} & \text{if } \mathsf{d} \text{ is not satisfiable} \end{cases}$$

Note that the notion of mode does not depend on the set of declarations D_i we are considering; that is, in the above definition we could equivalently use D_i rather than D_0. This is the content of the following:

PROPOSITION 4.4. *Let* $\mathsf{D}_0, \ldots, \mathsf{D}_n$ *be a transformation sequence,* A *be an agent and* d *a constraint. Then* $\langle \mathsf{D}_0.\mathsf{A}, \mathsf{d}\rangle \not\rightarrow$ *iff* $\langle \mathsf{D}_i.\mathsf{A}, \mathsf{d}\rangle \not\rightarrow$, *for any* $i \in [1, n]$.

PROOF. Immediate, by observing that a procedure call can be evaluated in D_0 iff it can be evaluated in D_i for any $i \in [1, n]$. □

In what follows, we refer to a fixed *transformation sequence* $\mathsf{D}_0, \ldots, \mathsf{D}_n$. We start with the following result, concerning partial correctness.

PROPOSITION 4.5 (Partial Correctness). *If, for each agent* A, $\mathcal{O}(\mathsf{D}_0.\mathsf{A}) = \mathcal{O}(\mathsf{D}_i.\mathsf{A})$ *holds, then, for each agent* A, $\mathcal{O}(\mathsf{D}_i.\mathsf{A}) \supseteq \mathcal{O}(\mathsf{D}_{i+1}.\mathsf{A})$.

PROOF (Sketch). We show that given an agent A and a satisfiable constraint c_I, if there exists a derivation $\xi = \langle \mathsf{D}_{i+1}.\mathsf{A}, \mathsf{c}_I\rangle \rightarrow^* \langle \mathsf{D}_{i+1}.\mathsf{B}, \mathsf{c}_F\rangle$, with $\mathsf{m}(\mathsf{B}, \mathsf{c}_F) \in \{\mathsf{ss}, \mathsf{dd}, \mathsf{ff}\}$, then there also exists a derivation $\xi' = \langle \mathsf{D}_i.\mathsf{A}, \mathsf{c}_I\rangle \rightarrow^* \langle \mathsf{D}_i.\mathsf{B}', \mathsf{c}'_F\rangle$ with $\exists_{-Var(\mathsf{A},\mathsf{c}_I)}\mathsf{c}'_F = \exists_{-Var(\mathsf{A},\mathsf{c}_I)}\mathsf{c}_F$ and $\mathsf{m}(\mathsf{B}', \mathsf{c}'_F) = \mathsf{m}(\mathsf{B}, \mathsf{c}_F)$. By Definition 4.1, this implies the thesis. The proof is by induction on the length l of the derivation.

($l = 0$). In this case, $\xi = \langle \mathsf{D}_{i+1}.\mathsf{A}, \mathsf{c}_I\rangle$. By the definition, $\langle \mathsf{D}_i.\mathsf{A}, \mathsf{c}_I\rangle$ is also a derivation of length 0, and then the thesis holds.

($l > 0$). If the first step of derivation ξ does not use rule **R4**, then the proof follows from the inductive hypothesis.

Now assume that the first step of derivation ξ uses rule **R4** and let $\mathsf{d}' \in \mathsf{D}_{i+1}$ be the declaration used in the first step of ξ. If d' was not modified in the transformation step from D_i to D_{i+1} (that is, $\mathsf{d}' \in \mathsf{D}_i$), then the result follows from the inductive hypothesis. We then assume that $\mathsf{d}' \notin \mathsf{D}_i$, d' is then the result of the transformation operation applied to obtain D_{i+1}. The proof proceeds by distinguishing various cases according to the operation itself. Here we consider only the operations of unfolding, tell elimination, tell introduction, and folding. The other cases are deferred to the Appendix.

Unfolding: If d' is the result of an unfolding operation, then the proof is immediate.

Tell elimination and introduction: If d' is the result of a tell elimination or of a tell introduction, the thesis follows from a straightforward analysis of the possible derivations that use d or d'. First, observe that for any derivation that uses a declaration $H \leftarrow C[\mathsf{tell}(\tilde{s} = \tilde{t}) \parallel B]$, we can construct another derivation such that the agent $\mathsf{tell}(\tilde{s} = \tilde{t})$ is evaluated before B. Moreover, for any constraint c such that $\exists_{dom(\sigma)} c = \exists_{dom(\sigma)} c\sigma$ (where σ is a relevant, most general, unifier of $\tilde{s}$ and $\tilde{t}$), there exists a derivation step $\langle D_i.B_1\sigma, c\sigma\rangle \rightarrow \langle D_i.B_2\sigma, c'\rangle$ if and only if there exists a derivation step $\langle D_i.B_1, c \wedge (\tilde{s} = \tilde{t})\rangle \rightarrow \langle D_i.B_2, c''\rangle$, where, for some constraint e, $c' = e\sigma$, $c'' = e \wedge (\tilde{s} = \tilde{t})$, and therefore $c' = \exists_{dom(\sigma)} c''$. Finally, since, by definition, σ is idempotent and the variables in the domain of σ do not occur either in C[ ] or in H for any constraint e, we have that $\exists_{-Var(A,c_I)} e\sigma = \exists_{-Var(A,c_I)}(e \wedge (\tilde{s} = \tilde{t}))$.

Folding: If d' is the result of a folding, then let

—d: $q(\tilde{r}) \leftarrow C[H]$ be the folded declaration ($\in D_i$),
—f: $p(\tilde{X}) \leftarrow H$ be the folding declaration ($\in D_0$),
—d′: $q(\tilde{r}) \leftarrow C[p(\tilde{X})]$ be the result of the folding operation ($\in D_{i+1}$)

where, by hypothesis, $Var(d) \cap Var(\tilde{X}) \subseteq Var(H)$ and $Var(H) \cap (Var(\tilde{r}) \cup Var(C)) \subseteq Var(\tilde{X})$. In this case, $\xi = \langle D_{i+1}.C_I[q(\tilde{v})], c_I\rangle \rightarrow \langle D_{i+1}.C_I[C[p(\tilde{x})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_{i+1}.B, c_F\rangle$, and we can assume, without loss of generality, that $Var(C_I[q(\tilde{v})], c_I) \cap Var(H) = \emptyset$.

By the inductive hypothesis, there exists a derivation

$$\chi = \langle D_i.C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_i.B'', c''_F\rangle,$$

with $\exists_{-Var(C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v}=\tilde{r})], c_I)} c''_F = \exists_{-Var(C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v}=\tilde{r})], c_I)} c_F$ and

$$m(B'', c''_F) = m(B, c_F). \tag{1}$$

Since $Var(C_I[q(\tilde{v})], c_I) \subseteq Var(C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)$, we have that

$$\exists_{-Var(C_I[q(\tilde{v})], c_I)} c''_F = \exists_{-Var(C_I[q(\tilde{v})], c_I)} c_F. \tag{2}$$

Since by hypothesis for any agent A′, $\mathcal{O}(D_0.A') = \mathcal{O}(D_i.A')$, there exists a derivation

$$\xi_0 = \langle D_0.C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_0.B_0, c_0\rangle$$

such that $\exists_{-Var(C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v}=\tilde{r})], c_I)} c_0 = \exists_{-Var(C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v}=\tilde{r})], c_I)} c''_F$ and $m(B_0, c_0) = m(B'', c''_F)$.

By (1), (2), and since $Var(C_I[q(\tilde{v})], c_I) \subseteq Var(C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)$, we have that

$$\exists_{-Var(C_I[q(\tilde{v})], c_I)} c_0 = \exists_{-Var(C_I[q(\tilde{v})], c_I)} c_F \text{ and } m(B_0, c_0) = m(B, c_F). \tag{3}$$

Let f': $p(\tilde{X}') \leftarrow H'$ be an appropriate renaming of f, which renames only the variables in $\tilde{X}$, such that $\mathit{Var}(d) \cap \mathit{Var}(f') = \emptyset$ (note that this is possible, since $\mathit{Var}(H) \cap (\mathit{Var}(\tilde{r}) \cup \mathit{Var}(C)) \subseteq \mathit{Var}(\tilde{X})$). By hypothesis, $\mathit{Var}(C_I[q(\tilde{v})], c_I) \cap \mathit{Var}(H) = \emptyset$. Then, without loss of generality, we can assume that $\mathit{Var}(\xi_0) \cap \mathit{Var}(f') \neq \emptyset$ if and only if the procedure call $p(\tilde{X})$ is evaluated, in which case declaration f' is used. Thus there exists a derivation

$$\langle D_0.C_I[C[H' \parallel \mathsf{tell}(\tilde{X} = \tilde{X}')] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_0.B'_0, c_0\rangle,$$

where $m(B'_0, c_0) = m(B_0, c_0)$. By (3), we have

$$m(B'_0, c_0) = m(B, c_F). \tag{4}$$

We now show that we can substitute H for $H' \parallel \mathsf{tell}(\tilde{X} = \tilde{X}')$ in the previous derivation. Since f': $p(\tilde{X}') \leftarrow H'$ is a renaming of f: $p(\tilde{X}) \leftarrow H$, the equality $\tilde{X} = \tilde{X}'$ is a conjunction of equations involving only distinct variables. Then, by replacing the variables $\tilde{X}$ with $\tilde{X}'$, and vice versa in the previous derivation, we obtain the derivation $\chi_0 = \langle D_0.C_I[C[H \parallel \mathsf{tell}(\tilde{X}' = \tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_0.B''_0, c'_0\rangle$ where $\exists_{-\mathit{Var}(C_I[C[H \parallel \mathsf{tell}(\tilde{X}' = \tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)} c'_0 = \exists_{-\mathit{Var}(C_I[C[H \parallel \mathsf{tell}(\tilde{X}' = \tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)} c_0$ and $m(B''_0, c'_0) = m(B'_0, c_0)$.

From (4), it follows that

$$m(B''_0, c'_0) = m(B, c_F). \tag{5}$$

Then, from (3), and since $\mathit{Var}(C_I[q(\tilde{v})], c_I) \subseteq \mathit{Var}(C_I[C[H \parallel \mathsf{tell}(\tilde{X}' = \tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)$, we obtain

$$\exists_{-\mathit{Var}(C_I[q(\tilde{v})], c_I)} c'_0 = \exists_{-\mathit{Var}(C_I[q(\tilde{v})], c_I)} c_F. \tag{6}$$

Moreover, we can drop the constraint $\mathsf{tell}(\tilde{X}' = \tilde{X})$, since the declarations used in the derivation are renamed apart and, by construction, $\mathit{Var}(C_I[C[H] \parallel \mathsf{tell}(\tilde{r} = \tilde{v})], c_I) \cap \mathit{Var}(\tilde{X}') = \emptyset$. Therefore, there exists a derivation $\langle D_0.C_I[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_0.\bar{B}_0, \bar{c}_0\rangle$ which performs exactly the same steps of χ_0, except (possibly) for the evaluation of $\mathsf{tell}(\tilde{X}' = \tilde{X})$, and such that $\exists_{-\mathit{Var}(C_I[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)} \bar{c}_0 = \exists_{-\mathit{Var}(C_I[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)} c'_0$ and $m(\bar{B}_0, \bar{c}_0) = m(B''_0, c'_0)$. From (5), (6), and since $\mathit{Var}(C_I[q(\tilde{v})], c_I) \subseteq \mathit{Var}(C_I[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)$, it follows that

$$m(\bar{B}_0, \bar{c}_0) = m(B, c_F) \text{ and } \exists_{-\mathit{Var}(C_I[q(\tilde{v})], c_I)} \bar{c}_0 = \exists_{-\mathit{Var}(C_I[q(\tilde{v})], c_I)} c_F. \tag{7}$$

Since $\mathcal{O}(D_0.A') = \mathcal{O}(D_i.A')$ holds by hypothesis for any agent A', there exists a derivation

$$\langle D_i.C_I[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_i.B', c'_F\rangle$$

where

$$\exists_{-\mathit{Var}(C_I[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)} c'_F = \exists_{-\mathit{Var}(C_I[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)} \bar{c}_0$$

and $m(B', c'_F) = m(\bar{B}_0, \bar{c}_0)$.

From (7) and since $\mathit{Var}(C_I[q(\tilde{v})], c_I) \subseteq \mathit{Var}(C_I[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I)$, we obtain

$$m(B', c'_F) = m(B, c_F) \text{ and } \exists_{-\mathit{Var}(C_I[q(\tilde{v})], c_I)} c'_F = \exists_{-\mathit{Var}(C_I[q(\tilde{v})], c_I)} c_F. \tag{8}$$

Finally, since d: $q(\tilde{r}) \leftarrow C[H] \in D_i$, there exists a derivation

$$\xi' = \langle D_i.C_I[q(\tilde{v})], c_I\rangle \rightarrow \langle D_i.C_I[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_i.B', c'_F\rangle$$

and then the thesis follows from (8). □

In order to prove total correctness, we need the following:

Definition 4.6 (*Weight*). Let ξ be a derivation. We denote by $wh(\xi)$ the number of derivation steps in ξ that use rule **R2**. Given an agent A and a pair of satisfiable constraints c, d, we then define the *success weight* $\mathsf{w_{ss}}(\mathsf{A}, \mathsf{c}, \mathsf{d})$ of the agent A w.r.t. the constraints c and d, as follows:

$$\begin{aligned}\mathsf{w_{ss}}(\mathsf{A}, \mathsf{c}, \mathsf{d}) = \min\{n \mid n = wh(\xi) \text{ and } \xi \text{ is a derivation}\\ \langle \mathsf{D_0.A}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D_0.Stop}, \mathsf{d'}\rangle \not\rightarrow\\ \text{with } \exists_{-Var(\mathsf{A},\mathsf{c})}\mathsf{d'} = \exists_{-Var(\mathsf{A},\mathsf{c})}\mathsf{d}\}\end{aligned}$$

Analogously, we define the *deadlock weight* $\mathsf{w_{dd}}(\mathsf{A}, \mathsf{c}, \mathsf{d})$ of the agent A w.r.t. the constraints c and d,

$$\begin{aligned}\mathsf{w_{dd}}(\mathsf{A}, \mathsf{c}, \mathsf{d}) = \min\{n \mid n = wh(\xi) \text{ and } \xi \text{ is a derivation}\\ \langle \mathsf{D_0.A}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D_0.B}, \mathsf{d'}\rangle \not\rightarrow\\ \text{with } \mathsf{B} \neq \mathsf{Stop} \text{ and } \exists_{-Var(\mathsf{A},\mathsf{c})}\mathsf{d'} = \exists_{-Var(\mathsf{A},\mathsf{c})}\mathsf{d}\}\end{aligned}$$

and the *failure weight* $\mathsf{w_{ff}}(\mathsf{A}, \mathsf{c}, \mathsf{d'})$ of the agent A w.r.t. constraints c and d′

$$\begin{aligned}\mathsf{w_{ff}}(\mathsf{A}, \mathsf{c}, \mathsf{d'}) = min\{n \mid n = wh(\xi) \text{ and } \xi \text{ is a derivation } \langle \mathsf{D_0.A}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D_0.B}, \mathsf{d'}\rangle\\ \text{with } \mathsf{d'} = \mathsf{false}\}\end{aligned}$$

Notice that $\mathsf{w_{ss}}(\mathsf{A}, \mathsf{c}, \mathsf{d'})$ is undefined, in case there is no successful derivation corresponding to the given constraints (and analogously for $\mathsf{w_{dd}}$ and $\mathsf{w_{ff}}$). Also, both $\mathsf{w_{ss}}(\mathsf{A}, \mathsf{c}, \mathsf{false})$ and $\mathsf{w_{dd}}(\mathsf{A}, \mathsf{c}, \mathsf{false})$ are undefined, as the success and deadlock weight consider only nonfailed derivations (i.e., derivations that do not produce the constraint, *false*).

As previously mentioned, this notion of weight is rather different from the one in Tamaki and Sato [1984], since the latter is based on the number of nodes in a proof tree for an atom by taking into account the fact that the predicate symbol appearing in that atom is "new" or "old."

In the total correctness proof, we also make use of the concept of *split derivations*. Intuitively, these are derivations that can be split into two parts: the first one, up to the first ask evaluation, is performed in the program $\mathsf{D_i}$, while the second is carried out in $\mathsf{D_0}$.

Definition 4.7 (*Split Derivation*). Let $\mathsf{D_0}, \dots, \mathsf{D_i}$ be a transformation sequence. We call a derivation in $\mathsf{D_i} \cup \mathsf{D_0}$ a *successful split derivation* if it has the form

$$\langle \mathsf{D_i.A_1}, \mathsf{c_1}\rangle \rightarrow^* \langle \mathsf{D_i.A_m}, \mathsf{c_m}\rangle \rightarrow \langle \mathsf{D_0.A_{m+1}}, \mathsf{c_{m+1}}\rangle \rightarrow^* \langle \mathsf{D_0.Stop}, \mathsf{c_n}\rangle \not\rightarrow$$

where $\mathsf{c_n}$ is a satisfiable constraint, $m \in [1, n]$,[5] and the following conditions hold:

(a) the first $m - 1$ derivation steps do not use rule **R2**;

(b) the m-th derivation step $\langle \mathsf{D_i.A_m}, \mathsf{c_m}\rangle \rightarrow \langle \mathsf{D_0.A_{m+1}}, \mathsf{c_{m+1}}\rangle$ uses rule **R2**;

(c) $\mathsf{w_{ss}}(\mathsf{A_1}, \mathsf{c_1}, \mathsf{c_n}) > \mathsf{w_{ss}}(\mathsf{A_{m+1}}, \mathsf{c_{m+1}}, \mathsf{c_n})$.

[5]If $m = n$, we can write either $\langle \mathsf{D_i.Stop}, \mathsf{c_n}\rangle$ or $\langle \mathsf{D_0.Stop}, \mathsf{c_n}\rangle$ to denote the last configuration of the derivation.

A *deadlocked split derivation* is defined analogously by replacing $\mathsf{w_{ss}}$ for $\mathsf{w_{dd}}$ and Stop for a generic agent $\mathsf{B} \neq \mathsf{Stop}$ in the last configuration of the derivation above. Finally, a *failed split derivation* is defined by replacing $\mathsf{w_{ss}}$ for $\mathsf{w_{ff}}$ and Stop for a generic agent (which is not necessarily terminated) and by assuming that $\mathsf{c_n} = \mathsf{false}$ in the last configuration of the derivation above.

In the following, we call split derivations successful, deadlocked, and failed split derivations. The previous definition is inspired by the definition of the *descent clause* of Kawamura and Kanamori [1988]; however, here we use a different notion of weight and rather different conditions on them. We need one final concept.

Definition 4.8. We call the program $\mathsf{D_i}$ *weight complete* iff, for any agent A, for any satisfiable constraint c and for any constraint d, the following hold: if there exists a derivation

$$\langle \mathsf{D_0.A, c}\rangle \rightarrow^* \langle \mathsf{D_0.B, d}\rangle$$

such that $\mathsf{m(B, d)} \in \{\mathsf{ss},\ \mathsf{dd},\ \mathsf{ff}\}$, then there exists a split derivation in $\mathsf{D_i} \cup \mathsf{D_0}$

$$\langle \mathsf{D_i.A, c}\rangle \rightarrow^* \langle \mathsf{D_0.B', d'}\rangle$$

where $\exists_{-Var(\mathsf{A,c})}\mathsf{d'} = \exists_{-Var(\mathsf{A,c})}\mathsf{d}$ and $\mathsf{m(B', d')} = \mathsf{m(B, d)}$.

So $\mathsf{D_i}$ is weight-complete if we can reconstruct the semantics of $\mathsf{D_0}$ by using only (successful, deadlocked, and failed) split derivations in $\mathsf{D_i} \cup \mathsf{D_0}$. We now show that if $\mathsf{D_i}$ is weight-complete, then no observables are lost during the transformation (i.e., the transformation is complete). This is the content of the following:

PROPOSITION 4.9. *If* $\mathsf{D_i}$ *is weight-complete, then, for any agent* A, $\mathcal{O}(\mathsf{D_0.A}) \subseteq \mathcal{O}(\mathsf{D_i.A})$.

PROOF. We consider only the case of successful derivations, since the case of deadlocked (failed) derivations can be proved analogously by considering the notion of deadlock (failure) weight and deadlocked (failed) split derivation. Assume that there exists a (finite, successful) derivation $\langle \mathsf{D_0.A, c}\rangle \rightarrow^* \langle \mathsf{D_0.Stop, d}\rangle$. We show, by induction on the success weight of $(\mathsf{A, c, d})$, that there exists a derivation $\langle \mathsf{D_i.A, c}\rangle \rightarrow^* \langle \mathsf{D_i.Stop, d'}\rangle$, where $\exists_{-Var(\mathsf{A,c})}\mathsf{d'} = \exists_{-Var(\mathsf{A,c})}\mathsf{d}$.

Base Case. If $\mathsf{w_{ss}(A, c, d)} = 0$, then, since $\mathsf{D_i}$ is weight-complete, from Definitions 4.7 and 4.8 it follows that there exists a (successful) split derivation in $\mathsf{D_i} \cup \mathsf{D_0}$ of the form $\langle \mathsf{D_i.A, c}\rangle \rightarrow^* \langle \mathsf{D_i.Stop, d'}\rangle$, where $\exists_{-Var(\mathsf{A,c})}\mathsf{d'} = \exists_{-Var(\mathsf{A,c})}\mathsf{d}$, rule **R2** is not used, and hence each derivation step is done in $\mathsf{D_i}$.

Inductive Case. Assume that $\mathsf{w_{ss}(A, c, d)} = \mathsf{n}$. Since $\mathsf{D_i}$ is weight-complete, there exists a (successful) split derivation in $\mathsf{D_i} \cup \mathsf{D_0}$

$$\xi\colon \langle \mathsf{D_i.A, c}\rangle \rightarrow^* \langle \mathsf{D_0.Stop, d'}\rangle,$$

where $\exists_{-Var(\mathsf{A,c})}\mathsf{d'} = \exists_{-Var(\mathsf{A,c})}\mathsf{d}$. If rule **R2** is not used in ξ, then the proof is the same as in the previous case. Otherwise ξ has the form

$$\langle \mathsf{D_i.A, c}\rangle \rightarrow^* \langle \mathsf{D_i.A_m, c_m}\rangle \rightarrow \langle \mathsf{D_0.A_{m+1}, c_{m+1}}\rangle \rightarrow^* \langle \mathsf{D_0.Stop, d'}\rangle$$

where $w_{ss}(A, c, d') > w_{ss}(A_{m+1}, c_{m+1}, d')$. Let ξ' be the derivation

$$\xi': \langle D_i.A, c\rangle \rightarrow^* \langle D_i.A_m, c_m\rangle \rightarrow \langle D_i.A_{m+1}, c_{m+1}\rangle.$$

By the inductive hypothesis, there exists a derivation

$$\xi'': \langle D_i.A_{m+1}, c_{m+1}\rangle \rightarrow^* \langle D_i.Stop, d''\rangle$$

where $\exists_{-Var(A_{m+1},c_{m+1})}d'' = \exists_{-Var(A_{m+1},c_{m+1})}d'$. Without loss of generality, we can assume that $Var(\xi') \cap Var(\xi'') = Var(A_{m+1}, c_{m+1})$, and hence there exists a derivation

$$\langle D_i.A, c\rangle \rightarrow^* \langle D_i.Stop, d''\rangle.$$

Finally, by our hypothesis on the variables and by construction,

$$\begin{aligned}
&\exists_{-Var(A,c)}d''\\
&\quad = \exists_{-Var(A,c)}(c_{m+1} \wedge \exists_{-Var(A_{m+1},c_{m+1})}d'')\\
&\quad = \exists_{-Var(A,c)}(c_{m+1} \wedge \exists_{-Var(A_{m+1},c_{m+1})}d')\\
&\quad = \exists_{-Var(A,c)}d'\\
&\quad = \exists_{-Var(A,c)}d
\end{aligned}$$

which concludes the proof. □

Before proving the total correctness result, we need some technical lemmas. Here and in the following, we use the notation w_t (with $t \in \{ss, dd, ff\}$) as a shorthand for indicating the success weight w_{ss}, the deadlock weight w_{dd}, and the failure weight w_{ff}.

LEMMA 4.10. *Let* $q(\tilde{r}) \leftarrow H \in D_0$, $t \in \{ss, dd, ff\}$ *and let* $C[\ ]$ *be a context. For any satisfiable constraint* c *and for any constraint* c'*, such that* $Var(C[q(\tilde{t})], c) \cap Var(\tilde{r}) = \emptyset$ *and* $w_t(C[q(\tilde{t})], c, c')$ *is defined, there exists a constraint* d' *such that* $w_t(C[q(\tilde{r}) \parallel tell(\tilde{t} = \tilde{r})], c, d') \leq w_t(C[q(\tilde{t})], c, c')$ *and* $\exists_{-Var(C[q(\tilde{t})],c)}d' = \exists_{-Var(C[q(\tilde{t})],c)}c'$.

PROOF. Immediate. □

LEMMA 4.11. *Let* $q(\tilde{r}) \leftarrow H \in D_0$ *and* $t \in \{ss, dd, ff\}$. *For any context* $C_l[\ ]$, *any satisfiable constraint* c *and for any constraint* c'*, the following holds:*

(1) *If* $Var(H) \cap Var(C_l, c) \subseteq Var(\tilde{r})$ *and* $w_t(C_l[q(\tilde{r})], c, c')$ *is defined, then there exists a constraint* d' *such that* $Var(d') \subseteq Var(C_l[H], c)$, $w_t(C_l[H], c, d') \leq w_t(C_l[q(\tilde{r})], c, c')$, *and* $\exists_{-Var(C_l[q(\tilde{r})],c)}d' = \exists_{-Var(C_l[q(\tilde{r})],c)}c'$.
(2) *If* $Var(H) \cap Var(C_l, c) \subseteq Var(\tilde{r})$, $Var(c') \cap Var(\tilde{r}) \subseteq Var(C_l[H], c)$ *and* $w_t(C_l[H], c, c')$ *is defined, then there exists a constraint* d' *such that* $w_t(C_l[q(\tilde{r})], c, d') \leq w_t(C_l[H], c, c')$ *and* $\exists_{-Var(C_l[q(\tilde{r})],c)}d' = \exists_{-Var(C_l[q(\tilde{r})],c)}c'$.

PROOF. Immediate. □

The following lemma is crucial in the proof of completeness.

LEMMA 4.12. *Let* $0 \leq i \leq n$, $t \in \{ss, dd, ff\}$, cl: $q(\tilde{r}) \leftarrow H \in D_i$, *and let* cl': $q(\tilde{r}) \leftarrow H'$ *be the corresponding declaration in* D_{i+1} *(in the case* $i < n$*). For any context* $C_l[\ ]$ *and any satisfiable constraint* c *and any constraint* c' *the following holds:*

(1) *If* $Var(\mathsf{H}) \cap Var(\mathsf{C_l}, \mathsf{c}) \subseteq Var(\tilde{\mathsf{r}})$ *and* $\mathsf{w_t}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})], \mathsf{c}, \mathsf{c}')$ *is defined, then there exists a constraint* d'*, such that* $Var(\mathsf{d}') \subseteq Var(\mathsf{C_l}[\mathsf{H}], \mathsf{c})$*,* $\mathsf{w_t}(\mathsf{C_l}[\mathsf{H}], \mathsf{c}, \mathsf{d}') \leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})], \mathsf{c}, \mathsf{c}')$ *and* $\exists_{-Var(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{d}' = \exists_{-Var(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{c}'$*;*

(2) *If* $Var(\mathsf{H}, \mathsf{H}') \cap Var(\mathsf{C_l}, \mathsf{c}) \subseteq Var(\tilde{\mathsf{r}})$*,* $Var(\mathsf{c}') \cap Var(\tilde{\mathsf{r}}) \subseteq Var(\mathsf{C_l}[\mathsf{H}], \mathsf{c})$ *and* $\mathsf{w_t}(\mathsf{C_l}[\mathsf{H}], \mathsf{c}, \mathsf{c}')$ *is defined, then there exists a constraint* d' *such that* $Var(\mathsf{d}') \subseteq Var(\mathsf{C_l}[\mathsf{H}'], \mathsf{c})$*,* $\mathsf{w_t}(\mathsf{C_l}[\mathsf{H}'], \mathsf{c}, \mathsf{d}') \leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{H}], \mathsf{c}, \mathsf{c}')$*, and* $\exists_{-Var(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{d}' = \exists_{-Var(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{c}'$.

Proof (Sketch). Observe that, for $i = 0$, the proof of 1 follows from the first part of Lemma 4.11. We prove here that, for each $i \geq 0$,

(a). if 1 holds for i, then 2 holds for i;
(b). if 1 and 2 hold for i, then 1 holds for $i + 1$.

The proof of the lemma then follows from straightforward inductive argument.

(a). If cl was not affected by the transformation step from $\mathsf{D_i}$ to $\mathsf{D_{i+1}}$, then the result is obvious by choosing $\mathsf{d}' = \exists_{-Var(\mathsf{C_l}[\mathsf{H}],\mathsf{c})}\mathsf{c}'$. Assume then that cl is affected when transforming $\mathsf{D_i}$ to $\mathsf{D_{i+1}}$. We have various cases, depending on the operation used to perform the transformation. Here we show only the proofs for the unfolding and the folding operations, the other cases are deferred to the Appendix.

Unfolding: Assume $\mathsf{cl}' \in \mathsf{D_{i+1}}$ was obtained from $\mathsf{D_i}$ by unfolding. In this case, the situation is the following:

—cl: $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})] \in \mathsf{D_i}$,
—u: $\mathsf{p}(\tilde{\mathsf{s}}) \leftarrow \mathsf{B} \in \mathsf{D_i}$,
—cl': $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})] \in \mathsf{D_{i+1}}$,

where cl and u are assumed to be renamed so that they do not share variables. Let $\mathsf{n} = \mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]], \mathsf{c}, \mathsf{c}')$. By the definition of transformation sequence, there exists a declaration $\mathsf{p}(\tilde{\mathsf{s}}) \leftarrow \mathsf{B_0} \in \mathsf{D_0}$. Moreover, by the hypothesis on the variables, $Var(\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})], \mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]) \cap Var(\mathsf{C_l}, \mathsf{c}) \subseteq Var(\tilde{\mathsf{r}})$, and then $Var(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]], \mathsf{c}) \cap Var(\tilde{\mathsf{s}}) = \emptyset$. Therefore, by Lemma 4.10, there exists a constraint $\mathsf{d_1}$ such that

$$\mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{s}}) \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]], \mathsf{c}, \mathsf{d_1}) \leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]], \mathsf{c}, \mathsf{c}') = \mathsf{n} \tag{9}$$

and

$$\exists_{-Var(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]],\mathsf{c})}\mathsf{d_1} = \exists_{-Var(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]],\mathsf{c})}\mathsf{c}'. \tag{10}$$

By the hypothesis on the variables and since u is renamed apart from cl, $Var(\mathsf{B}) \cap Var(\mathsf{C_l}, \mathsf{C}, \tilde{\mathsf{t}}, \mathsf{c}) = \emptyset$, and therefore $Var(\mathsf{B}) \cap Var(\mathsf{C_l}[\mathsf{C}[\ ] \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})], \mathsf{c}) \subseteq Var(\tilde{\mathsf{s}})$. Then, by Point 1, there exists a constraint d' such that

$$\begin{aligned} Var(\mathsf{d}') &\subseteq Var(\mathsf{C_l}[\mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]], \mathsf{c}) \\ \mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]], \mathsf{c}, \mathsf{d}') &\leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{s}}) \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]], \mathsf{c}, \mathsf{d_1}) \\ \exists_{-Var(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{s}})\ \parallel\ \mathsf{tell}(\tilde{\mathsf{t}}=\tilde{\mathsf{s}})]],\mathsf{c})}\mathsf{d}' &= \exists_{-Var(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{s}})\ \parallel\ \mathsf{tell}(\tilde{\mathsf{t}}=\tilde{\mathsf{s}})]],\mathsf{c})}\mathsf{d_1}. \end{aligned}$$

By (9), $w_t(C_l[C[B \parallel tell(\tilde{t} = \tilde{s})]], c, d') \leq n$. Furthermore, by hypothesis and construction,

$$Var(c', d') \cap Var(\tilde{r}) \subseteq Var(C_l[C[p(\tilde{t})]], c)$$

and, without loss of generality, we can assume that

$$Var(d_1) \cap Var(\tilde{r}) \subseteq Var(C_l[C[p(\tilde{t})]], c).$$

Then, by (10) and since $Var(C_l[C[p(\tilde{t})]], c) \subseteq Var(C_l[C[p(\tilde{s}) \parallel tell(\tilde{t} = \tilde{s})]], c)$, we have that $\exists_{-Var(C_l[q(\tilde{r})],c)} d' = \exists_{-Var(C_l[q(\tilde{r})],c)} c'$, and this completes the proof.

Folding: Let

—cl: $q(\tilde{r}) \leftarrow C[B]$ be the folded declaration ($\in D_i$),
—f: $p(\tilde{X}) \leftarrow B$ be the folding declaration ($\in D_0$),
—cl′: $q(\tilde{r}) \leftarrow C[p(\tilde{X})]$ be the result of the folding operation ($\in D_{i+1}$),

where, by hypothesis, $Var(cl) \cap Var(\tilde{X}) \subseteq Var(B)$, $Var(B) \cap Var(\tilde{r}, C) \subseteq Var(\tilde{X})$, $Var(C[B], C[p(\tilde{X})]) \cap Var(C_l, c) \subseteq Var(\tilde{r})$, $Var(c') \cap Var(\tilde{r}) \subseteq Var(C_l[C[B]], c)$, and there exists n such that $w_t(C_l[C[B]], c, c') = n$. Then,

$$Var(B) \cap Var(C_l[C[\ ]], c) \subseteq Var(B) \cap Var(\tilde{r}, C) \subseteq Var(\tilde{X}) \tag{11}$$

and

$$Var(c') \cap Var(\tilde{r}) \subseteq Var(C_l[C[B]], c) \cap Var(\tilde{r}) \subseteq Var(C_l[C[p(\tilde{X})]], c) \tag{12}$$

hold. Moreover, we can assume, without loss of generality, that $Var(c') \cap Var(\tilde{X}) \subseteq Var(C_l[C[B]], c)$.

Since $f \in D_0$, from (11) and point 2 of Lemma 4.11, it follows that there exists a constraint d' such that $w_t(C_l[C[p(\tilde{x})]], c, d') \leq w_t(C_l[C[B]], c, c')$ and

$$\exists_{-Var(C_l[C[p(\tilde{X})]],c)} d' = \exists_{-Var(C_l[C[p(\tilde{X})]],c)} c'. \tag{13}$$

We can assume, without loss of generality, that $Var(d') \subseteq Var(C_l[C[p(\tilde{X})]], c)$. Then, by using (12) and (13), we obtain that $\exists_{-Var(C_l[q(\tilde{r})],c)} d' = \exists_{-Var(C_l[q(\tilde{r})],c)} c'$, which concludes the proof of (a).

(b) Assume that the parts 1 and 2 of this lemma hold for $i \geq 0$. We prove that 1 holds for $i + 1 > 0$.
Let cl: $q(\tilde{r}) \leftarrow H \in D_{i+1}$, and let $\overline{cl}$: $q(\tilde{r}) \leftarrow \overline{H}$ be the corresponding declaration in D_i. Moreover, let $C_l[\ ]$ be a context, c a satisfiable constraint and let c' be a constraint, such that $Var(H) \cap Var(C_l, c) \subseteq Var(\tilde{r})$ and $w_t(C_l[q(\tilde{r})], c, c')$ is defined. Without loss of generality, we can assume that $Var(\overline{H}) \cap Var(C_l, c) \subseteq Var(\tilde{r})$. Then, since by inductive hypothesis, part 1 holds for i, there exists a constraint d_1 such that $Var(d_1) \subseteq Var(C_l[\overline{H}], c)$,

$$w_t(C_l[\overline{H}], c, d_1) \leq w_t(C_l[q(\tilde{r})], c, c') \text{ and } \exists_{-Var(C_l[q(\tilde{r})],c)} d_1 = \exists_{-Var(C_l[q(\tilde{r})],c)} c'. \tag{14}$$

Since by inductive hypothesis part 2 holds for i, there exists a constraint d', such that $Var(d') \subseteq Var(C_l[H], c)$, $w_t(C_l[H], c, d') \leq w_t(C_l[\overline{H}], c, d_1)$, and $\exists_{-Var(C_l[q(\tilde{r})],c)} d' = \exists_{-Var(C_l[q(\tilde{r})],c)} d_1$. By (14), $w_t(C_l[H], c, d') \leq w_t(C_l[q(\tilde{r})], c, c')$ and $\exists_{-Var(C_l[q(\tilde{r})],c)} d' = \exists_{-Var(C_l[q(\tilde{r})],c)} c'$, and then the thesis follows. □

We finally obtain our first main theorem.

THEOREM 4.13 (Total Correctness). *Let* $\mathsf{D_0}, \ldots, \mathsf{D_n}$ *be a transformation sequence. Then, for any agent* A, $\mathcal{O}(\mathsf{D_0.A}) = \mathcal{O}(\mathsf{D_n.A})$.

PROOF SKETCH. The proof proceeds by showing simultaneously, by induction on i, that for $i \in [0, n]$:

(1) for any agent A, $\mathcal{O}(\mathsf{D_0.A}) = \mathcal{O}(\mathsf{D_i.A})$;
(2) $\mathsf{D_i}$ is weight-complete.

Base Case. We just need to prove that $\mathsf{D_0}$ is weight-complete. Assume that there exists a derivation $\langle \mathsf{D_0.A, c_I}\rangle \rightarrow^* \langle \mathsf{D_0.B, c_F}\rangle$, where $\mathsf{c_I}$ is a satisfiable constraint and $\mathsf{m(B, c_F)} \in \{\mathsf{ss, dd, ff}\}$. Then there exists a derivation ξ: $\langle \mathsf{D_0.A, c_I}\rangle \rightarrow^* \langle \mathsf{D_0.B', c'_F}\rangle$ such that $\mathsf{m(B', c'_F)} = \mathsf{m(B, c_F)}$, whose weight is minimal and where $\exists_{-Var(\mathsf{A,c_I})}\mathsf{c'_F} = \exists_{-Var(\mathsf{A,c_I})}\mathsf{c_F}$. It follows from Definition 4.7 that ξ is a split derivation.

Induction Step. By the inductive hypothesis for any agent A, $\mathcal{O}(\mathsf{D_0.A}) = \mathcal{O}(\mathsf{D_{i-1}.A})$ and $\mathsf{D_{i-1}}$ is weight-complete. From Propositions 4.5 and 4.9, it follows that if $\mathsf{D_i}$ is weight-complete, then for any agent A, $\mathcal{O}(\mathsf{D_0.A}) = \mathcal{O}(\mathsf{D_i.A})$. So in order to prove parts 1 and 2, we only have to show that $\mathsf{D_i}$ is weight-complete.

Assume then that there exists a derivation $\langle \mathsf{D_0.A, c_I}\rangle \rightarrow^* \langle \mathsf{D_0.B, c_F}\rangle$ such that $\mathsf{c_I}$ is a satisfiable constraint and $\mathsf{m(B, c_F)} \in \{\mathsf{ss, dd, ff}\}$. From the inductive hypothesis, it follows that there exists a split derivation

$$\chi = \langle \mathsf{D_{i-1}.A, c_I}\rangle \rightarrow^* \langle \mathsf{D_{i-1}.A_m, c_m}\rangle \rightarrow \langle \mathsf{D_0.A_{m+1}, c_{m+1}}\rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F}\rangle$$

where

$$\exists_{-Var(\mathsf{A,c_I})}\mathsf{c''_F} = \exists_{-Var(\mathsf{A,c_I})}\mathsf{c_F} \text{ and } \mathsf{m(B'', c''_F)} = \mathsf{m(B, c_F)}. \qquad (15)$$

Let $\mathsf{d} \in \mathsf{D_{i-1}} \backslash \mathsf{D_i}$ be the modified clause in the transformation step from $\mathsf{D_{i-1}}$ to $\mathsf{D_i}$.

If, in the first m steps of χ, there is no procedure call that uses d, then clearly there exists a split derivation ξ in $\mathsf{D_i} \cup \mathsf{D_0}$,

$$\xi = \langle \mathsf{D_i.A, c_I}\rangle \rightarrow^* \langle \mathsf{D_i.A_m, c_m}\rangle \rightarrow \langle \mathsf{D_0.A_{m+1}, c_{m+1}}\rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F}\rangle,$$

which performs the same steps of χ, and then the thesis holds.

Otherwise, assume, without loss of generality, that **R4** is the rule used in the first step of derivation χ and that d is the clause employed in the first step of χ. We also assume that the declaration d is used only once in χ, since the extension to the general case is immediate.

We have to distinguish various cases according to what happens to the clause d when moving from $\mathsf{D_{i-1}}$ to $\mathsf{D_i}$. As before, we consider here the unfolding and the folding cases only, the others being deferred to the Appendix.

Unfolding: Assume that d is unfolded and let $\mathsf{d'}$ be the corresponding declaration in $\mathsf{D_i}$. The situation is the following:

—d: $\mathsf{q(\tilde{r}) \leftarrow C[p(\tilde{t})]} \in \mathsf{D_{i-1}}$,
—u: $\mathsf{p(\tilde{s}) \leftarrow H} \in \mathsf{D_{i-1}}$, and
—$\mathsf{d'}$: $\mathsf{q(\tilde{r}) \leftarrow C[H \parallel tell(\tilde{t} = \tilde{s})]} \in \mathsf{D_i}$,

where d and u are assumed to be renamed apart. By the definition of split derivation, χ has the form

$$\langle D_{i-1}.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_{i-1}.C_l[C[p(\tilde{t})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle \rightarrow^* \langle D_{i-1}.A_m, c_m\rangle \\ \rightarrow \langle D_0.A_{m+1}, c_{m+1}\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle.$$

Without loss of generality, we can assume that $Var(\chi) \cap Var(u) \neq \emptyset$ if and only if $p(\tilde{t})$ is evaluated in the first m steps of χ, in which case u is used for evaluating it. We have to distinguish two cases.
(1). There exists $k < m$ such that the k-th derivation step of χ is the procedure call $p(\tilde{t})$. In this case χ has the form

$$\langle D_{i-1}.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_{i-1}.C_l[C[p(\tilde{t})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle \rightarrow^* \langle D_{i-1}.C_k[p(\tilde{t})], c_k\rangle \\ \rightarrow \langle D_{i-1}.C_k[H \parallel tell(\tilde{t} = \tilde{s})], c_k\rangle \rightarrow^* \langle D_{i-1}.A_m, c_m\rangle \rightarrow \langle D_0.A_{m+1}, c_{m+1}\rangle \\ \rightarrow^* \langle D_0.B'', c''_F\rangle.$$

Then there exists a corresponding derivation in $D_i \cup D_0$,

$$\xi = \langle D_i.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_i.C_l[C[H \parallel tell(\tilde{t} = \tilde{s})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle \\ \rightarrow^* \langle D_i.C_k[H \parallel tell(\tilde{t} = \tilde{s})], c_k\rangle \rightarrow^* \langle D_i.A_m, c_m\rangle \rightarrow \langle D_0.A_{m+1}, c_{m+1}\rangle \\ \rightarrow^* \langle D_0.B'', c''_F\rangle,$$

which performs exactly the same steps of χ, except for a procedure call to $p(\tilde{t})$. In this case the proof follows by observing that, since by the inductive hypothesis, χ is a split derivation, the same holds for ξ.

(2). There is no procedure call to $p(\tilde{t})$ in the first m steps. Therefore, χ has the form

$$\langle D_{i-1}.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_{i-1}.C_l[C[p(\tilde{t})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle \rightarrow^* \langle D_{i-1}.C_m[p(\tilde{t})], c_m\rangle \\ \rightarrow \langle D_0.C_{m+1}[p(\tilde{t})], c_m\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle.$$

Then, by the definition of D_i, there exists a derivation

$$\xi_0 = \langle D_i.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_i.C_l[C[H \parallel tell(\tilde{t} = \tilde{s})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle \\ \rightarrow^* \langle D_i.C_m[H \parallel tell(\tilde{t} = \tilde{s})], c_m\rangle \rightarrow \langle D_0.C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m\rangle.$$

Observe that from the derivation $\langle D_0.C_{m+1}[p(\tilde{t})], c_m\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle$ and (15), it follows that

$$w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F) \text{ is defined, where } t = m(B, c_F). \tag{16}$$

The hypothesis on the variables implies that $Var(C_{m+1}[p(\tilde{t})], c_m) \cap Var(u) = \emptyset$. Then, by the definition of the transformation sequence and since $u \in D_{i-1}$, there exists a declaration $p(\tilde{s}) \leftarrow H_0 \in D_0$. By Lemma 4.10 and part 1 of Lemma 4.12, it follows that there exists a constraint d_F such that

$$w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, d_F) \leq w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F) \tag{17}$$

and

$$\exists_{-Var(C_{m+1}[p(\tilde{t})],c_m)} d_F = \exists_{-Var(C_{m+1}[p(\tilde{t})],c_m)} c''_F. \tag{18}$$

Therefore, by the definition of w_t, by (17) and since $w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F)$ is defined, there exists a derivation

$$\xi_1 = \langle D_0.C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m\rangle \rightarrow^* \langle D_0.B', c'_F\rangle,$$

where $\exists_{-Var(C_{m+1}[H \parallel tell(\tilde{t}=\tilde{s})],c_m)} c'_F = \exists_{-Var(C_{m+1}[H \parallel tell(\tilde{t}=\tilde{s})],c_m)} d_F$ and, by (16),

$$m(B', c'_F) = m(B, c_F). \tag{19}$$

By (18),

$$\exists_{-Var(C_{m+1},c_m)} c'_F = \exists_{-Var(C_{m+1},c_m)} c''_F \tag{20}$$

holds, and, by definition of weight, we obtain

$$w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, c'_F) = w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, d_F). \tag{21}$$

Moreover, we can assume without loss of generality that $Var(\xi_0) \cap Var(\xi_1) = Var(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m)$. Then, by the definition of procedure call.

$$Var(C_l[q(\tilde{v})], c_l) \cap (Var(c'_F) \cup Var(c''_F)) \subseteq Var(C_{m+1}, c_m), \tag{22}$$

and there exists a derivation

$$\begin{aligned} \xi = {} & \langle D_i.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_i.C_l[C[H \parallel tell(\tilde{t} = \tilde{s})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle \\ & \rightarrow^* \langle D_i.C_m[H \parallel tell(\tilde{t} = \tilde{s})], c_m\rangle \rightarrow \langle D_0.C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m\rangle \\ & \rightarrow^* \langle D_0.B', c'_F\rangle \end{aligned}$$

such that the first $m - 1$ derivation steps do not use rule **R2** and the m-th derivation step uses rule **R2**. We now have the following equalities:

$$\begin{aligned} & \exists_{-Var(C_l[q(\tilde{v})],c_l)} c'_F && = \text{by (22) and construction} \\ & \exists_{-Var(C_l[q(\tilde{v})],c_l)} (c_m \wedge \exists_{-Var(C_{m+1},c_m)} c'_F) && = \text{by (20)} \\ & \exists_{-Var(C_l[q(\tilde{v})],c_l)} (c_m \wedge \exists_{-Var(C_{m+1},c_m)} c''_F) && = \text{by (22) and construction} \\ & \exists_{-Var(C_l[q(\tilde{v})],c_l)} c''_F && = \text{by the first statement in (15)} \\ & \exists_{-Var(C_l[q(\tilde{v})],c_l)} c_F. \end{aligned}$$

By the definition of weight, $w_t(C_l[q(\tilde{v})], c_l, c'_F) = w_t(C_l[q(\tilde{v})], c_l, c''_F)$, by (21) and (17), $w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, c'_F) \leq w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F)$, and $w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F) < w_t(C_l[q(\tilde{v})], c_l, c''_F)$, since χ is a split derivation. Therefore, $w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, c'_F) < w_t(C_l[q(\tilde{v})], c_l, c'_F)$ and then, by definition, ξ is a split derivation in $D_i \cup D_0$. This, together with (19), implies the thesis.

Folding: Assume that d is folded and let

—d: $q(\tilde{r}) \leftarrow C[H]$ be the folded declaration ($\in D_{i-1}$),
—f: $p(\tilde{X}) \leftarrow H$ be the folding declaration ($\in D_0$),
—d′: $q(\tilde{r}) \leftarrow C[p(\tilde{X})]$ be the result of the folding operation ($\in D_i$),

where, by definition of folding, $Var(d) \cap Var(\tilde{X}) \subseteq Var(H)$ and $Var(H) \cap (Var(\tilde{r}) \cup Var(C)) \subseteq Var(\tilde{X})$. Since C[ ] is a guarding context, the agent H in C[H] appears in the scope of an ask guard. By definition of split derivation, χ has the form

$$\begin{aligned} & \langle D_{i-1}.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_{i-1}.C_l[C[H] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle \rightarrow^* \langle D_{i-1}.C_m[H], c_m\rangle \\ & \quad \rightarrow \langle D_0.C_{m+1}[H], c_m\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle, \end{aligned}$$

where $C_m[\ ]$ is a guarding context. Without loss of generality, we can assume that $Var(\chi) \cap Var(\tilde{x}) \subseteq Var(H)$. Then, from the definition of D_i it follows that there exists a derivation

$$\begin{aligned} \xi_0 = {} & \langle D_i.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_i.C_l[C[p(\tilde{X})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle \\ & \rightarrow^* \langle D_i.C_m[p(\tilde{X})], c_m\rangle \rightarrow \langle D_0.C_{m+1}[p(\tilde{X})], c_m\rangle, \end{aligned}$$

which performs exactly the first m steps as χ. Since $\langle \mathsf{D_0.C_{m+1}[H], c_m} \rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F} \rangle$, the definition of weight implies that $\mathsf{w_t(C_{m+1}[H], c_m, c''_F)}$ is defined, where $\mathsf{t = m(B'', c''_F)}$. Then, by (15), we have that

$$\mathsf{t = m(B, c_F)}. \tag{23}$$

The definitions of derivation and folding imply that $Var(\mathsf{H}) \cap Var(\mathsf{C_{m+1}, c_m}) \subseteq Var(\mathsf{H}) \cap (Var(\mathsf{C}, \tilde{\mathsf{r}})) \subseteq Var(\tilde{\mathsf{X}})$ holds. Moreover, from the assumptions on the variables, we obtain that $Var(\mathsf{c''_F}) \cap Var(\tilde{\mathsf{t}}) \subseteq Var(\mathsf{H})$. Thus, from part 2 of Lemma 4.11, it follows that there exists a constraint $\mathsf{d'}$ such that

$$\mathsf{w_t(C_{m+1}[p(\tilde{x})], c_m, d') \leq w_t(C_{m+1}[H], c_m, c''_F)} \tag{24}$$

and

$$\exists_{-Var(\mathsf{C_{m+1}[p(\tilde{x})], c_m})} \mathsf{d'} = \exists_{-Var(\mathsf{C_{m+1}[p(\tilde{x})], c_m})} \mathsf{c''_F}. \tag{25}$$

From the definition of weight and the fact that $\mathsf{w_t(C_{m+1}[H], c_m, c''_F)}$ is defined, it follows that there exists a derivation $\xi_1 = \langle \mathsf{D_0.C_{m+1}[p(\tilde{x})], c_m} \rangle \rightarrow^* \langle \mathsf{D_0.B', c'_F} \rangle$, where $\mathsf{m(B', c'_F) = t}$ and $\exists_{-Var(\mathsf{C_{m+1}[p(\tilde{x})], c_m})} \mathsf{c'_F} = \exists_{-Var(\mathsf{C_{m+1}[p(\tilde{x})], c_m})} \mathsf{d'}$. Then, by the definition of weight, $\mathsf{w_t(C_{m+1}[p(\tilde{x})], c_m, c'_F) = w_t(C_{m+1}[p(\tilde{x})], c_m, d')}$, and therefore, by (24 and 25),

$$\exists_{-Var(\mathsf{C_{m+1}[p(\tilde{x})], c_m})} \mathsf{c'_F} = \exists_{-Var(\mathsf{C_{m+1}[p(\tilde{x})], c_m})} \mathsf{c''_F} \tag{26}$$

and

$$\mathsf{w_t(C_{m+1}[p(\tilde{X})], c_m, c'_F) \leq w_t(C_{m+1}[H], c_m, c''_F)} \tag{27}$$

hold. Moreover, from (23) we obtain

$$\mathsf{m(B', c'_F) = m(B, c_F)}. \tag{28}$$

Without loss of generality, we can now assume that

$$Var(\xi_0) \cap Var(\xi_1) = Var(\mathsf{C_{m+1}[p(\tilde{X})], c_m}).$$

Then, by (26), (27), and (15) it follows that

$$\begin{aligned} \exists_{-Var(\mathsf{C_l[q(\tilde{v})], c_l})} \mathsf{c'_F} &= \exists_{-Var(C_l[q(\tilde{v})], c_l)} (c_m \wedge \exists_{-Var(C_{m+1}[p(\tilde{x})], c_m)} c'_F) \\ &= \exists_{-Var(\mathsf{C_l[q(\tilde{v})], c_l})} (\mathsf{c_m} \wedge \exists_{-Var(\mathsf{C_{m+1}[p(\tilde{x})], c_m})} \mathsf{c''_F}) \\ &= \exists_{-Var(\mathsf{C_l[q(\tilde{v})], c_l})} \mathsf{c''_F} = \exists_{-Var(\mathsf{C_l[q(\tilde{v})], c_l})} \mathsf{c_F}. \end{aligned} \tag{29}$$

From the definition of weight $\mathsf{w_t(C_l[q(\tilde{v})], c_l, c'_F) = w_t(C_l[q(\tilde{v})], c_l, c''_F)}$ and since χ is a split derivation, we obtain $\mathsf{w_t(C_l[q(\tilde{v})], c_l, c''_F) > w_t(C_{m+1}[H], c_m, c''_F)}$. Then, from (29), it follows that

$$\mathsf{w_t(C_l[q(\tilde{v})], c_l, c'_F) > w_t(C_{m+1}[p(\tilde{X})], c_m, c'_F)}, \tag{30}$$

and therefore by construction

$$\begin{aligned} \xi \ = \ & \langle \mathsf{D_i.C_l[q(\tilde{v})], c_l} \rangle \rightarrow \langle \mathsf{D_i.C_l[C[p(\tilde{X})] \parallel tell(\tilde{v} = \tilde{r})], c_l} \rangle \rightarrow^* \langle \mathsf{D_i.C_m[p(\tilde{X})], c_m} \rangle \\ & \rightarrow \langle \mathsf{D_0.C_{m+1}[p(\tilde{X})], c_m} \rangle \rightarrow^* \langle \mathsf{D_0.B', c'_F} \rangle \end{aligned}$$

is a derivation in $\mathsf{D_i \cup D_0}$ such that: (a) rule **R2** is not used in the first $m - 1$ steps; (b) rule **R2** is used in the m-th step. The thesis then follows from (29), (28), and (30), thus concluding the proof. □

It is important to see that, given the definition of observables we adopt (Definition 4.1), the initial program D_0 and the final one, D_n, have exactly the same successful derivations, the same deadlocked derivations, and the same failed derivations. The first feature (regarding successful derivations) is to some extent the one we expect and require from a transformation, because it corresponds to the intuition that D_n "produces the same results" as D_0. Nevertheless, the second feature (preservation of deadlock derivations) has an important role also. First, it ensures that the transformation does not introduce deadlock points, which is of crucial importance when we are using the transformation for optimizing a program. Second, as exemplified in Section 6, this feature allows us to use the transformation as a tool for proving deadlock-freeness (i.e., absence of deadlock). In fact, if, after the transformation, we can prove or see that process $D_n.A$ never deadlocks, then we are also sure that $D_0.A$ does not deadlock either.

5. CORRECTNESS FOR NONTERMINATING COMPUTATIONS

The correctness results obtained so far consider terminating (successful and deadlocked) and failed computations only. This is satisfactory for many applications of concurrent constraint programming that have a "transformational" behavior, i.e., are supposed to produce a (finite) output for a given (finite) input. In this respect, it is worth noting that the two main semantic models of CCP consider essentially the same notion of observables we used. In fact, the model based on linear sequences defined in de Boer and Palamidessi [1991] characterizes (in a fully abstract way) the results of terminating computations, together with a termination mode indicating success, deadlock, or failure.[6] Such a model was proved [de Boer and Palamidessi 1992] to be isomorphic to the semantics based on (bounded) closure operators introduced in Saraswat et al. [1991], provided that the termination mode and the consistency checks are eliminated.

So our correctness results are adequate, in the sense that they ensure that the standard semantics of CCP is preserved. On the other hand, as is the case for any other concurrent programming paradigm, CCP programs may have a "reactive" nature: rather than producing a final result, they produce a (possibly nonterminating) sequence of intermediate results in response to some external stimuli. For these programs the notion of observables employed in Theorem 4.13 and the related results are not adequate, since they exclude nonterminating computations.

When considering nonterminating computations we is interested in observing (possibly in terms of traces) the intermediate results, that is, the constraints produced by nonmaximal derivations also, rather than the final limit of the computation (note, however, that in CCP such a notion of limit makes sense, as the

[6]There are irrelevant differences between the observables considered in de Boer and Palamidessi [1991] and the ones we use, due to the treatment of failure and to the existential quantification on local variables.

store grows monotonically). Therefore, in the remainder of this section we first discuss the correctness of our system w.r.t. this new class of observables. We then show a modification of our transformation system and present a stronger correctness result, which guarantees that (traces of) intermediate results are preserved.

5.1 Partial Preservation of Intermediate Results

It is easy to see that the system we have proposed does not preserve the intermediate results of computations. More precisely, let us define these observables as follows:

$$\mathcal{O}_i(\mathsf{D.A}) = \big\{ \big\langle \mathsf{c}, \exists_{-Var(\mathsf{A,c})}\mathsf{d}, \mathsf{pp} \big\rangle \mid \mathsf{c} \text{ and } \mathsf{d} \text{ are satisfiable, and there exists a derivation } \langle \mathsf{D.A, c}\rangle \rightarrow^* \langle \mathsf{D.B, d}\rangle \big\}$$

(here the symbol pp indicates that we consider results obtained from "partial," that is, possibly not maximal, derivations). Now it is easy to see that the operations of ask and tell simplification are neither partially nor totally correct w.r.t. the semantics $\mathcal{O}_i(\mathsf{D.A})$. In fact, the ask simplification allows us to transform the agent

$$\mathsf{A: tell(c)} \parallel \mathsf{ask(true)} \rightarrow \mathsf{tell(d)}$$

into the agent

$$\mathsf{A': tell(c)} \parallel \mathsf{ask(c)} \rightarrow \mathsf{tell(d)}.$$

While the agent A, when evaluated in the empty store, produces the intermediate result d, this is not the case for the agent A′ (we assume that $\mathsf{c} \wedge \mathsf{d} \neq \mathsf{d}$). Analogously, assuming that $\mathcal{D} \models \mathsf{d} \rightarrow \mathsf{c}$ and $\mathcal{D} \models \mathsf{d} \rightarrow \mathsf{c'}$, the tell simplification allows us to transform

$$\mathsf{B: tell(c)} \parallel \mathsf{tell(d)}$$

into the agent

$$\mathsf{B': tell(c')} \parallel \mathsf{tell(d)}$$

and the agents B and B′ have different intermediate results. Other operations that are not correct w.r.t. the above semantics are the distribution and the tell elimination and introduction.

Nevertheless, the system we have defined already preserves a form of intermediate results. This is shown by the following theorem.

THEOREM 5.1 (Total Correctness 2). *Let* $\mathsf{D_0}, \ldots, \mathsf{D_n}$ *be a transformation sequence, and* A *be an agent.*

—*If there exists a derivation* $\langle \mathsf{D_0.A, c}\rangle \rightarrow^* \langle \mathsf{D_0.B, d}\rangle$*, then there exists a derivation* $\langle \mathsf{D_n.A, c}\rangle \rightarrow^* \langle \mathsf{D_n.B', d'}\rangle$ *such that* $\mathcal{D} \models \exists_{-\mathsf{Var(A,c)}}\mathsf{d'} \rightarrow \exists_{-\mathsf{Var(A,c)}}\mathsf{d}$.

—*Conversely, if there exists a derivation* $\langle \mathsf{D_n.A, c}\rangle \rightarrow^* \langle \mathsf{D_n.B, d}\rangle$*, then there exists a derivation* $\langle \mathsf{D_0.A, c}\rangle \rightarrow^* \langle \mathsf{D_0.B', d'}\rangle$ *with* $\mathcal{D} \models \exists_{-\mathsf{Var(A,c)}}\mathsf{d'} \rightarrow \exists_{-\mathsf{Var(A,c)}}\mathsf{d}$.

PROOF. The proof of this result is essentially the same as the one for total correctness Theorem 4.13 provides, that in such a proof, as well as in the proofs of the related preliminary results, we perform the following changes:

(1) Rather than considering terminating derivations, we consider any (possibly nonmaximal) finite derivations.
(2) Whenever we write in a proof that, given a derivation ξ, a derivation ξ' is constructed that performs the same steps ξ does, possibly in a different order, we now write that a derivation ξ'' is constructed that performs the same step of ξ (possibly in a different order), plus other additional steps. Since the store grows monotonically in CCP derivations, then clearly if a constraint c is the result of the derivation ξ, then a constraint c'' is the result of ξ'' such that $\mathcal{D} \models \mathsf{c}'' \rightarrow \mathsf{c}$ holds. For example, for case 2 in the proof of Proposition 4.5, when considering a (nonmaximal) derivation ξ that uses the declaration $\mathsf{H} \leftarrow \mathsf{C}[\mathsf{tell}(\tilde{\mathsf{s}} = \tilde{\mathsf{t}})] \parallel \mathsf{B}]$, we can always construct a derivation ξ'' that performs all the steps of ξ (plus, possibly, others) and such that the $\mathsf{tell}(\tilde{\mathsf{s}} = \tilde{\mathsf{t}})$ agent is evaluated before B. Differently from the previous proof, we are now not sure that the result of ξ is the same as that of ξ'', since ξ is nonmaximal (thus ξ could also avoid the evaluation of $\mathsf{tell}(\tilde{\mathsf{s}} = \tilde{\mathsf{t}})$). However, we are ensured that the result of ξ'' is stronger (i.e., implies) than one of ξ. $\square$

This result ensures that the original and the transformed program, have the same intermediate results up to logical implication: If the evaluation of an agent in the original program produces a constraint d, then a constraint stronger than d is produced in the transformed program, and vice versa. The vice versa is important, as it ensures that the transformed program will never produce something that could not be produced by the original program, up to implication. Clearly, this result is relevant in presence of nonterminating computations (which were not covered by Theorem 4.13).

In order to maintain a consistent notation throughout the article, the above result can be reformulated in terms of the following class of observables:

$$\mathcal{O}_{\mathsf{ic}}(\mathsf{D.A}) = \Big\{\Big\langle \mathsf{c}, \exists_{-Var(\mathsf{A},\mathsf{c})}\mathsf{d}, \mathsf{pp}\Big\rangle \mid \begin{array}{l}\mathsf{c} \text{ is satisfiable, and there exists a derivation} \\ \langle \mathsf{D.A}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D.B}, \mathsf{d}'\rangle \\ \text{and } \mathcal{D} \models \exists_{-Var(\mathsf{A},\mathsf{c})}\mathsf{d}' \rightarrow \exists_{-Var(\mathsf{A},\mathsf{c})}\mathsf{d}\Big\}\end{array}$$

where the subscript ic stands for *implication closure* (of intermediate results). We then have following corollary, whose proof is immediate.

COROLLARY 5.2. *Let* $\mathsf{D}_0, \ldots, \mathsf{D}_\mathsf{n}$ *be a transformation sequence. Then, for any agent* A, $\mathcal{O}_{\mathsf{ic}}(\mathsf{D}_0.\mathsf{A}) = \mathcal{O}_{\mathsf{ic}}(\mathsf{D}_\mathsf{n}.\mathsf{A})$.

This result guarantees a degree of correctness that should be sufficient for many reactive programs employing nonterminating computations. In fact, when transforming a program, we should probably not expect to be able to preserve exactly each intermediate result that the original program produced.

Nevertheless, it is of interest to check if it is possible to modify the system in order to obtain stronger correctness results. We do this in the following section.

5.2 Full Preservation of Intermediate Results

In this section we introduce a few restrictions on our transformation system, and prove that they guarantee the preservation of the whole sequence of intermediate results of a program.

As previously mentioned, the only operations not preserving the intermediate results are the ask and tell simplification, the distribution and the tell elimination, and introduction. As it may appear from the example above, the problem in using ask and tell simplification lies in the fact that we can modify the arguments of ask and tell agents by taking into account (via the "produced constraint") the constraints introduced by tell actions appearing in the parallel context also (see Definitions 3.4 and 3.6). This can clearly affect the intermediate results of the computations, since no order is imposed on the evaluation of parallel agents. This reasoning applies to the distribution operation as well.

We then have to modify the ask and tell simplification and the distribution by considering a weaker notion of "produced constraint," which includes only those constraints that have *certainly* been produced before reaching the ask or tell agent we are simplifying. Such a notion is defined as follows.

Definition 5.3. Given a context $\mathsf{C}[\,]$ the *weakest produced constraint* $\mathsf{wpc}(\mathsf{C}[\,])$ of $\mathsf{C}[\,]$ is inductively defined as follows:

$$\begin{aligned}\mathsf{wpc}([\,]) &= \mathsf{true}\\ \mathsf{wpc}(\mathsf{C}'[\,] \parallel \mathsf{B}) &= \mathsf{wpc}(\mathsf{C}'[\,])\\ \mathsf{wpc}\left(\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c_i}) \rightarrow \mathsf{A_i}\right) &= \mathsf{c_j} \wedge \mathsf{wpc}(\mathsf{C}'[\,])\text{where } \mathsf{j} \in [1, \mathsf{n}] \text{ and } \mathsf{A_j} = \mathsf{C}'[\,].\end{aligned}$$

For example, the weakest produced constraint of $[\,] \parallel \mathsf{tell(c)}$ is true, while the weakest produced constraint of $\mathsf{tell(c)} \parallel \mathsf{ask(d)} \rightarrow (\mathsf{ask(e)} \rightarrow [\,])$ is $\mathsf{d} \wedge \mathsf{e}$. We can then define the weak equivalence of two constraints within a given context $\mathsf{C}[\,]$ as follows:

Definition 5.4. Let c, c' be constraints, $\mathsf{C}[\,]$ be a context, and $\tilde{\mathsf{Z}}$ be a set of variables. We say that c is *weakly equivalent to* c' *within* $\mathsf{C}[\,]$ and *w.r.t. the variables in* $\tilde{\mathsf{Z}}$ iff $\mathcal{D} \models \exists_{-\tilde{\mathsf{Z}}}\,(\mathsf{wpc}(\mathsf{C}[\,]) \wedge \mathsf{c}) \leftrightarrow \exists_{-\tilde{\mathsf{Z}}}\,(\mathsf{wpc}(\mathsf{C}[\,]) \wedge \mathsf{c}')$.

Using this definition we can modify the operations of ask and tell simplification and distribution by simply replacing the context equivalence used in Definition 3.6 with the above notion of weak context equivalence. For the sake of clarity, we state the resulting definitions below.

Definition 5.5 (*Restricted Ask and Tell Simplification*). Let D be a set of declarations.

(1) Let $\mathsf{d}\colon \mathsf{H} \leftarrow \mathsf{C}[\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c_i}) \rightarrow \mathsf{A_i}]$ be a declaration of D. Suppose that $\mathsf{c}'_1, \ldots, \mathsf{c}'_n$ are constraints such that, for $j \in [1, n]$, c'_j is weakly equivalent to $\mathsf{c_j}$ within $\mathsf{C}[\,]$ and w.r.t. the variables in $Var(\mathsf{C}, \mathsf{H}, \mathsf{A_j})$. We can then replace d with d': $\mathsf{H} \leftarrow \mathsf{C}[\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c}'_i) \rightarrow \mathsf{A_i}]$ in D. We call this a *restricted ask simplification* operation.

(2) Let d: $\mathsf{H} \leftarrow \mathsf{C}[\mathsf{tell}(\mathsf{c})]$ be a declaration of D. Suppose that the constraint c' is weakly equivalent to c within $\mathsf{C}[\ ]$ and w.r.t. the variables in $\mathit{Var}(\mathsf{C}, \mathsf{H})$. We can then replace d with d': $\mathsf{H} \leftarrow \mathsf{C}[\mathsf{tell}(\mathsf{c}')]$ in D. We call this a *restricted tell simplification* operation.

Definition 5.6 (*Restricted Distribution*). Let D be a set of declarations and let

$$\mathsf{d}\colon\ \mathsf{H} \leftarrow \mathsf{C}\left[\mathsf{A} \parallel \sum_{i=1}^{n} \mathsf{ask}(\mathsf{c_i}) \rightarrow \mathsf{B_i}\right]$$

be a declaration in D. Also let $\mathsf{e} = \mathsf{wpc}(\mathsf{C}[\ ])$. The *restricted distribution* of A in d yields the definition

$$\mathsf{d}'\colon\ \mathsf{H} \leftarrow \mathsf{C}\left[\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c_i}) \rightarrow (\mathsf{A} \parallel \mathsf{B_i})\right],$$

provided that for every constraint c such that $\mathit{Var}(\mathsf{c}) \cap \mathit{Var}(\mathsf{d}) \subseteq \mathit{Var}(\mathsf{H}, \mathsf{C})$, if $\langle \mathsf{D.A}, \mathsf{c} \wedge \mathsf{e}\rangle$ is productive, then both the following conditions hold:

(a) there exists at least one $i \in [1, n]$ such that $\mathcal{D} \models (\mathsf{c} \wedge \mathsf{e}) \rightarrow \mathsf{c_i}$,
(b) for each $i \in [1, n]$, either $\mathcal{D} \models (\mathsf{c} \wedge \mathsf{e}) \rightarrow \mathsf{c_i}$ or $\mathcal{D} \models (\mathsf{c} \wedge \mathsf{e}) \rightarrow \neg\mathsf{c_i}$.

Remark 3.12 is also sufficient for guaranteeing that the restricted distribution operation is applicable. Thus, we have the following.

Remark 5.7. Referring to Definition 5.6. If A requires a variables that does not occur in $\mathsf{H}, \mathsf{C}[\ ]$, then the restricted distribution operation is applicable.

The tell elimination and the tell introduction operations do not preserve the intermediate results of computations either. This is not due to the presence of the *produced constraint*, but rather to the very nature of the operation that can eliminate or introduce constraints which, via the local variables, can also (temporarily) affect the values of global variables. For example, the declaration

$$\mathsf{d}\colon \mathsf{p}(\mathsf{Y}) \leftarrow \mathsf{tell}(\mathsf{Z} = \mathsf{a}) \parallel \mathsf{tell}(\mathsf{Y} = \mathsf{f}(\mathsf{Z}))$$

can be transformed via a *tell elimination* into

$$\mathsf{d}'\colon \mathsf{p}(\mathsf{Y}) \leftarrow \mathsf{tell}(\mathsf{Y} = \mathsf{f}(\mathsf{a})).$$

The evaluation of $\mathsf{p}(\mathsf{Y})$ in the empty store and using d produces the (intermediate) result $\mathsf{Y} = \mathsf{f}(\mathsf{Z})$, while this is not the case if we use the declaration d'. We can solve this problem by simply requiring that if we eliminate a tell by applying the resulting substitution to the parallel context B, then B does not contain any variable that appears in the head or in the outer context. Thus, we have the following:

Definition 5.8 (*Restricted Tell Elimination and Tell Introduction*). The declaration

$$\mathsf{d}\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{tell}(\tilde{\mathsf{s}} = \tilde{\mathsf{t}}) \parallel \mathsf{B}]$$

can be transformed via a *restricted tell elimination* into

$$d'\colon H \leftarrow C[B\sigma]$$

where σ is a relevant, most general, unifier of $\tilde{s}$ and $\tilde{t}$, provided that the variables the domain of σ do not occur in either $C[\,]$ or H, and that $Var(B) \cap Var(H, C) = \emptyset$. Again, this operation is applicable either when the computational domain admits a most general unifier, or when $\tilde{s}$ and $\tilde{t}$ are a sequence of distinct variables, in which case σ is simply a renaming. On the other hand, the declaration

$$d\colon H \leftarrow C[B\sigma]$$

can be transformed via a *restricted tell introduction* into

$$d'\colon H \leftarrow C[tell(\tilde{X} = \tilde{X}\sigma) \parallel B],$$

provided that σ is a substitution such that $\tilde{X} = Dom(\sigma)$ and $Dom(\sigma) \cap (Var(C[\,], H) \cup Ran(\sigma)) = \emptyset$, and that $Var(B) \cap Var(H, C) = \emptyset$.

At this point it is worth recalling that the tell elimination is often used for making variable bindings explicit after an unfolding operation: In fact, we start from a definition of the form $d\colon H \leftarrow C[p(\tilde{t})]$ and by unfolding $p(\tilde{t})$, we end with $d'\colon H \leftarrow C[B \parallel tell(\tilde{t} = \tilde{s})]$ (provided that p is defined by $u:\ p(\tilde{s}) \leftarrow B$). We then want to eliminate $tell(\tilde{t} = \tilde{s})$ from d' in order to perform "parameter-passing." Since d and u are always renamed apart—clearly, the additional condition of the restricted tell elimination ($Var(B) \cap Var(H, C) = \emptyset$) is always satisfied here. So, in general, this operation is applicable every time that $\tilde{t}$ is an instance of $\tilde{s}$.

We can finally define the restricted transformation system, as follows:

Definition 5.9. A *restricted transformation sequence* is a sequence of programs $D_0, \ldots, D_n$ in which D_0 is a initial program and each D_{i+1} is obtained from D_i via one of the following operations: unfolding, backward instantiation, restricted tell elimination, restricted tell introduction, restricted ask and tell simplification, branch elimination, conservative ask elimination, restricted distribution, and folding.

Clearly, the restricted transformation operations are applicable in fewer situations than their nonrestricted counterparts, yet they are useful in many cases. Example 6.1 shows a case of an unfold-fold transformation sequence using only restricted operations, and the other examples contain several occurrences of them. We now prove that the restricted system is correct w.r.t. the trace semantics of CCP. Here and in the following we denote by $c_1; c_2; \ldots; c_n$ a sequence of constraints, also called trace constraints.

Definition 5.10 (*Traces*). Let D.A be a CCP process. We define $\mathcal{O}_t(D.A) =$

$$\begin{array}{ll}\{\langle c_1; c_2; \ldots; c_n, ss\rangle \mid & \text{there exists a derivation} \\ & \langle D.A, d_1\rangle \rightarrow \langle D.A_2, d_2\rangle \rightarrow \cdots \rightarrow \langle D.Stop, d_n\rangle \\ & d_i \text{ is satisfiable for each } i \in [1, n], \\ & c_1 = d_1 \text{ and } c_j = \exists_{-Var(A,c_1)} d_j \text{ for each } j \in [2, n]\}\end{array}$$

$$
\begin{array}{ll}
\cup & \\
\{\langle \mathsf{c_1;c_2;\ldots;c_n,dd}\rangle & |\ \text{there exists a derivation}\\
 & \langle \mathsf{D.A,d_1}\rangle \rightarrow \langle \mathsf{D.A_2,d_2}\rangle \rightarrow \cdots \rightarrow \langle \mathsf{D.A_n,d_n}\rangle \not\rightarrow\\
 & \mathsf{A_n} \neq \mathsf{Stop},\ \mathsf{d_i} \text{ is satisfiable for each } \mathsf{i} \in [1,\mathsf{n}],\\
 & \mathsf{c_1 = d_1} \text{ and } \mathsf{c_j} = \exists_{-Var(\mathsf{A,c_1})}\mathsf{d_j} \text{ for each } \mathsf{j} \in [2,\mathsf{n}]\}\\
\cup & \\
\{\langle \mathsf{c_1;c_2;\ldots;c_n,pp}\rangle & |\ \text{there exists a derivation}\\
 & \langle \mathsf{D.A,d_1}\rangle \rightarrow \langle \mathsf{D.A_2,d_2}\rangle \rightarrow \cdots \rightarrow \langle \mathsf{D.A_n,d_n}\rangle\\
 & \mathsf{d_i} \text{ is satisfiable for each } \mathsf{i} \in [1,\mathsf{n}],\\
 & \mathsf{c_1 = d_1} \text{ and } \mathsf{c_j} = \exists_{-Var(\mathsf{A,c_1})}\mathsf{d_j} \text{ for each } \mathsf{j} \in [2,\mathsf{n}]\}\\
\cup & \\
\{\langle \mathsf{c_1;c_2;\ldots;c_n,ff}\rangle & |\ \text{there exists a derivation}\\
 & \langle \mathsf{D.A,d_1}\rangle \rightarrow \langle \mathsf{D.A_2,d_2}\rangle \rightarrow \cdots \rightarrow \langle \mathsf{D.A_n,d_n}\rangle \not\rightarrow\\
 & \mathsf{d_i} \text{ is satisfiable for each } \mathsf{i} \in [1,\mathsf{n}-1],\ \mathsf{d_n} = \mathsf{false}\\
 & \mathsf{c_1 = d_1} \text{ and } \mathsf{c_j} = \exists_{-Var(\mathsf{A,c_1})}\mathsf{d_j} \text{ for each } \mathsf{j} \in [2,\mathsf{n}].\}
\end{array}
$$

Thus what we observe are the finite traces consisting of the constraints produced by any (possibly nonterminating) derivations. As before, we abstract from the values for the local variables in the results, and we make distinctions among the successful traces (termination mode ss), the deadlocked ones (dd), the partial (i.e., possibly nonmaximal) traces (pp), and the failed ones (ff). Note that, due to the monotonic computational model of CCP, which does not allow us to retract information from the global store, the traces we observe are monotonically increasing. That is, given a trace $\mathsf{c_1;c_2;\ldots;c_n}$ appearing in the observables, we have that $\mathcal{D} \models \mathsf{c_i} \rightarrow \mathsf{c_j}$ for each $i, j \in [1, n]$ such that $i \geq j$. Before giving the correctness result, we need one last definition.

Definition 5.11. We say that a trace $\mathsf{c_1;c_2;\ldots;c_n}$ is *simulated by* a trace $\mathsf{d_1;d_2;}$ ldots$\mathsf{;d_m}$, notation $\mathsf{c_1;c_2;\ldots;c_n} \preceq \mathsf{d_1;d_2;\ldots;d_m}$, iff there exists $\{j_1, \ldots j_n\} \subseteq \{1, 2, \ldots, m\}$ such that

(1) $\mathsf{c_i} = \mathsf{d_{j_i}}$ for each $i \in [1, n]$;

(2) $\mathsf{j_1} = 1, \mathsf{j_n} = \mathsf{m}$, and $j_i \leq j_k$ iff $i < k$.

So a trace s is simulated by a trace s' iff they have the same first and last elements, and all components appearing in s appear, in the same order, in s'.

We can now state our strongest correctness result. Its proof, contained in the Appendix, follows the guidelines for the one for Theorem 4.13. In fact, the definitions of mode, weight, split derivation and weight-complete program can readily be extended to consider traces and weakest produced constraints, rather than input/output pairs and produced constraints. Then it is easy to extend all the technical lemmas needed for Theorem 4.13 in order to obtain the preliminary results needed in the proof of the following:

THEOREM 5.12 (Strong Total Correctness). *Let* $\mathsf{D_0, \ldots, D_n}$ *be a restricted transformation sequence, and* A *be an agent.*

—*If* $\langle \mathsf{s, x}\rangle \in \mathcal{O}_\mathsf{t}(\mathsf{D_0.A})$ (*with* $\mathsf{x} \in \{\mathsf{ss, dd, pp, ff}\}$), *then there exists* $\langle \mathsf{s', x}\rangle \in \mathcal{O}_\mathsf{t}(\mathsf{D_n.A})$ *such that* $\mathsf{s} \preceq \mathsf{s'}$.

—*Conversely, if* $\langle \mathsf{s}, \mathsf{x}\rangle \in \mathcal{O}_t(\mathsf{D}_n.\mathsf{A})$*, then there exists* $\langle \mathsf{s}', \mathsf{x}\rangle \in \mathcal{O}_t(\mathsf{D}_0.\mathsf{A})$ *such that* $\mathsf{s} \preceq \mathsf{s}'$.

Since it results from the definition of $\preceq$, we do not have the equality of traces exactly, since in some traces we might introduce some intermediate steps. However, note that these additional steps do not introduce new values, rather they can be seen as a different "approximation" to obtain a given constraint, since we consider here monotonically increasing traces. This can best be explained by means of an example. Consider the following one-line program D_0: $\mathsf{p(Y)} \leftarrow \mathsf{tell(X = f(a, W))} \parallel \mathsf{tell(X = f(Z, b))} \parallel \mathsf{tell(X = Y)}$. Its trace semantics $\mathcal{O}_t(\mathsf{D}_0.\mathsf{p(Y)})$ contains $\langle \mathsf{t}, \ \mathsf{ss}\rangle$, where t is the trace (true; true; true; $\mathsf{Y = f(a, b)}$). If we apply a restricted tell evaluation to $\mathsf{tell(X = Y)}$, we obtain the program D_1: $\mathsf{p(Y)} \leftarrow \mathsf{tell(Y = f(a, W))} \parallel \mathsf{tell(Y = f(Z, b))}$. Now, $\mathcal{O}_t(\mathsf{D}_1.\mathsf{p(Y)})$ does not contain t: we cannot obtain $\mathsf{Y = f(a, b)}$ from true in one step. On the other hand, $\mathcal{O}_t(\mathsf{D}_1.\mathsf{p(Y)})$ contains $\langle(\mathsf{true};\ \exists_{\mathsf{W}}\, \mathsf{Y = f(a, W)};\ \mathsf{Y = f(a, b)}),\ \mathsf{ss}\rangle$ and $\langle(\mathsf{true};\ \exists_{\mathsf{Z}}\, \mathsf{Y = f(Z, b)};\ \mathsf{Y = f(a, b)}),\ \mathsf{ss}\rangle$, and both the traces appearing in these pairs simulate t. Note also that the intermediate result semantics is now preserved. In fact, the following is an immediate consequence of Theorem 5.12.

COROLLARY 5.13. *Let* $\mathsf{D}_0, \ldots, \mathsf{D}_n$ *be a restricted transformation sequence and* A *be an agent. Then,* $\mathcal{O}_i(\mathsf{D}_0.\mathsf{A}) = \mathcal{O}_i(\mathsf{D}_n.\mathsf{A})$.

5.3 Preservation of Infinite Traces

It is worth noting that Theorem 5.12 can be extended to consider infinite traces also, as we show below.

In the following, we indicate by $|\mathsf{s}_i|$ the length of a trace s_i and we say that a configuration $\langle \mathsf{D.A}, \mathsf{c}_1\rangle$ produces the trace $\mathsf{c}_1; \mathsf{c}_2; \ldots;\ \mathsf{c}_n$ iff there exists a derivation $\langle \mathsf{D.A}, \mathsf{d}_1\rangle \rightarrow \langle \mathsf{D.A}_2, \mathsf{d}_2\rangle \ \rightarrow\ \cdots\ \rightarrow\ \langle \mathsf{D.A}_n, \mathsf{d}_n\rangle$ such that $\mathsf{c}_1 = \mathsf{d}_1$ and $\mathsf{c}_j = \exists_{-Var(\mathsf{A}, \mathsf{c}_1)}\mathsf{d}_j$ for each $j \in [2, n]$. This notion can be extended to consider infinite computations (and infinite traces) in the obvious way. We also call an infinite trace $\mathsf{c}_1; \mathsf{c}_2 \ldots$ "active" iff, for any $i \geq 1$, there exists $j > i$ such that $\mathcal{D} \models \neg(\mathsf{c}_i \rightarrow \mathsf{c}_j)$ (on the other hand, the implication $\mathcal{D} \models (\mathsf{c}_j \rightarrow \mathsf{c}_i)$ holds for any $j \ \geq\ i$ when considering traces produced by CCP derivations, since they are monotonically increasing). So an active trace is that produced by a computation that continuously updates the store by adding new constraints. Clearly, when considering infinite computations, one is interested mainly in those producing active traces, as the others are essentially pure loops, which stop producing new results after a finite number of steps.

The essential result we use for extending Theorem 5.12 to infinite traces is the following. If a CCP configuration can produce all the finite prefixes of an infinite trace, then it can produce the infinite trace itself. The following lemma contains a slightly stronger version of it. With a minor abuse of notation, in the following by; we also denote the operator that concatenates traces. Thus, if s_i are traces and c_i are constraints for $i \in [1, n]$, then $\mathsf{c}_1; \mathsf{s}_1; \mathsf{c}_2; \mathsf{s}_2 \ldots; \mathsf{s}_{n-1}; \mathsf{c}_n$ denotes the trace obtained by concatenating the s_i e c_i in the obvious way.

LEMMA 5.14. *Let* D.A *be a CCP process and* c_0 *be a constraint. Assume that* $\langle \mathsf{D.A}, \mathsf{c}_0\rangle$ *produces the* (*infinitely many*) *finite traces*

$$\begin{array}{l} \mathsf{c}_0 \\ \mathsf{c}_0; \mathsf{s}_{1,1}; \mathsf{c}_1 \\ \mathsf{c}_0; \mathsf{s}_{2,1}; \mathsf{c}_1; \mathsf{s}_{2,2}; \mathsf{c}_2 \\ \mathsf{c}_0; \mathsf{s}_{3,1}; \mathsf{c}_1; \mathsf{s}_{3,2}; \mathsf{c}_2; \mathsf{s}_{3,3}; \mathsf{c}_3 \\ \vdots \end{array}$$

where the $\mathsf{c}_0, \mathsf{c}_1, \mathsf{c}_2 \ldots$ *are different constraints* (*i.e., for any* i, $\mathcal{D} \models \neg(\mathsf{c}_i \rightarrow \mathsf{c}_{i+1})$) *and the* $\mathsf{s}_{i,j}$ *are* (*finite*) *subtraces such that, for each* $j \geq 1$, *the* (*infinite*) *set containing the lengths* $\{|\mathsf{s}_{1,j}|, |\mathsf{s}_{2,j}|, |\mathsf{s}_{3,j}|, \ldots\}$ *admits a* (*finite*) *maximal element. Then* $\langle \mathsf{D.A}, \mathsf{c}_0 \rangle$ *also produces the infinite trace* $\mathsf{c}_0; \mathsf{s}_1; \mathsf{c}_1; \mathsf{s}_2; \mathsf{c}_2; \mathsf{s}_3; \mathsf{c}_3; \ldots$ *where, for each* $j \geq 1$, $\mathsf{s}_j = \mathsf{s}_{i,j}$ *for some* $i \geq 1$.

PROOF. The proof uses the Koenig lemma and the fact that the transition system defining the CCP operational semantics is finitely branching.

Let us denote by m_j the maximal element appearing in the set $(\{|\mathsf{s}_{1,j}|, |\mathsf{s}_{2,j}|, |\mathsf{s}_{3,j}|, \ldots\}$ for each $j \geq 1$, that is, m_j is the maximal length of the subtraces $\mathsf{s}_{i,j}$ for a fixed j and $i = 1, 2, \ldots$. We now construct a tree T representing the (infinitely many) finite traces

$$\begin{array}{l} \mathsf{c}_0 \\ \mathsf{c}_0; \mathsf{s}_{1,1}; \mathsf{c}_1 \\ \mathsf{c}_0; \mathsf{s}_{2,1}; \mathsf{c}_1; \mathsf{s}_{2,2}; \mathsf{c}_2 \\ \mathsf{c}_0; \mathsf{s}_{3,1}; \mathsf{c}_1; \mathsf{s}_{3,2}; \mathsf{c}_2; \mathsf{s}_{3,3}; \mathsf{c}_3 \\ \vdots \end{array}$$

produced by $\langle \mathsf{D.A}, \mathsf{c}_0 \rangle$. The nodes of the tree T are labeled by configurations of the form $\langle \mathsf{D.B}, \mathsf{c}_i \rangle$, for some i, and the edges are labeled by the subtraces $\mathsf{s}_{i,j}$. More precisely, the tree T is defined inductively, as follows:

Base Step. The root (level 0) of T is labeled by $\langle \mathsf{D.A}, \mathsf{c}_0 \rangle$. For each derivation of the form $\langle \mathsf{D.A}, \mathsf{c}_0 \rangle \rightarrow^* \langle \mathsf{D.A}_{i,1}, \mathsf{c}_1 \rangle$ which performs at most $\mathsf{m}_1 + 1$ transition steps and produces the trace $\mathsf{c}_0; \mathsf{s}_{i,1}$ we add a son N of the root (at level 1) labeled by $\langle \mathsf{D.A}_{i,1}, \mathsf{c}_1 \rangle$ and an edge, labeled by $\mathsf{s}_{i,1}$, connecting the root and N.

Inductive Step. Assume that T has depth $n - 1$ and let $\langle \mathsf{D.A}_{i,n-1}, \mathsf{c}_{n-1} \rangle$ be a configuration labeling a node N at level $n - 1$. For each derivation of the form $\langle \mathsf{D.A}_{i,n-1}, \mathsf{c}_{n-1} \rangle \rightarrow^* \langle \mathsf{D.A}_{i,n}, \mathsf{c}_n \rangle$ that performs at most $\mathsf{m}_n + 1$ transition steps we add a son N' of N labeled by $\langle \mathsf{D.A}_{i,n}, \mathsf{c}_n \rangle$ and add an edge labeled by $\mathsf{s}_{i,n}$, connecting N and N'.

Note that the number of configurations $\langle \mathsf{D.A}_{i,n}, \mathsf{c}_n \rangle$ obtained in this way is finite, since we allow at most $\mathsf{m}_n + 1$ transition steps. Therefore, we construct a finitely branching tree.

On the other hand, such a tree contains infinitely many nodes, as it contains all the (different) constraints c_i with $i \geq 1$. Then, from the Koenig lemma, it follows that the tree contains an infinite branch and this, by construction of the tree, implies that $\langle \mathsf{D.A}, \mathsf{c}_0 \rangle$ produces the infinite trace $\mathsf{c}_0; \mathsf{s}_1; \mathsf{c}_1; \mathsf{s}_2; \mathsf{c}_2 \ldots \mathsf{s}_n; \mathsf{c}_n; \ldots$ where, for each $j \geq 1$, $\mathsf{s}_j = \mathsf{s}_{i,j}$ for some $i \geq 1$. □

We also need the following Lemma.

LEMMA 5.15. *Let* $\mathsf{D}_0, \ldots, \mathsf{D}_n$ *be a restricted transformation sequence and* A *be an agent. If* $\langle \mathsf{D}_0.\mathsf{A}, \mathsf{c}_0 \rangle$ *produces the trace* $\mathsf{c}_0; \mathsf{s}_1; \mathsf{c}_1; \mathsf{s}_2; \mathsf{c}_2 \ldots \mathsf{s}_m; \mathsf{c}_m$, *where the* c_i *are different constraints and the* s_i *are subtraces of constraints all equal to* c_{i-1}, *then* $\langle \mathsf{D}_n.\mathsf{A}, \mathsf{c}_0 \rangle$ *produces the trace* $\mathsf{c}_0; \mathsf{s}'_1; \mathsf{c}_1; \mathsf{s}'_2; \mathsf{c}_2 \ldots \mathsf{s}'_m; \mathsf{c}_m$ *such that, for any* $i \in [1, m]$, *there exists* k_i *such that* $|\mathsf{s}'_i| \leq |\mathsf{s}_i| + \mathsf{k}_i$. *Furthermore, the vice versa (obtained by exchanging* D_0 *with* D_n *in the previous statement) holds as well.*

PROOF. The first part follows from Theorem 5.12. The part concerning the length is a direct consequence of the definition of the transformation sequence, since each transformation operation can at most add or delete a finite number of computation steps. □

We then obtain the following extension of Theorem 5.12. Here we consider the obvious extension of the relation $\preceq$ to the case of infinite traces.

THEOREM 5.16. *Let* $\mathsf{D}_0, \ldots, \mathsf{D}_n$ *be a restricted transformation sequence and* A *be an agent.*

—*If* $\langle \mathsf{D}_0.\mathsf{A}, \mathsf{c}_0 \rangle$ *produces the infinite active trace* s, *then* $\langle \mathsf{D}_n.\mathsf{A}, \mathsf{c}_0 \rangle$ *produces an infinite trace* s' *such that* $\mathsf{s} \preceq \mathsf{s}'$.
—*Conversely, if* $\langle \mathsf{D}_n.\mathsf{A}, \mathsf{c}_0 \rangle$ *produces the infinite active trace* s, *then* $\langle \mathsf{D}_0.\mathsf{A}, \mathsf{c}_0 \rangle$ *produces an infinite trace* s' *such that* $\mathsf{s} \preceq \mathsf{s}'$.

PROOF. Assume that $\langle \mathsf{D}_0.\mathsf{A}, \mathsf{c}_0 \rangle$ produces the infinite active trace

$$\mathsf{t} : \mathsf{c}_0; \mathsf{s}_1; \mathsf{c}_1; \mathsf{s}_2; \mathsf{c}_2; \mathsf{s}_3; \mathsf{c}_3 \ldots$$

where, in order to simplify the notation, we assume that the c_i are different constraints while the s_i are sequences of constraints all equal to c_{i-1} (so the s_i are sequences of stuttering steps). Clearly, by definition of produced sequence, $\langle \mathsf{D}_0.\mathsf{A}, \mathsf{c}_0 \rangle$ also produces the (infinitely many) finite prefixes of t:

$$\begin{array}{l}
\mathsf{c}_0 \\
\mathsf{c}_0; \mathsf{s}_1; \mathsf{c}_1 \\
\mathsf{c}_0; \mathsf{s}_1; \mathsf{c}_1; \mathsf{s}_2; \mathsf{c}_2 \\
\mathsf{c}_0; \mathsf{s}_1; \mathsf{c}_1; \mathsf{s}_2; \mathsf{c}_2; \mathsf{s}_3; \mathsf{c}_3 \\
\vdots
\end{array}$$

From Lemma 5.15, it follows that $\langle \mathsf{D}_n.\mathsf{A}, \mathsf{c}_0 \rangle$ produces the traces

$$\begin{array}{l}
\mathsf{c}_0 \\
\mathsf{c}_0; \mathsf{s}'_{1,1}; \mathsf{c}_1 \\
\mathsf{c}_0; \mathsf{s}'_{2,1}; \mathsf{c}_1; \mathsf{s}'_{2,2}; \mathsf{c}_2 \\
\mathsf{c}_0; \mathsf{s}'_{3,1}; \mathsf{c}_1; \mathsf{s}'_{3,2}; \mathsf{c}_2; \mathsf{s}'_{3,3}; \mathsf{c}_3 \\
\vdots
\end{array}$$

where, for any $j \geq 1$, there exists k_j such that for any $i \in [1, j]$ we have that $|\mathsf{s}'_{i,j}| \leq |\mathsf{s}_j| + \mathsf{k}_j$. Therefore, the set $\{|\mathsf{s}_{1,j}|, |\mathsf{s}_{2,j}|, |\mathsf{s}_{3,j}|, \ldots\}$ admits a (finite) maximal element for each j. Lemma 5.14 then implies that $\langle \mathsf{D_n.A}, \mathsf{c_0} \rangle$ produces the infinite trace t': $\mathsf{c_0}; \mathsf{s}'_1; \mathsf{c_1}; \mathsf{s}'_2; \mathsf{c_2}; \mathsf{s}'_3; \mathsf{c_3} \ldots$, and clearly, by construction, $\mathsf{t} \preceq \mathsf{t}'$ holds. And, analogously, vice versa. $\square$

5.3.1 *Preservation of Termination.* The results we have presented guarantee the correctness of the transformation system w.r.t. various semantics based on produced constraints. We should mention however that these results do not imply that the system preserves nondeclarative properties such as *termination*. In fact, in case of *nonactive* traces (that from a certain point do not generate any new constraint), the semantics we have considered equate infinite and finite traces.

A full treatment of infinite computations is beyond the scope of this article, and is left for future work.

Nevertheless, we claim that the transformation system we propose here cannot introduce nontermination. That is, if the initial program, for a given configuration, does not produce any infinite computations, then this is also the case for the transformed program.

We now provide a sketch of a proof of this claim by considering a specific class of declarations and by showing the intuitive, informal, argument that indicates the proof methodology to be used for the general case.

Let us then assume that declarations do not contain mutually recursive definitions (note that mutually recursive definitions can usually be eliminated by means of unfolding). We also concentrate on the restricted system, which preserves active traces. In the following, we say that a configuration $\langle \mathsf{D.A}, \mathsf{c} \rangle$ *terminates* if it produces only finite computations, while we say that it does not terminate if it also produces at least one infinite derivation.

Let $\mathsf{D_0}, \ldots, \mathsf{D_n}$ be a transformation sequence, and assume that $\langle \mathsf{D_n.A}, \mathsf{c} \rangle$ has an infinite (nonactive)[7] trace. This implies that there exists a derivation $\xi = \langle \mathsf{D_n.A}, \mathsf{c} \rangle \rightarrow \langle \mathsf{D_n.A_1}, \mathsf{c_1} \rangle \rightarrow^* \cdots \rightarrow^* \langle \mathsf{D_n.A_j}, \mathsf{c_j} \rangle \rightarrow^* \cdots$, where for some k, for each $i \geq k$, $\exists_{\mathsf{Var(A,c)}} \mathsf{c_i} = \exists_{\mathsf{Var(A,c)}} \mathsf{c_k}$ holds. Assume also that for each $\mathsf{i} \in [0, \mathsf{n}-1]$, $\langle \mathsf{D_i.A}, \mathsf{c} \rangle$ terminates.

It is easy to see that the only operation that might introduce nontermination is the folding one (all other operations are clearly “safe” in this respect). So the situation is the following:

$$\begin{array}{ll} \mathsf{d}\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{A}'] & \in \mathsf{D_{n-1}} \\ \mathsf{f}\colon \mathsf{B} \leftarrow \mathsf{A}' & \in \mathsf{D_0} \\ \mathsf{d}'\colon \mathsf{H} \leftarrow \mathsf{C}[\mathsf{B}] & \in \mathsf{D_n} \end{array}$$

This operation can introduce nontermination only when it introduces recursion, i.e., when the definition of B depends on the one of H. The typical case is when B and H have the same predicate, and in the following, for the sake of simplicity, we assume that this is the case, so we assume that

[7]In case of active traces, our result on the preservation of intermediate results guarantees the preservation of termination.

$$\begin{array}{ll} \mathsf{d}\colon \mathsf{p}(\tilde{\mathsf{X}}) \leftarrow \mathsf{C}[\mathsf{A}'] & \in \mathsf{D}_{n-1} \\ \mathsf{f}\colon \mathsf{p}(\tilde{\mathsf{Y}}) \leftarrow \mathsf{A}' & \in \mathsf{D}_0 \\ \mathsf{d}'\colon \mathsf{p}(\tilde{\mathsf{X}}) \leftarrow \mathsf{C}[\mathsf{p}(\tilde{\mathsf{Y}})] & \in \mathsf{D}_n \end{array}$$

From the definition of folding, we have that $\mathsf{C}[\ ]$ is a *guarding context* and $\mathsf{Var}(\mathsf{A}') \cap \mathsf{Var}(\mathsf{C}, \tilde{\mathsf{X}}) \subseteq \tilde{\mathsf{Y}}$ (f and d are suitably renamed so that the variables they have in common are only those occurring in A′). Since C is a guarding context, let us assume that $\mathsf{C}[\ ] = \mathsf{C}'[\sum_{i=1}^{n} \mathsf{ask}(\mathsf{c}'_i) \to \mathsf{A}'_i]$, where $\mathsf{A}'_1 = \mathsf{C}''[\ ]$ and $\mathsf{C}'[\ ]$ and $\mathsf{C}''[\ ]$ are nonguarding contexts. If the infinite computation is due to the folding operation, then the derivation ξ must contain an infinite number of calls of the form $\mathsf{p}(\tilde{\mathsf{Y}})\sigma_i$, where, for each $i \geq 1$, σ_i is a renaming and the current store d_i entails $\mathsf{c}'_1\sigma_i$. Moreover, assume that A is of the form $\mathsf{C}_0[\mathsf{p}(\tilde{\mathsf{v}})]$.

Now, by the definition of the transformation sequence, the unfolding is the only operation that can introduce a new ask action, thus the guard c'_1 in the context C[ ] was certainly introduced during an unfolding operation of an agent in A′ with a recursive definition (recall that d must be obtained from f, thus by unfolding A′ we must obtain C[A′], and we are restricted to the case of direct recursion). Therefore, A′ must contain an atom q, whose definition in D_0 is

$$\mathsf{d}\colon \mathsf{q}(\tilde{\mathsf{Z}}) \leftarrow \mathsf{D}[\mathsf{q}(\tilde{\mathsf{W}})] \quad \in \mathsf{D}_0$$

where the weakest produced constraint of D is precisely $\mathsf{c}'_1\rho$, for some appropriate renaming. Notice also that all tell actions present in D can be skipped (they are always in parallel with the rest, they don't form a guard). Because of this, by taking c as an initial store, we can show that there exist an infinite derivation starting from $\langle \mathsf{D}_0.\mathsf{C}_0[\mathsf{p}(\tilde{\mathsf{V}})], \mathsf{c}\rangle$ where, from a certain point of the derivation j, the current store d satisfies $\exists_{\mathsf{Var}(\mathsf{C}_0[\mathsf{p}(\tilde{\mathsf{v}})],\mathsf{c})}\mathsf{d}_j = \exists_{\mathsf{Var}(\mathsf{C}_0[\mathsf{p}(\tilde{\mathsf{v}})],\mathsf{c})}\mathsf{c}'_1$.

This is in contrast with the hypothesis made on the original program, thus showing that no new infinite computation is generated.

In the rest of the article we provide some extra examples of transformations and, in the Appendix, the technical proofs of the correctness results.

6. MORE EXAMPLES

The following example is inspired by the one in Etalle et al. [1998]. It shows that the transformation system can be used to simplify the dynamic behavior of a program to the point where it can be used to prove deadlock-freeness. All the operations in it are of the restricted sort; thus the transformation preserves the semantics of the intermediate results, as well as those of terminating derivations.

Here and in the following, we say that a variable X is *instantiated* to a term t in case the current store entails X = t. Accordingly, we also say that an agent instantiates a variable X to t in case that the agent adds the constraint X = t to the store. Finally, we say that X is instantiated if the store entails X = t for some nonvariable term t.

Example 6.1. Consider the following simple Collect-Deliver program, which uses a buffer of length one:

```
collect_deliver ← collect(Xs) ∥ deliver(Xs).

collect(Xs) ← % collects tokens and puts them in the queue Xs
        ask(∃_{X,Xs′} Xs = [X|Xs′]) → tell(Xs = [X|Xs′]) ∥ get_token(X) ∥ collect(Xs′)
    + ask(Xs = [ ]) → stop.

deliver([Y|Ys]) ← % delivers the tokens in the queue Xs
        ask(Y = eof) → tell(Ys = [ ])
    + ask(Y ≠ eof) → deliver_token(Y) ∥ deliver(Ys).
```

The dynamic behavior of this program is not elementary. collect(Xs) behaves as follows: (a) it waits until more information for the variable Xs is produced (b1) if Xs is instantiated to [X|Xs′] (i.e. when the store entails $\exists_{X,Xs'}$ Xs = [X|Xs′]), then (by using get_token(X)) it instantiates X with the value it collects (b2) if Xs is instantiated to [ ] it stops. On the other hand, deliver(Xs) performs the actions: (a) it instantiates Xs to [Y|Ys] (this activates collect(Xs)); then (b) it waits until Y is instantiated. Now there are two possibilities: (c1) if Y is the *end of file* character, then it instantiates Ys to [ ] (this will also stop the collector); (c2) otherwise it delivers Y (by using deliver_token(Y)) and proceeds with the recursive call (which will further activate collect).

Thus, collect-deliver actually implements a communication channel with a buffer of length one, and Xs is a bidirectional communication channel. Note also that proving that this program is deadlock-free is not trivial.

We now proceed with the transformation. First, we unfold deliver(Xs) in the body of the first definition. The result, after cleaning up the definition via a (restricted) tell elimination, is

```
collect_deliver ← collect([Y|Ys]) ∥
   (ask(Y = eof) → tell(Ys = [ ])
   + ask(Y ≠ eof) → deliver_token(Y) ∥ deliver(Ys)).
```

We then, unfold collect([Y|Ys]) in the resulting definition, and obtain

```
collect_deliver ←
        (ask(∃_{X,Xs} [Y|Ys] = [X|Xs]) → tell([Y|Ys] = [X|Xs]) ∥ get_token(X) ∥ collect(Xs)
        + ask([Y|Ys] = [ ]) → stop)
    ∥
        (ask(Y = eof) → tell(Ys = [ ])
        + ask(Y ≠ eof) → deliver_token(Y) ∥ deliver(Ys))
```

This definition can be simplified: first, by an ask simplification, we obtain

```
collect_deliver ←
        (ask(true) → tell([Y|Ys] = [X|Xs]) ∥ get_token(X) ∥ collect(Xs)
        + ask(false) → stop)
    ∥
        (ask(Y = eof) → tell(Ys = [ ])
        + ask(Y ≠ eof) → deliver_token(Y) ∥ deliver(Ys))
```

We can then eliminate the branch ask(false) → stop, eliminate tell([Y|Ys] = [X|Xs]), and eliminate the ask(true); the result is

```
collect_deliver ← get_token(Y) ∥ collect(Ys) ∥
    (ask(Y = eof) → tell(Ys = [ ])
    + ask(Y ≠ eof) → deliver_token(Y) ∥ deliver(Ys))
```

We now apply the restricted distributive operation in order to bring collect(Ys) inside the scope of the ask construct. Notice that collect(Ys) requires Ys. Remark 5.7 allows us to apply the operation.

```
collect_deliver ← get_token(Y) ∥
    (ask(Y = eof) → collect(Ys) ∥ tell(Ys = [ ])
    + ask(Y ≠ eof) → deliver_token(Y) ∥ collect(Ys) ∥ deliver(Ys))
```

We can now fold collect(Ys) ∥ deliver(Ys), using the original definition collect_deliver ← collect(Xs) ∥ deliver(Xs). We obtain

```
collect_deliver ← get_token(Y) ∥
    (ask(Y = eof) → collect(Ys) ∥ tell(Ys = [ ])
    + ask(Y ≠ eof) → deliver_token(Y) ∥ collect_deliver)
```

To clean up the result, we can now eliminate tell(Ys = [ ]), unfold the obtained collect([ ]) agent, and perform the usual clean-up operations on the result. Our final program is the simple

```
collect_deliver ← get_token(Y) ∥
    (ask(Y = eof) → stop
    + ask(Y ≠ eof) → deliver_token(Y) ∥ collect_deliver)
```

It is important to compare this to the initial program. In particular, three aspects are worth noticing.

First, as opposed to the initial program, the resulting one has a straightforward dynamic behavior. For instance, if we consider the agent collect_deliver, we can easily see that it is deadlock-free in the latter program, while in the original one this is not at all immediate. After proving that the transformation does not introduce *or eliminate* any deadlocking branch in the semantics of the program, we are able to state that "since the resulting program is deadlock-free then the initial program is also deadlock-free." Thus, the program's transformations can profitably be used as an analysis tool: it is in fact often easier to prove deadlock-freeness for a transformed version of a program than for the original one.

Second, the resulting program is more efficient than the initial one: in fact, it does not need to use the global store as heavily as the initial one for passing the parameters between collect and deliver.

Finally, it is straightforward to check that all transformation operations used here are of the restricted kind; hence, by Theorem 5.12 Strong Total Correctness this transformation is correct w.r.t. the sequence of intermediate results.

We now show an application of our methodology with a third example, containing an *extended folding* operation (see the discussion after Definition 3.15): this is the case when the replaced agent coincides with an instance of the body of the folding definition.

Example 6.2. We consider a stream protocol problem where two input streams are merged into an output stream. An input stream consists of lines of messages, and each line has to be passed to the output stream without interruption. Input and output streams are dynamically constructed by a reader and a monitor process, respectively. A reader communicates with the monitor by means of a buffer of length one, and is synchronized in such a way that it can read a new message only when the buffer is empty (i.e., when the previous message has been processed by the monitor). On the other hand, the monitor can only access a buffer when it is not empty (i.e., when the corresponding reader has put a message into its buffer). This protocol is implemented by the following program, STREAMER:

```
streamer ← reader(left,Ls) ∥ reader(right,Rs) ∥ monitor(Ls,Rs,idle)

reader(Channel,Xs) ←
   ask(∃X,Xs′ Xs = [X|Xs′]) → tell(Xs = [X|Xs′]) ∥ read(Channel,X) ∥
   reader(Channel,Xs′) + ask(Xs = [ ]) → stop.

monitor([L|Ls],[R|Rs],State) ←
   (ask(State = idle) → % waiting for an input
        (ask(char(L)) → monitor([L|Ls],[R|Rs],left)
        + ask(char(R)) → monitor([L|Ls],[R|Rs],right))

   + ask(State = left) → ask(char(L)) → write(L) ∥    % processing the left stream
        (ask(L = eof) → tell(Ls = [ ]) ∥ onestream([R|Rs])
        + ask(L = eol) → monitor(Ls,[R|Rs],idle)
        + ask(L ≠ eol AND L ≠ eof) → monitor(Ls,[R|Rs],left))

   + ask(State = right) → ask(char(R)) → → ... )     % analogously for the right stream

onestream([X|Xs]) ←
   ask(char(X)) →
        (ask(X = eof) → tell(Xs = [ ])
        + ask(X ≠ eof) → write(X) ∥ onestream(Xs))
```

Here, the primitive agent read(Channel,X) is supposed to read an input token from channel Channel and instantiate X with the read value; similarly, write(X) writes the value of X to the (unique) output stream. The primitive constraint predicate char is *true* if its argument is either a printable (e.g., ASCII) character or if it is equal to eol or eof, which are constants denoting the *end of line* and the *end of file* characters, respectively. Furthermore, the agent reader(Channel,Xs) waits to process Channel until Xs is instantiated; monitor(Ls,Rs,State) takes care of merging Ls and Rs and of writing to the output; the agent onestream(Xs) takes care of handling the single stream Xs (when one of the streams has finished). Finally, the constants left, right, and idle describe the state of the monitor, i.e., if it is processing a message from the left stream, from the right stream, or if it is in an idle situation, respectively.

Note that reader(Channel,Xs) suspends until Xs is instantiated and that Xs will eventually be instantiated by the monitor process.

We can now transform the STREAMER program in order to improve its efficiency. First, we add the following new declaration to the original program:

```
handle_two(L,R,State) ← reader(left,Ls) ∥ reader(right,Rs) ∥ monitor([L|Ls],[R|Rs],State)
```

Next, we unfold the agent monitor([L Ls],[R|Rs],State) in the new declaration, and then perform the subsequent tell eliminations (these are restricted by virtue of the argument presented after Definition 5.8). The result of these operations is the following program:

```
handle_two(L,R,State) ← reader(left,Ls) ∥ reader(right,Rs) ∥
   (ask(State = idle) →     % waiting for an input
        (ask(char(L)) → monitor([L|Ls],[R|Rs],left)
        + ask(char(R)) → monitor([L|Ls],[R|Rs],right))

   + ask(State = left) → ask(char(L)) → write(L) ∥     % processing the left stream
        (ask(L = eof) → tell(Ls = [ ]) ∥ onestream([R|Rs])
        + ask(L = eol) → monitor(Ls,[R|Rs],idle)
        + ask(L ≠ eol AND L ≠ eof) → monitor(Ls,[R|Rs],left))

   + ask(State = right) → ask(char(R)) → ··· )     % analogously for the right stream
```

According to Definition 3.11, the agent reader(left, Ls) requires the variable Ls, and reader(right, Rs) requires the variable Rs. By Remark 3.12, it is possible for us to apply the distribution operation twice[8] and bring it inside the ask constructs. The result is the following program:

```
handle_two(L,R,State) ←
   (ask(State = idle) →     % waiting for an input
        (ask(char(L)) → reader(left,Ls) ∥ reader(right,Rs) ∥
                        monitor([L|Ls],[R|Rs],left)
        + ask(char(R)) → reader(left,Ls) ∥ reader(right,Rs) ∥
                        monitor([L|Ls],[R|Rs],right))

   + ask(State = left) → ask(char(L)) → write(L) ∥     % processing the left stream
        (ask(L = eof) → reader(left,Ls) ∥ reader(right,Rs) ∥ tell(Ls = [ ])
                             ∥ onestream([R|Rs])
        + ask(L = eol) → reader(left,Ls) ∥ reader(right,Rs) ∥ monitor(Ls,[R|Rs],idle)
        + ask(L ≠ eol AND L ≠ eof) → reader(left,Ls) ∥ reader(right,Rs) ∥
                             monitor(Ls,[R|Rs],left))

   + ask(State = right) → ask(char(R)) → ... )     % analogously for the right stream
```

In this program we can now eliminate tell(Ls = [ ]) in the agent reader(left, Ls) ∥ reader(right, Rs) ∥ tell(Ls = [ ]) ∥ onestream([R|Rs]) thus obtaining.[9]

[8]Remark 5.7 also guarantees in both cases that it is a restricted distribution operation.

[9]Again, it is true that the variable Ls here also occurs elsewhere in the definition, but since it occurs only on choice-branches different than the one on which the agent in consideration lies, we can assume it to be renamed.

```
handle_two(L,R,State) ←
    (ask(State = idle) →    % waiting for an input
          (ask(char(L)) → reader(left,Ls) ‖ reader(right,Rs) ‖
                          monitor([L|Ls],[R|Rs],left)
          + ask(char(R)) → reader(left,Ls) ‖ reader(right,Rs) ‖
                           monitor([L|Ls],[R|Rs],right))

    + ask(State = left) → ask(char(L)) → write(L) ‖    % processing the left stream
          (ask(L = eof) → reader(left,[ ]) ‖ reader(right,Rs) ‖ onestream([R|Rs])
          + ask(L = eol) → reader(left,Ls) ‖ reader(right,Rs) ‖ monitor(Ls,[R|Rs],idle)
          + ask(L ≠ eol AND L ≠ eof) → reader(left,Ls) ‖ reader(right,Rs) ‖
                                       monitor(Ls,[R|Rs],left))

    + ask(State = right) → ask(char(R)) → ··· )    % analogously for the right stream
```

In this program, the unfolding of the agent reader(left, [ ]) yields as result the agent

```
          ask(∃X,Xs′ [ ] = [X|Xs′]) → tell([ ] = [X|Xs′]) ‖ read(Channel,X)
                                      ‖ reader(Channel,Xs′)
    + ask([ ] = [ ]) → stop.
```

By (trivial) guard simplification, this can become

```
          ask(false) → tell([ ] = [X|Xs′]) ‖ read(Channel,X) ‖ reader(Channel,Xs′)
      + ask(true) → stop.
```

Now, by using branch elimination, we can eliminate the first branch and by applying the conservative ask elimination we can transform the second branch into stop. The application of these operations yield the following:

```
handle_two(L,R,State) ←
    (ask(State = idle) →    % waiting for an input
          (ask(char(L)) → reader(left,Ls) ‖ reader(right,Rs) ‖
                          monitor([L|Ls],[R|Rs],left)
          + ask(char(R)) → reader(left,Ls) ‖ reader(right,Rs) ‖
                           monitor([L|Ls],[R|Rs],right))

    + ask(State = left) → ask(char(L)) → write(L) ‖    % processing the left stream
          (ask(L = eof) → reader(right,Rs) ‖ onestream([R|Rs])
          + ask(L = eol) → reader(left,Ls) ‖ reader(right,Rs) ‖ monitor(Ls,[R|Rs],idle)
          + ask(L ≠ eol AND L ≠ eof) → reader(left,Ls) ‖ reader(right,Rs) ‖
                                       monitor(Ls,[R|Rs],left))

    + ask(State = right) → ask(char(R)) → ... )    % analogously for the right stream
```

We now apply the backward instantiation operation to monitor(Ls,[R|Rs],idle) and to monitor(Ls,[R|Rs],left). Cleaning up the result with a tell elimination,[10] amounts to instantiating Ls to [L′|Ls′]. Hence we have obtained

[10]This is the first operation in this example that is not restricted.

```
handle_two(L,R,State) ←
    (ask(State = idle) →     % waiting for an input
          (ask(char(L)) → reader(left,Ls) ∥ reader(right,Rs) ∥
                                   monitor([L|Ls],[R|Rs],left)
          + ask(char(R)) → reader(left,Ls) ∥ reader(right,Rs) ∥
                                   monitor([L|Ls],[R|Rs],right))

    + ask(State = left) → ask(char(L)) → write(L) ∥     % processing the left stream
          (ask(L = eof) → reader(right,Rs) ∥ onestream([R|Rs])
          + ask(L = eol) → reader(left,[L′|Ls′]) ∥ reader(right,Rs) ∥
                                   monitor([L′|Ls′],[R|Rs],idle)
          + ask(L≠eol AND L≠eof) → reader(left,[L′|Ls′]) ∥ reader(right,Rs) ∥
                                   monitor([L′|Ls′],[R|Rs],left))

    + ask(State = right) → ask(char(R)) → ... )     % analogously for the right stream
```

In order to prepare the program for the folding operation, we need one more clean-up phase: using the unfolding and some simplification operations, we can replace each call reader(left, [L′|Ls′]) with read(left, L′) ∥ reader(left, Ls′). The result of these operations is the following program:

```
handle_two(L,R,State) ←
    (ask(State = idle) →     % waiting for an input
          (ask(char(L)) → reader(left,Ls) ∥ reader(right,Rs) ∥
                                   monitor([L|Ls],[R|Rs],left)
          + ask(char(R)) → reader(left,Ls) ∥ reader(right,Rs) ∥
                                   monitor([L|Ls],[R|Rs],right))

    + ask(State = left) → ask(char(L)) → write(L) ∥     % processing the left stream
          (ask(L = eof) → reader(right,Rs) ∥ onestream([R|Rs])
          + ask(L = eol) → read(left,L′) ∥ reader(left,Ls′) ∥ reader(right,Rs) ∥
                                   monitor([L′|Ls′],[R|Rs],idle)
          + ask(L≠eol AND L≠eof) → read(left,L′) ∥ reader(left,Ls′) ∥ reader(right,Rs) ∥
                                   monitor([L′|Ls′],[R|Rs],left))

    + ask(State = right) → ask(char(R)) → ... )     % analogously for the right stream
```

We can now apply the *extended folding* operation twice. The first folding allows us to replace reader(left,Ls) ∥ reader(right,Rs) ∥ monitor([L|Ls],[R|Rs],left) with handle_two(L,R,left). With the second one we replace reader(left,Ls) ∥ reader(right,Rs) ∥ monitor([L|Ls],[R|Rs],right) with handle_two(L,R,right). Recall that the extended folding operation, as described in Section 3.7, occurs when the replaced agent coincides with a nontrivial *instance* of the body of the folding definition; as already explained in the discussion after Definition 3.15, this is only a shorthand for a sequence of tell introduction, folding and tell elimination, as described in Section 3.7. After these two operations the resulting program is

```
handle_two(L,R,State) ←
    (ask(State = idle) →     % waiting for an input
          (ask(char(L)) → handle_two(L,R,left)
          + ask(char(R)) → handle_two(L,R,right)
```

```
  + ask(State = left) → ask(char(L)) → write(L) ∥    % processing the left stream
        (ask(L = eof) → reader(right,Rs) ∥ onestream([R|Rs])
        + ask(L = eol) → read(left,L′) ∥ reader(left,Ls′) ∥ reader(right,Rs) ∥
                             monitor([L′|Ls′],[R|Rs],idle)
        + ask(L≠eol AND L≠eof) → read(left,L′) ∥ reader(left,Ls′) ∥ reader(right,Rs) ∥
                             monitor([L′|Ls′],[R|Rs],left))

  + ask(State = right) → ask(char(R)) → ... )    % analogously for the right stream
```

We then perform two more extended foldings: in the first we replace the agent reader(left,Ls′) ∥ reader(right,Rs) ∥ monitor([L′|Ls′],[R|Rs],idle); with the agent handle_two(L′,R,idle); in the latter, we replace reader(left,Ls′) ∥ reader(right,Rs) ∥ monitor([L′|Ls′],[R|Rs],left) with handle_two(L′,R,left). The resulting program is

```
handle_two(L,R,State) ←
  ask(State = idle) →    % waiting for an input
        (ask(char(L)) → handle_two(L,R,left)
        + ask(char(R)) → handle_two(L,R,right)

  + ask(State = left) → ask(char(L)) → write(L) ∥    % processing the left stream
        (ask(L = eof) → reader(right,Rs) ∥ onestream([R|Rs])
        + ask(L = eol) → read(left,L′) ∥ handle_two(L′,R,idle)
        + ask(L ≠ eol AND L ≠ eof) → read(left,L′) ∥ handle_two(L′,R,left)

  + ask(State = right) → ask(char(R)) → ... )    % analogously for the right stream
```

Note that now the definition of handle_two is recursive. Moreover, the above program is almost completely independent of the definition of reader. In order to eliminate the atom reader(right,Rs) as well, we use an unfold/fold transformation similar to (but simpler than) the previous one. This transformation starts with the new definition:[11]

```
handle_one(X, Channel) ← reader(Channel,Xs) ∥ onestream([X|Xs])
```

After the transformation, we end up with the following definition:

```
handle_one(X, Channel) ← ask(char(X)) →
  (ask(X = eof) → stop
  + ask(X≠eof) → write(X) ∥ read(Channel,X′) ∥ handle_one(X′,Channel))
```

In this case also, the folding operation allows us to save computational space. In fact, the parallel composition of reader and onestream in the original definition leads to the construction of a list containing all the data read so far. In a concurrent setting, this list could easily be of unbounded size and monotonically increasing. The initial definition employs a computational space which is linear in the input. After the transformation we have a definition that does not build the list any longer, and could be optimized to employ only constant space (this could be achieved by a using a garbage-collection mechanism that allows us to reuse the space allocated for local variables).

[11]This definition is presented here for the sake of clarity; however, recall that we assume that it is added to the original program at the beginning of the transformation.

We now continue with the last steps of our example. By folding handle_one into the last definition of handle_two, we obtain

```
handle_two(L,R,State) ←
    (ask(State = idle) →      % waiting for an input
          (ask(char(L)) → handle_two(L,R,left)
          + ask(char(R)) → handle_two(L,R,right)

    + ask(State = left) → ask(char(L)) → write(L) ||      % processing the left stream
          (ask(L = eof) → handle_one(R,right)
          + ask(L = eol) → read(left,L′) || handle_two(L′,R,idle)
          + ask(L ≠ eol AND L ≠ eof) → read(left,L′) || handle_two(L′,R,left)

    + ask(State = right) → ask(char(R)) → ... )      % analogously for the right stream
```

We now want to let streamer benefit from the improvements we have obtained via this transformation. First, we transform its definition by applying the backward instantiation to monitor(Ls,Rs,idle), and obtain

```
streamer ← reader(left,[L|Ls]) || reader(right,[R|Rs]) || monitor([L|Ls],[R|Rs],idle).
```

Next, we unfold the two reader atoms and eliminate the redundant ask and tell guards.

```
streamer ← read(left,L) || reader(left,Ls) ||
           read(right,R) || reader(right,Rs) || monitor([L|Ls],[R|Rs],idle).
```

We can now fold handle_two in it (via an extended folding operation), obtaining

```
streamer ← read(left,L) || read(right,R) || handle_two(L,R,idle).
```

Note that this last folding operation is applied to a nonguarding context. As discussed in Remark 3.16, we can apply folding in this case also because the definition of streamer is never modified or used by the transformation. So we can simply assume that the original definition of streamer contained a dummy ask guard, as in

```
streamer ← ask(true) → ( read(left,L) || reader(left,Ls) || read(right,R) ||
                         reader(right,Rs) || monitor([L|Ls],[R|Rs],idle))
```

We then assume that the folding operation is applied to this definition, and that the guard ask(true) will eventually be removed by an ask elimination operation.

In the final program, we only need the definitions of streamer and handle_two together with the those of the built-it predicates. Observe that the definition of streamer is much more efficient than the original one. First, it now benefits from a straightforward left-to-right dataflow. In the initial program the variables Ls and Rs are employed as bidirectional communication channels. In fact, there exist two agents (reader and monitor) that alternate in "instantiating" them further. This is not the case in the final program, where for each variable it is clear which is the agent that is supposed to "instantiate" it (i.e., to progressively add information to the store about it). This fact implies that in the final program a number of powerful compile-time (low-level) optimizations are possible that are not possible in the first program.

Second, the number of suspension points is dramatically reduced: in the original program, reader had to suspend and awaken itself at each input token. In the final program, streamer is independent from reader and has to suspend less often.

Last, but certainly not least, as previously mentioned, streamer does not now construct the list and could be optimized to employ a constant computational space, while in its initial version it employed a space linear in the input; that is possibly unbounded. It is worth remarking that in a concurrent setting, processes are often not meant to end their computation, in which case it is of vital importance that the computational space remain bounded in size; thus, in this context, a space gain like the one obtained in the above example makes the difference between a viable and a nonviable definition.

Example 6.3. This is a variation on a standard example for unfold/fold transformations: a program computing the sum and the length of the elements in a list. The variation consists in the fact that we consider only the elements of the list that are larger than the given parameter Limit. We assume here that the constraint system being used incorporates some arithmetic domain. Hence, in the following program, we also use arithmetic constraints, with the obvious intended meaning.

```
sumlen(Xs,Limit,S,L) ← sum(Xs,Limit,S) ∥ len(Xs,Limit,L)

sum(Xs,Limit,S) ←
   (ask(Xs = [ ]) → tell(S = 0)
   + ask(∃_{Y,Ys} (Xs = [Y|Ys] ∧ Y ≤ Limit)) → tell(Xs = [Y|Ys]) ∥
            sum(Ys,Limit,S)
   + ask(∃_{Y,Ys} (Xs = [Y|Ys] ∧ Y > Limit)) → tell (Xs = [Y|Ys]) ∥
            sum(Ys,Limit,S′) ∥
            tell(S = S′+Y))

len(Xs,Limit,L) ←
   (ask(Xs = [ ]) → tell(L = 0)
   + ask(∃_{Y,Ys} (Xs = [Y|Ys] ∧ Y ≤ Limit)) → tell (Xs = [Y|Ys]) ∥
            len(Ys,Limit,L)
   + ask(∃_{Y,Ys} (Xs = [Y|Ys] ∧ Y > Limit)) → tell (Xs = [Y|Ys]) ∥
            len(Ys,Limit,L′) ∥
            tell(L = L′ + 1))
```

With two unfoldings, we obtain

```
sumlen(Xs,Limit,S,L) ←
   (ask(Xs = [ ]) → tell(S = 0)
   + ask(∃_{Y,Ys} (Xs = [Y|Ys] ∧ Y ≤ Limit)) → tell (Xs = [Y|Ys]) ∥
            sum(Ys,Limit,S)
   + ask(∃_{Y,Ys} (Xs = [Y|Ys] ∧ Y > Limit)) → tell (Xs = [Y|Ys]) ∥
            sum(Ys,Limit,S′) ∥
            tell(S = S′ + Y))
    ∥
   (ask(Xs = [ ]) → tell(L = 0)
    + ask($\exists_{Y',Ys'}$ (Xs = [Y'|Ys']  ∧ Y' ≤ Limit)) → tell (Xs = [Y'|Ys']) ‖
                len(Ys',Limit,L)
    + ask($\exists_{Y',Ys'}$ (Xs = [Y'|Ys']  ∧ Y' > Limit)) → tell (Xs = [Y'|Ys']) ‖
                len(Ys',Limit,L') ‖
                tell(L = L' + 1))
```

We now apply the (restricted) distribution operation; in practice, we now bring one choice inside the other:

```
sumlen(Xs,Limit,S,L) ←
    (ask(Xs = [ ]) → tell(S = 0) ‖
                (ask(Xs = [ ]) → tell(L = 0)
                + ask($\exists_{Y',Ys'}$ (Xs = [Y'|Ys']  ∧ Y' ≤ Limit)) → tell (Xs = [Y'|Ys']) ‖
                                len(Ys',Limit,L)
                + ask($\exists_{Y',Ys'}$ = [Y'|Ys'] and Y' > Limit)) → tell (Xs = [Y'|Ys']) ‖
                                len(Ys',Limit,L') ‖
                                tell(L = L'+1))
    + ask($\exists_{Y,Ys}$ (Xs = [Y|Ys]  ∧ Y ≤ Limit)) → tell (Xs = [Y|Ys]) ‖
                sum(Ys,Limit,S) ‖
                (ask(Xs = [ ]) → tell(L = 0)
                + ask($\exists_{Y',Ys'}$ = [Y'|Ys'] and Y' ≤ limit)) → tell (Xs = [Y'|Ys']) ‖
                                len(Ys',Limit,L)
                + ask($\exists_{Y',Ys'}$ = [Y'|Ys'] and Y' > Limit) → tell (Xs = [Y'|Ys']) ‖
                                len(Ys',Limit,L') ‖
                                tell(L = L' + 1))
    + ask($\exists_{Y,Ys}$ (Xs = [Y|Ys]  ∧ Y > Limit)) → tell (Xs = [Y|Ys]) ‖
                sum(Ys,Limit,S') ‖
                tell(S = S'+Y) ‖
                (ask(Xs = [ ]) → tell(L = 0)
                + ask($\exists_{Y',Ys'}$ (Xs = [Y'|Ys']  ∧ Y' ≤ Limit)) → tell (Xs = [Y'|Ys']) ‖
                                len(Ys',Limit,L)
                + ask($\exists_{Y',Ys'}$ (Xs = [Y'|Ys']  ∧ Y' > Limit)) → tell (Xs = [Y'|Ys']) ‖
                                len(Ys',Limit,L') ‖
                                tell(L = L'+1)))
```

It is worth noting that the applicability conditions of Definition 3.10 are trivially satisfied, thanks to the fact that both choices depend on the same variable Xs. Note also that in this case we cannot apply Remark 3.12, in fact this is an example of a distribution operation that is not possible with the tools of Etalle et al. [1998].

By using ask simplification, followed by branch elimination and a conservative ask elimination, we obtain the following program. Note that the ask simplification is possible here because we can take arithmetic constraints into account.

```
sumlen(Xs,Limit,S,L) ←
    (ask(Xs = [ ]) → tell(S = 0) ‖ tell(L = 0)
    + ask($\exists_{Y,Ys}$ (Xs = [Y|Ys]  ∧ Y ≤ Limit)) → tell (Xs = [Y|Ys]) ‖
```

```
                sum(Ys,Limit,S) ‖
                tell (Xs = [Y′|Ys′]) ‖
                len(Ys′,Limit,L)
    + ask(∃Y,Ys (Xs = [Y|Ys] ∧ Y > Limit)) → tell (Xs = [Y|Ys]) ‖
                sum(Ys,Limit,S′) ‖
                tell(S = S′+Y) ‖
                tell (Xs = [Y′|Ys′]) ‖
                len(Ys′,Limit,L′) ‖
                tell(L = L′+1))
```

Via a tell simplification (first and last nonrestricted operation in this example), we can transform tell(Xs = $[Y'|Ys']$) into tell([Y|Ys] = $[Y'|Ys']$), and subsequently apply a tell elimination, and obtain

```
sumlen(Xs,Limit,S,L) ←
    (ask(Xs = [ ]) → tell(S = 0) ‖ tell(L = 0)
    + ask(∃Y,Ys (Xs = [Y|Ys] ∧ Y ≤ Limit)) → tell (Xs = [Y|Ys]) ‖
                sum(Ys,Limit,S) ‖
                len(Ys,Limit,L)
    + ask(∃Y,Ys (Xs = [Y|Ys] ∧ Y > Limit)) → tell (Xs = [Y|Ys]) ‖
                sum(Ys,Limit,S′) ‖
                tell(S = S′+Y) ‖
                len(Ys,Limit,L′) ‖
                tell(L = L′+1))
```

We can now apply the folding operation.

```
sumlen(Xs,Limit,S,L) ←
    (ask(Xs = [ ]) → tell(S = 0) ‖ tell(L = 0)
    + ask(∃Y,Ys (Xs = [Y|Ys] ∧ Y ≤ Limit)) → tell (Xs = [Y|Ys]) ‖
                sumlen(Ys,Limit,S,L)
    + ask(∃Y,Ys (Xs = [Y|Ys] ∧ Y > Limit)) → tell (Xs = [Y|Ys]) ‖
                sumlen(Ys,Limit,S′,L′) ‖
                tell(S = S′+Y) ‖
                tell(L = L′+1))
```

Again, we have reached a point in which the main definition is directly recursive. Moreover, the number of choice points encountered while traversing a list is now half of what it was initially.

7. RELATED WORK

As mentioned in the Introduction, this is one of the few attempts to apply fold/unfold techniques in the field of concurrent languages. In fact, in the literature we find only three papers that are relatively closely related to the present one: Ueda and Furukawa [1988] defined transformation systems for the concurrent logic language GHC; Ueda [1986] and Sahlin [1995] defined a partial evaluator for AKL, while in 1996 de Francesco and Santone presented a transformation system for CCS [Milner 1989].

The transformation system we are proposing builds on the systems defined in the papers above, and can be considered their extension. Differently from the previous cases, our system is defined for a generic (concurrent) constraint language. Thus, together with some new transformations such as distribution, backward instantiation, and branch elimination, we also introduce specific operations that allow constraint simplification and elimination (although some constraint simplification is done in Sahlin [1995]).

It is interesting but not straightforward to compare our system with that of Ueda and Furukawa [1988]. It is specific for the GHC language, which has a different syntactic structure from CCP, and uses the Herbrand universe as a computational domain. Owing to this, [Ueda and Furukawa [1988] employ operations that are completely different from ours. In particular, our unfolding operation is replaced by immediate execution and case splitting [Ueda and Furukawa 1988]. Our unfolding is a weaker operation with a broader applicability than case-splitting, since the latter operation involves moving synchronization points, and therefore requires suitable applicability conditions. Furthermore, the distribution operation is not present in Ueda and Furukawa [1988], as it would not be possible in the syntactic structure of GHC. However, in many cases the effect of distribution can be achieved in Ueda and Furukawa [1988] by introducing a new clause followed by case-splitting. In order to clarify this, we report below how the transformations of Example 6.1 could be mimicked in GHC by using the operations of Ueda and Furukawa [1988]. The transformation in the following example was provided by a reviewer of this article.

Example 7.1. The initial program collect_deliver considered in Example 6.1, in terms of the GHC syntax, is

```
1: collect_deliver :- | collect(Xs), deliver(Xs).
2: collect([X|Xs]) :- | get_token(X), collect(Xs).
3: collect([])     :- | true.
4: deliver(Ys0) :- | Ys0$=[Y|Ys], deliver_2(Y,Ys).
5: deliver_2(eof,Ys) :- | Ys$=[].
6: deliver_2(Y,Ys) :- Y\=$eof | deliver_token(Y), deliver(Ys).
```

The presence of `deliver_2` is due to the fact that GHC does not allow nested guards. The first operation to be used is an *immediate execution*, applied to clause (1). The result is

```
7: collect_deliver :- | collect(Xs), Xs=[Y|Ys], deliver_2(Y,Ys).
```

By normalizing this, we obtain

```
8: collect_deliver :- | collect([Y|Ys]), deliver_2(Y,Ys).
```

Another *immediate execution* operation yields

```
9: collect_deliver :- | get_token(Y), collect(Ys), deliver_2(Y,Ys).
```

Now, we need to *introduce a new definition*.

```
10: collect_deliver_2(Y) :- | collect(Ys), deliver_2(Y,Ys).
```

By applying the *case splitting* operation to this, we obtain

```
11: collect_deliver_2(eof) :- | collect(Ys), Ys=[].

12: collect_deliver_2(Y) :- Y\=eof | collect(Ys),
                            deliver_token(Y), deliver(Ys).
```

By normalizing clause 11, and subsequently applying an *immediate execution* operation, we obtain

```
13: collect_deliver_2(eof) :- | true.
```

We can apply the folding operation to (12) and (9), and the resulting program is

```
collect_deliver :- | get_token(Y), collect_deliver_2(Y).
collect_deliver_2(eof) :- | true.
collect_deliver_2(Y) :- Y\=eof| deliver_token(Y),
                        collect_deliver.
```

It is worth noting how it is possible to achieve a program that is basically identical to the one in Example 6.1, despite the completely different nature of the operation.

Unlike Ueda and Furukawa [1988], we provide a more flexible definition for the folding operation, which allows the folding clause to be recursive (really a step forward in the context of folding operations that are themselves capable of introducing recursion) and frees the *initial program* from having to be partitioned into P_{new} and P_{old}. In fact, as opposed to virtually all fold operations that enable the introduction of recursion presented so far (the only exception being Francesco and Santone [1996]), the applicability of the folding operation does not depend on the transformation history (which has always been one of its "obscure sides"), but relies on plain syntactic criteria. The idea of using a guarded folding in order to obtain applicability conditions independent of the transformation history was first introduced by de Francesco and Santone [1996] in the CCS setting. However, their technical development is rather different from ours, in particular our correctness results and proofs are completely different from those sketched in Francesco and Santone [1996].

As mentioned previously, and unlike our case, in Sahlin [1995] a definition of *ask elimination* allows us to remove potentially selectable branches; the consequence is that the resulting transformation system is only *partially* (thus not totally) correct. We should mention that in Sahlin [1995] two preliminary assumptions on "scheduling" are made in such a way that this limitation is actually less constraining than it might appear.

8. CONCLUSIONS

We have introduced an unfold/fold transformation system for CCP, and have proved its total correctness w.r.t. the input/output semantics defined by the observables $\mathcal{O}$, which also takes into account the termination modes. This semantics corresponds (modulo irrelevant differences due to the treatment of failure and local variables) to the one proposed in de Boer and Palamidessi [1991]. This is one of the two fully abstract "standard" semantics for CCP; the other being the one defined in Saraswat et al. [1991]. (Actually, these two semantic models have been proved to be isomorphic [de Boer and Palamidessi 1992], provided that the termination mode and the consistency checks are eliminated.)

We have also shown that the proposed transformation system preserves another, stronger, semantics that takes into account the intermediate results of computations up to logical implication (Theorem 5.1). We argue that this result should be strong enough to also transform programs that might not terminate, in particular to transform reactive programs. Nevertheless, in addition to this, we present a restricted transformation system, obtained from the initial one by adding some (relatively mild) restrictions on some operations. We show that this second system preserves the trace semantics of programs (up to simulation, Theorem 5.12), and therefore is totally correct w.r.t. the semantics, $\mathcal{O}_i$—which takes into account all the intermediate results (Corollary 5.13). We also proved that this system preserves active infinite computations, and we claim that, more generally, this system does not introduce into the transformed program any new infinite computation not present in the original one.

As shown by the examples, this system can be used for the optimization of concurrent constraint programs, both in terms of time and space. In fact, it allows us to eliminate unnecessary suspension points (and therefore to reduce sequentiality), reduce the number of communication channels, and avoid the construction of some global data structures. The system can also be used to simplify the dynamic behavior of a program, thus allowing us to directly prove absence of deadlock.

Concerning future work, there exist other techniques for proving deadlock-freeness for CCP programs, notably in Codish et al. [1994] a methodology based on abstract interpretation is defined. It could be interesting to investigate an integration of our methodology with abstract interpretation tools. We are also considering a formal comparison of various transformation systems (in particular, our system and that of Ueda and Furukawa [1988]) to assess their relative strengths. This task is not immediate, since the target languages are different.

APPENDIX

In this Appendix we provide the detailed proofs for the results that ensure that the transformation system we have defined is totally correct. In particular, we provide the detailed proofs for Theorems 4.13 and 5.12. In order to obtain a self-contained Appendix, some technical lemmas in the article are repeated here. In what follows, we are going to refer to a fixed *transformation sequence* $D_0, \ldots, D_n$.

LEMMA A.1. *Assume that there exists a derivation* $\langle D.C[A], c\rangle \rightarrow^* \langle D.C'[A], c'\rangle$ *where* c *is a satisfiable constraint, and the context* C′[ ] *has the form*

$$A_1 \parallel \ldots \parallel \bar{C}[\ ] \parallel \ldots \parallel A_n$$

and, for each $j \in [1, n]$, A_j *is either a choice agent, a procedure call, or the agent* Stop. *Then* $\mathcal{D} \models (\mathrm{pc}(\bar{C}[\ ]) \wedge c') \rightarrow \mathrm{pc}(C[\ ])$ *holds, and in case* $\bar{C}[\ ]$ *is the empty* *context,* $\mathcal{D} \models \mathsf{c}' \rightarrow \mathsf{pc}(\mathsf{C}[\,])$ *also holds.*

Proof. By a straightforward inductive argument, it follows that if there exists a derivation $\langle \mathsf{D.C[A]}, \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D.C}'[\underline{\mathsf{A}}], \mathsf{c}'\rangle$, then $\mathcal{D} \models (\mathsf{pc}(\mathsf{C}'[\,]) \wedge \mathsf{c}') \rightarrow \mathsf{pc}(\mathsf{C}[\,])$. Now, if $\mathsf{C}'[\,]$ has the form $\mathsf{A}_1 \parallel \ldots \parallel \bar{\mathsf{C}}[\,] \parallel \ldots \parallel \mathsf{A}_n$, where each A_j is either a choice agent or a procedure call or Stop, then $\mathsf{pc}(\mathsf{C}'[\,]) = \mathsf{pc}(\bar{\mathsf{C}}[\,])$, which implies $\mathcal{D} \models (\mathsf{pc}(\bar{\mathsf{C}}[\,]) \wedge \mathsf{c}') \rightarrow \mathsf{pc}(\mathsf{C}[\,])$. Obviously, if $\bar{\mathsf{C}}[\,]$ is the empty context, then $\mathsf{pc}(\mathsf{C}'[\,]) = \mathsf{true}$, and the second part of the lemma follows. □

We now prove Proposition 4.5.

Proposition 4.5 (Partial Correctness). *If, for each agent* A, $\mathcal{O}(\mathsf{D}_0.\mathsf{A}) = \mathcal{O}(\mathsf{D}_i.\mathsf{A})$*, then, for each agent* A, $\mathcal{O}(\mathsf{D}_i.\mathsf{A}), \supseteq \mathcal{O}(\mathsf{D}_{i+1}.\mathsf{A})$.

Proof. We now show that given an agent A and a satisfiable constraint c_I, if there exists a derivation $\xi = \langle \mathsf{D}_{i+1}.\mathsf{A}, \mathsf{c}_I\rangle \rightarrow^* \langle \mathsf{D}_{i+1}.\mathsf{B}, \mathsf{c}_F\rangle$, with $m(\mathsf{B}, \mathsf{c}_F) \in \{\mathsf{ss}, \mathsf{dd}, \mathsf{ff}\}$, then there also exists a derivation $\xi' = \langle \mathsf{D}_i.\mathsf{A}, \mathsf{c}_I\rangle \rightarrow^* \langle \mathsf{D}_i.\mathsf{B}', \mathsf{c}'_F\rangle$ with $\exists_{-Var(\mathsf{A},\mathsf{c}_I)}\mathsf{c}'_F = \exists_{-Var(\mathsf{A},\mathsf{c}_I)}\mathsf{c}_F$ and $m(\mathsf{B}', \mathsf{c}'_F) = m(\mathsf{B}, \mathsf{c}_F)$. By Definition 4.1, this implies the thesis. The proof is by induction on the length l of the derivation.

($l = 0$). In this case $\xi = \langle \mathsf{D}_{i+1}.\mathsf{A}, \mathsf{c}_I\rangle$. By the definition, $\langle \mathsf{D}_i.\mathsf{A}, \mathsf{c}_I\rangle$ is also a derivation of length 0, and then the thesis holds.

($l > 0$). If the first step of derivation ξ does not use rule **R4**, then the proof follows from the inductive hypothesis. In fact, if $\xi = \langle \mathsf{D}_{i+1}.\mathsf{A}, \mathsf{c}_I\rangle \rightarrow \langle \mathsf{D}_{i+1}.\mathsf{A}_1, \mathsf{c}_1\rangle \rightarrow^* \langle \mathsf{D}_{i+1}.\mathsf{B}, \mathsf{c}_F\rangle$, then by the inductive hypothesis, there exists a derivation

$$\xi'' = \langle \mathsf{D}_i.\mathsf{A}_1, \mathsf{c}_1\rangle \rightarrow^* \langle \mathsf{D}_i.\mathsf{B}', \mathsf{c}'_F\rangle$$

with $\exists_{-Var(\mathsf{A}_1,\mathsf{c}_1)}\mathsf{c}'_F = \exists_{-Var(\mathsf{A}_1,\mathsf{c}_1)}\mathsf{c}_F$ and $m(\mathsf{B}', \mathsf{c}'_F) = m(\mathsf{B}, \mathsf{c}_F)$. We can assume, without loss of generality, that $Var(\mathsf{A}, \mathsf{c}_I) \cap Var(\xi'') \subseteq Var(\mathsf{A}_1, \mathsf{c}_1)$. Therefore, there exists a derivation $\xi' = \langle \mathsf{D}_i.\mathsf{A}, \mathsf{c}_I\rangle \rightarrow^* \langle \mathsf{D}_i.\mathsf{B}', \mathsf{c}'_F\rangle$. Now, to prove the thesis, it is sufficient to observe that, by the hypothesis on the variables, $\exists_{-Var(\mathsf{A},\mathsf{c}_I)}\mathsf{c}'_F = \exists_{-Var(\mathsf{A},\mathsf{c}_I)}(\mathsf{c}_1 \wedge \exists_{-Var(\mathsf{A}_1,\mathsf{c}_1)}\mathsf{c}'_F) = \exists_{-Var(\mathsf{A},\mathsf{c}_I)}(\mathsf{c}_1 \wedge \exists_{-Var(\mathsf{A}_1,\mathsf{c}_1)}\mathsf{c}_F) = \exists_{-Var(\mathsf{A},\mathsf{c}_I)}\mathsf{c}_F$.

Now assume that the first step of derivation ξ uses rule **R4**, and let $\mathsf{d}' \in \mathsf{D}_{i+1}$ be the declaration in the first step of ξ. If d' was not modified in the transformation step from D_i to D_{i+1} (that is, $\mathsf{d}' \in \mathsf{D}_i$), then the result follows from the inductive hypothesis. We then assume that $\mathsf{d}' \notin \mathsf{D}_i$, d' is the result of the transformation operation applied to obtain D_{i+1}, and we now distinguish various cases according to the operation itself.

Case 1. d' is the result of an unfolding operation.
In this case the proof is straightforward.

Case 2. d' is the result of a tell elimination or of a tell introduction.
In this case the thesis follows from a straightforward analysis of the possible derivations that use d or d'. First, observe that for any derivation that uses a declaration $\mathsf{H} \leftarrow \mathsf{C}[\mathsf{tell}(\tilde{\mathsf{s}} = \tilde{\mathsf{t}}) \parallel \mathsf{B}]$, we can construct another derivation such that the agent $\mathsf{tell}(\tilde{\mathsf{s}} = \tilde{\mathsf{t}})$ is evaluated before B. Moreover, for any constraint c

such that $\exists_{\mathsf{dom}(\sigma)}\mathsf{c} = \exists_{\mathsf{dom}(\sigma)}\mathsf{c}\sigma$ (where σ is a relevant most general unifier of $\tilde{\mathsf{s}}$ and $\tilde{\mathsf{t}}$), there exists a derivation step $\langle \mathsf{D_i.B_1}\sigma, \mathsf{c}\sigma\rangle \rightarrow \langle \mathsf{D_i.B_2}\sigma, \mathsf{c}'\rangle$ if and only if there exists a derivation step $\langle \mathsf{D_i.B_1}, \mathsf{c} \wedge (\tilde{\mathsf{s}} = \tilde{\mathsf{t}})\rangle \rightarrow \langle \mathsf{D_i.B_2}, \mathsf{c}''\rangle$, where, for some constraint e, $\mathsf{c}' = \mathsf{e}\sigma$, $\mathsf{c}'' = \mathsf{e} \wedge (\tilde{\mathsf{s}} = \tilde{\mathsf{t}})$, and therefore $\mathsf{c}' = \exists_{\mathsf{dom}(\sigma)}\mathsf{c}''$. Finally, since, by definition, σ is idempotent and the variables in the domain of σ do not occur in either $\mathsf{C}[\ ]$ or H for any constraint e, we have that $\exists_{-\mathit{Var}(\mathsf{A},\mathsf{c_I})}\mathsf{e}\sigma = \exists_{-\mathit{Var}(\mathsf{A},\mathsf{c_I})}(\mathsf{e} \wedge (\tilde{\mathsf{s}} = \tilde{\mathsf{t}}))$.

Case 3. d' is the result of a backward instantiation.

Let d be the corresponding declaration in $\mathsf{D_i}$. The situation is the following:

—d: $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]$,
—d': $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}}) \parallel \mathsf{tell(b)} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]$,

where f: $\mathsf{p}(\tilde{\mathsf{s}}) \leftarrow \mathsf{tell(b)} \parallel \mathsf{H} \in \mathsf{D_i}$ has no variable in common with d (the case d': $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}}) \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]$ is analogous, and hence omitted). In this case

$$\begin{aligned}\xi = \ & \langle \mathsf{D_{i+1}.C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}\rangle \rightarrow \langle \mathsf{D_{i+1}.C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}}) \parallel \mathsf{tell(b)} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I}\rangle \\ & \rightarrow^* \langle \mathsf{D_{i+1}.B}, \mathsf{c_F}\rangle.\end{aligned}$$

By the inductive hypothesis, there exists a derivation

$$\chi = \langle \mathsf{D_i.C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}}) \parallel \mathsf{tell(b)} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I}\rangle \rightarrow^* \langle \mathsf{D_i.B}'', \mathsf{c}''_{\mathsf{F}}\rangle,$$

with

$$\begin{aligned}&\exists_{-\mathit{Var}(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}}) \parallel \mathsf{tell(b)} \parallel \mathsf{tell}(\tilde{\mathsf{t}}=\tilde{\mathsf{s}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\mathsf{c}''_{\mathsf{F}} \\ &\quad = \exists_{-\mathit{Var}(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}}) \parallel \mathsf{tell(b)} \parallel \mathsf{tell}(\tilde{\mathsf{t}}=\tilde{\mathsf{s}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\mathsf{c_F}\end{aligned}$$

and

$$\mathsf{m}(\mathsf{B}'', \mathsf{c}''_{\mathsf{F}}) = \mathsf{m}(\mathsf{B}, \mathsf{c_F}). \tag{31}$$

Moreover, since $\mathit{Var}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}) \subseteq \mathit{Var}(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}}) \parallel \mathsf{tell(b)} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I})$, we have that

$$\exists_{-\mathit{Var}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c}''_{\mathsf{F}} = \exists_{-\mathit{Var}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c_F}. \tag{32}$$

If $\mathsf{p}(\tilde{\mathsf{t}})$ is not evaluated in χ, then the proof is immediate. Otherwise, by the definition of χ and since $\mathsf{f} \in \mathsf{D_i}$, there also exists a derivation

$$\chi' = \langle \mathsf{D_i.C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I}\rangle \rightarrow^* \langle \mathsf{D_i.B}', \mathsf{c}'_{\mathsf{F}}\rangle$$

such that $\exists_{-\mathit{Var}(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\mathsf{c}'_{\mathsf{F}} = \exists_{-\mathit{Var}(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\mathsf{c}''_{\mathsf{F}}$ and $\mathsf{m}(\mathsf{B}', \mathsf{c}'_{\mathsf{F}}) = \mathsf{m}(\mathsf{B}'', \mathsf{c}''_{\mathsf{F}})$. Therefore, by (32) and (31),

$$\exists_{-\mathit{Var}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c}'_{\mathsf{F}} = \exists_{-\mathit{Var}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c_F} \text{ and } \mathsf{m}(\mathsf{B}', \mathsf{c}'_{\mathsf{F}}) = \mathsf{m}(\mathsf{B}, \mathsf{c_F}). \tag{33}$$

By the definition of χ', $\mathit{Var}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}) \cap \mathit{Var}(\chi') \subseteq \mathit{Var}(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I})$. Then, by the definition of derivation and since $\mathsf{d} \in \mathsf{D_i}$,

$$\langle \mathsf{D_i.C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}\rangle \rightarrow \langle \mathsf{D_i.C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I}\rangle \rightarrow^* \langle \mathsf{D_i.B}', \mathsf{c}'_{\mathsf{F}}\rangle,$$

the thesis then follows from (33).

Case 4. d' is obtained from d by either an ask simplification or a tell simplification. We consider only the first case (the proof of the other is analogous, and hence omitted). Let

—d': $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j') \rightarrow \mathsf{A}_j]$, and
—d: $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j) \rightarrow \mathsf{A}_j]$,

where for $j \in [1, n]$, $\mathcal{D} \models \exists_{-Var(\mathsf{q}(\tilde{\mathsf{r}}),\mathsf{C},\mathsf{A}_j)} (\mathsf{pc}(\mathsf{C}[\,]) \wedge \mathsf{c}_j) \leftrightarrow (\mathsf{pc}(\mathsf{C}[\,]) \wedge \mathsf{c}_j')$. According to the definition of pc and by Lemma A.1, for any derivation χ for

$$\left\langle \mathsf{D}_i.\mathsf{C}_l \left[\mathsf{C} \left[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j') \rightarrow \mathsf{A}_j \right] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}}) \right], \mathsf{c}_l \right\rangle$$

there exists a derivation χ' for

$$\left\langle \mathsf{D}_i.\mathsf{C}_l \left[\mathsf{C} \left[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j) \rightarrow \mathsf{A}_j \right] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}}) \right], \mathsf{c}_l \right\rangle,$$

which performs the same steps of χ (possibly in a different order) and such that whenever the choice agent inside $\mathsf{C}[\,]$ is evaluated, the current store implies $\mathsf{pc}(\mathsf{C}[\,])$. The thesis follows from the above equivalence.

Case 5. d' is the result of a branch elimination or of a conservative ask elimination. The proof is straightforward by noting that: (a) according to Definition 4.1, we also consider inconsistent stores resulting from nonterminated computations; (b) an ask action of the form $\mathsf{ask}(\mathsf{true})$ always succeeds.

Case 6. d' is the result of a distribution operation. Let

—d: $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j) \rightarrow \mathsf{B}_j] \in \mathsf{D}_i$,
—d': $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j) \rightarrow (\mathsf{H} \parallel \mathsf{B}_j)] \in \mathsf{D}_{i+1}$,

where $\mathsf{e} = \mathsf{pc}(\mathsf{C}[])$ and for every constraint c such that $Var(\mathsf{c}) \cap Var(\mathsf{d}) \subseteq Var(\mathsf{q}(\tilde{\mathsf{r}}), \mathsf{C})$, if $\langle \mathsf{D}_i.\mathsf{H}, \mathsf{c} \wedge \mathsf{e} \rangle$ is productive, then both the following conditions hold:

—there exists at least one $j \in [1, n]$ such that $\mathcal{D} \models (\mathsf{c} \wedge \mathsf{e}) \rightarrow \mathsf{c}_j$
—for each $j \in [1, n]$, either $\mathcal{D} \models (\mathsf{c} \wedge \mathsf{e}) \rightarrow \mathsf{c}_j$; or $\mathcal{D} \models (\mathsf{c} \wedge \mathsf{e}) \rightarrow \neg \mathsf{c}_j$.

In this case, $\xi = \langle \mathsf{D}_{i+1}.\mathsf{C}_l[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_l \rangle \rightarrow \langle \mathsf{D}_{i+1}.\mathsf{C}_l[\mathsf{C}[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j) \rightarrow (\mathsf{H} \parallel \mathsf{B}_j)] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c}_l \rangle \rightarrow^* \langle \mathsf{D}_{i+1}.\mathsf{B}, \mathsf{c}_F \rangle$. By the inductive hypothesis, there exists a derivation

$$\chi = \left\langle \mathsf{D}_i.\mathsf{C}_l \left[\mathsf{C} \left[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j) \rightarrow (\mathsf{H} \parallel \mathsf{B}_j) \right] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}}) \right], \mathsf{c}_l \right\rangle \rightarrow^* \langle \mathsf{D}_i.\mathsf{B}'', \mathsf{c}_F'' \rangle$$

with

$$\begin{aligned} &\exists_{-Var\left(\mathsf{C}_l\left[\mathsf{C}\left[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j) \rightarrow (\mathsf{H} \parallel \mathsf{B}_j)\right] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})\right], \mathsf{c}_l\right)} \mathsf{c}_F'' \\ &\quad = \exists_{-Var\left(\mathsf{C}_l\left[\mathsf{C}\left[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c}_j) \rightarrow (\mathsf{H} \parallel \mathsf{B}_j)\right] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})\right], \mathsf{c}_l\right)} \mathsf{c}_F \end{aligned}$$

and

$$\mathsf{m}(\mathsf{B}'', \mathsf{c}_F'') = \mathsf{m}(\mathsf{B}, \mathsf{c}_F). \tag{34}$$

Moreover, since $\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_l}) \subseteq \mathit{Var}(\mathsf{C_l}[\mathsf{C}[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_l})$, we have that

$$\exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_l})}\mathsf{c}''_\mathsf{F} = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_l})}\mathsf{c_F}. \tag{35}$$

We now distinguish two cases:

(1) $\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})$ is not evaluated in χ. In this case the proof is obvious.

(2) $\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})$ is evaluated in χ. We have two more possibilities:

(2a) There exists $h \in [1, n]$, such that

$$\begin{aligned}\chi \;=\; & \left\langle \mathsf{D_i}.\mathsf{C_l}\left[\mathsf{C}\left[\sum_{j=1}^{n}\mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})\right] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})\right], \mathsf{c_l}\right\rangle \\ & \rightarrow^* \left\langle \mathsf{D_i}.\mathsf{C_m}\left[\sum_{j=1}^{n}\mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})\right], \mathsf{c_m}\right\rangle \rightarrow \langle \mathsf{D_i}.\mathsf{C_m}[\mathsf{H} \parallel \mathsf{B_h}], \mathsf{c_m}\rangle \\ & \rightarrow^* \langle \mathsf{D_i}.\mathsf{B}'', \mathsf{c}''_\mathsf{F}\rangle,\end{aligned}$$

where $\mathcal{D} \models \mathsf{c_m} \rightarrow \mathsf{c_h}$. In this case the thesis follows immediately, since by using d we can obtain the agent $\mathsf{C_m}[\mathsf{H} \parallel \mathsf{B_h}]$ after having evaluated the choice agent in $\mathsf{C}[\;]$.

(2b) There is no $h \in [1, n]$ such that $\mathcal{D} \models \mathsf{c}''_\mathsf{F} \rightarrow \mathsf{c_h}$. In this case

$$\mathsf{c}''_\mathsf{F} \text{ is satisfiable, } \mathsf{m}(\mathsf{B}'', \mathsf{c}''_\mathsf{F}) = \mathsf{dd}, \tag{36}$$

B'' is the agent $\mathsf{C_F}[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})]$, and

$$\begin{aligned}\chi \;=\; & \left\langle \mathsf{D_i}.\mathsf{C_l}\left[\mathsf{C}\left[\sum_{j=1}^{n}\mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})\right] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})\right], \mathsf{c_l}\right\rangle \\ & \rightarrow^* \left\langle \mathsf{D_i}.\mathsf{C_F}\left[\sum_{j=1}^{n}\mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})\right], \mathsf{c}''_\mathsf{F}\right\rangle \not\rightarrow .\end{aligned}$$

From the definition of derivation, the definition of B'' and the hypothesis that $\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})$ is evaluated in χ, it follows that $\mathsf{C_F}[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})]$ is of the form $\mathsf{A_1} \parallel \ldots \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j}) \parallel \ldots \parallel \mathsf{A_l}$, where either $\mathsf{A_k}$ is a choice agent or $\mathsf{A_k} = \mathsf{Stop}$. By Lemma A.1, $\mathcal{D} \models \mathsf{c}''_\mathsf{F} \rightarrow \mathsf{pc}(\mathsf{C}[\;])$ and by definition of derivation $\mathit{Var}(\mathsf{c}''_\mathsf{F}) \cap \mathit{Var}(\mathsf{d}) \subseteq \mathit{Var}(\mathsf{q}(\tilde{\mathsf{r}}), \mathsf{C})$. Then, since there is no $j \in [1, n]$ such that $\mathcal{D} \models \mathsf{c}''_\mathsf{F} \rightarrow \mathsf{c_j}$, by definition of distribution, $\langle \mathsf{D_i}.\mathsf{H}, \mathsf{c}''_\mathsf{F}\rangle$ is not productive. Then, by definition, $\langle \mathsf{D_i}.\mathsf{H}, \mathsf{c}''_\mathsf{F}\rangle$ has at least one finite derivation $\chi_1 = \langle \mathsf{D_i}.\mathsf{H}, \mathsf{c}''_\mathsf{F}\rangle \rightarrow^* \langle \mathsf{D_i}.\mathsf{H}', \mathsf{c}'_\mathsf{F}\rangle \not\rightarrow$ such that $\mathcal{D} \models \exists_{-\tilde{\mathsf{Z}}}\, \mathsf{c}''_\mathsf{F} \leftrightarrow \exists_{-\tilde{\mathsf{Z}}}\, \mathsf{c}'_\mathsf{F}$, where $\tilde{\mathsf{Z}} = \mathit{Var}(\mathsf{H})$. Moreover, since in a derivation we can add to the store only constraints on the variables occurring in the agents, $\mathsf{c}''_\mathsf{F} = \exists_{-\mathit{Var}(\mathsf{H},\mathsf{c}''_\mathsf{F})}\, \mathsf{c}''_\mathsf{F} = \exists_{-\mathit{Var}(\mathsf{H},\mathsf{c}''_\mathsf{F})}\, \mathsf{c}'_\mathsf{F}$ holds.

Without loss of generality, we can assume that $\mathit{Var}(\chi_1) \cap \mathit{Var}(\chi) \subseteq \mathit{Var}(\mathsf{H}, \mathsf{c}''_\mathsf{F})$. Therefore, by the previous observation,

$$\exists_{-\mathit{Var}\left(\mathsf{C_F}\left[\sum_{j=1}^{n}\mathsf{ask}(\mathsf{c_j}) \rightarrow (\mathsf{H} \parallel \mathsf{B_j})\right)\right], \mathsf{c}''_\mathsf{F})}\, \mathsf{c}'_\mathsf{F} = \mathsf{c}''_\mathsf{F}, \tag{37}$$

and since $\langle D_i.C_F[\sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow (H \parallel B_j)], c''_F\rangle \not\rightarrow$ and $\langle D_i.H', c'_F\rangle \not\rightarrow$, there exists a derivation

$$\chi' = \left\langle D_i.C_I\left[C\left[H \parallel \sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow B_j\right] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})\right], c_I\right\rangle \rightarrow^* \left\langle D_i.C_F\left[H \parallel \sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow B_j\right], c''_F\right\rangle \rightarrow^* \left\langle D_i.C_F[H' \parallel \sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow B_j], c'_F\right\rangle \not\rightarrow .$$

Moreover, since $d \in D_i$, there exists a derivation

$$\xi' = \langle D_i.C_I[q(\tilde{v})], c_I\rangle \rightarrow \left\langle D_i.C_I\left[C\left[H \parallel \sum_{j=1}^n \mathsf{ask}(c_j\right) \rightarrow B_j\right] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})\right], c_I\right\rangle \rightarrow^* \left\langle D_i.C_F\left[H \parallel \sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow B_j\right], c''_F\right\rangle \rightarrow^* \left\langle D_i.C_F\left[H' \parallel \sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow B_j\right], c'_F\right\rangle \not\rightarrow .$$

Finally, to prove the thesis, it is sufficient to observe that from (34), (36), (37), and from the definition of $B' = C_F[H' \parallel \sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow B_j]$, it follows that $m(B', c'_F) = m(B, c_F) = dd$. Moreover,

$$\begin{aligned}
&\exists_{-Var(C_I[q(\tilde{v})],c_I)} c'_F && = \text{ by construction}\\
&\exists_{-Var(C_I[q(\tilde{v})],c_I)}(c''_F \wedge \exists_{-Var(C_F[H \,\parallel\, \sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow B_j], c'_F)} c'_F) && = \text{ by } (37)\\
&\exists_{-Var(C_I[q(\tilde{v})],c_I)} c''_F && = \text{ by } (35)\\
&\exists_{-Var(C_I[q(\tilde{v})],c_I)} c_F
\end{aligned}$$

which concludes the proof of this case.

Case 7. d′ is the result of a folding.
Let

—d: $q(\tilde{r}) \leftarrow C[H]$ be the folded declaration ($\in D_i$),
—f: $p(\tilde{X}) \leftarrow H$ be the folding declaration ($\in D_0$),
—d′: $q(\tilde{r}) \leftarrow C[p(\tilde{X})]$ be the result of the folding operation ($\in D_{i+1}$),

where, by hypothesis, $Var(d) \cap Var(\tilde{X}) \subseteq Var(H)$ and $Var(H) \cap (Var(\tilde{r}) \cup Var(C)) \subseteq Var(\tilde{X})$. In this case, $\xi = \langle D_{i+1}.C_I[q(\tilde{v})], c_I\rangle \rightarrow \langle D_{i+1}.C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_{i+1}.B, c_F\rangle$, and we can assume, without loss of generality, that $Var(C_I[q(\tilde{v})], c_I) \cap Var(H) = \emptyset$.

By the inductive hypothesis, there exists a derivation

$$\chi = \langle D_i.C_I[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_i.B'', c''_F\rangle,$$

with $\exists_{-Var(C_I[C[p(\tilde{X})] \,\parallel\, \mathsf{tell}(\tilde{v}=\tilde{r})],c_I)} c''_F = \exists_{-Var(C_I[C[p(\tilde{X})] \,\parallel\, \mathsf{tell}(\tilde{v}=\tilde{r})],c_I)} c_F$ and

$$m(B'', c''_F) = m(B, c_F). \tag{38}$$

Since $Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}) \subseteq Var(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I})$, we have that

$$\exists_{-Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c}''_\mathsf{F} = \exists_{-Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c_F}. \tag{39}$$

Since by hypothesis for any agent A', $\mathcal{O}(\mathsf{D_0.A}') = \mathcal{O}(\mathsf{D_i.A}')$, there exists a derivation

$$\xi_0 = \langle \mathsf{D_0.C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I}\rangle \rightarrow^* \langle \mathsf{D_0.B_0}, \mathsf{c_0}\rangle$$

such that $\exists_{-Var(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\mathsf{c_0} = \exists_{-Var(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\mathsf{c}''_\mathsf{F}$ and $\mathsf{m}(\mathsf{B_0}, \mathsf{c_0}) = \mathsf{m}(\mathsf{B}'', \mathsf{c}''_\mathsf{F})$. By (38), (39), and since $Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}) \subseteq Var(\mathsf{C_I}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I})$, we have that

$$\exists_{-Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c_0} = \exists_{-Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c_F} \text{ and } \mathsf{m}(\mathsf{B_0}, \mathsf{c_0}) = \mathsf{m}(\mathsf{B}, \mathsf{c_F}). \tag{40}$$

Let f': $\mathsf{p}(\tilde{\mathsf{X}}') \leftarrow \mathsf{H}'$ be an appropriate renaming of f, which renames only the variables in $\tilde{\mathsf{X}}$, such that $Var(\mathsf{d}) \cap Var(\mathsf{f}') = \emptyset$ (note that this is possible, since $Var(\mathsf{H}) \cap (Var(\tilde{\mathsf{r}}) \cup Var(\mathsf{C})) \subseteq Var(\tilde{\mathsf{X}})$). Moreover, by hypothesis, $Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}) \cap Var(\mathsf{H}) = \emptyset$. Then, without loss of generality, we can assume that $Var(\xi_0) \cap Var(\mathsf{f}') \neq \emptyset$ if and only if the procedure call $\mathsf{p}(\tilde{\mathsf{X}})$ is evaluated, in which case declaration f' is used.

Thus there exists a derivation

$$\langle \mathsf{D_0.C_I}[\mathsf{C}[\mathsf{H}' \parallel \mathsf{tell}(\tilde{\mathsf{X}} = \tilde{\mathsf{X}}')] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I}\rangle \rightarrow^* \langle \mathsf{D_0.B}'_0, \mathsf{c_0}\rangle,$$

where $\mathsf{m}(\mathsf{B}'_0, \mathsf{c_0}) = \mathsf{m}(\mathsf{B_0}, \mathsf{c_0})$. By (40), we have

$$\mathsf{m}(\mathsf{B}'_0, \mathsf{c_0}) = \mathsf{m}(\mathsf{B}, \mathsf{c_F}). \tag{41}$$

We now show that we can substitute H for $\mathsf{H}' \parallel \mathsf{tell}(\tilde{\mathsf{X}} = \tilde{\mathsf{X}}')$ in the previous derivation. Since f': $\mathsf{p}(\tilde{\mathsf{X}}') \leftarrow \mathsf{H}'$ is a renaming of f: $\mathsf{p}(\tilde{\mathsf{X}}) \leftarrow \mathsf{H}$, the equality $\tilde{\mathsf{X}} = \tilde{\mathsf{X}}'$ is a conjunction of equations involving only distinct variables. Then, by replacing $\tilde{\mathsf{X}}$ with $\tilde{\mathsf{X}}'$ and vice versa in the previous derivation, we obtain the derivation $\chi_0 = \langle \mathsf{D_0.C_I}[\mathsf{C}[\mathsf{H} \parallel \mathsf{tell}(\tilde{\mathsf{X}}' = \tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I}\rangle \rightarrow^* \langle \mathsf{D_0.B}''_0, \mathsf{c}'_0\rangle$ where

$$\exists_{-Var(\mathsf{C_I}[\mathsf{C}[\mathsf{H} \parallel \mathsf{tell}(\tilde{\mathsf{X}}'=\tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\mathsf{c}'_0 = \exists_{-Var(\mathsf{C_I}[\mathsf{C}[\mathsf{H} \parallel \mathsf{tell}(\tilde{\mathsf{X}}'=\tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\mathsf{c_0}$$
$$\text{and } \mathsf{m}(\mathsf{B}''_0, \mathsf{c}'_0) = \mathsf{m}(\mathsf{B}'_0, \mathsf{c_0}).$$

From (41), it follows that

$$\mathsf{m}\left(\mathsf{B}''_0, \mathsf{c}'_0\right) = \mathsf{m}(\mathsf{B}, \mathsf{c_F}). \tag{42}$$

Then, from (40) and since $Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}) \subseteq Var(\mathsf{C_I}[\mathsf{C}[\mathsf{H} \parallel \mathsf{tell}(\tilde{\mathsf{X}}' = \tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I})$, we obtain

$$\exists_{-Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c}'_0 = \exists_{-Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c_F}. \tag{43}$$

Moreover, we can drop the constraint $\mathsf{tell}(\tilde{\mathsf{X}}' = \tilde{\mathsf{X}})$, since the declarations used in the derivation are renamed apart and, by construction, $Var(\mathsf{C_I}[\mathsf{C}[\mathsf{H}] \parallel \mathsf{tell}(\tilde{\mathsf{r}} = \tilde{\mathsf{v}})], \mathsf{c_I}) \cap Var(\tilde{\mathsf{X}}') = \emptyset$. Hence there exists a derivation $\langle \mathsf{D_0.C_I}[\mathsf{C}[\mathsf{H}] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c_I}\rangle \rightarrow^* \langle \mathsf{D_0}.\overline{\mathsf{B}}_0, \overline{\mathsf{c}}_0\rangle$, which performs exactly the same steps of χ_0, (possibly) except for the evaluation of $\mathsf{tell}(\tilde{\mathsf{X}}' = \tilde{\mathsf{X}})$, and such that $\exists_{-Var(\mathsf{C_I}[\mathsf{C}[\mathsf{H}] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\overline{\mathsf{c}}_0 = \exists_{-Var(\mathsf{C_I}[\mathsf{C}[\mathsf{H}] \parallel \mathsf{tell}(\tilde{\mathsf{v}}=\tilde{\mathsf{r}})],\mathsf{c_I})}\mathsf{c}'_0$ and $\mathsf{m}(\overline{\mathsf{B}}_0, \overline{\mathsf{c}}_0) = \mathsf{m}(\mathsf{B}''_0, \mathsf{c}'_0)$.

From (42), (43), and since $\mathit{Var}(\mathsf{C_I[q(\tilde{v})], c_I}) \subseteq \mathit{Var}(\mathsf{C_I[C[H] \parallel tell(\tilde{v} = \tilde{r})], c_I})$, it follows that

$$\mathsf{m(\bar{B}_0, \bar{c}_0) = m(B, c_F)} \text{ and } \exists_{-\mathit{Var}(\mathsf{C_I[q(\tilde{v})],c_I})}\mathsf{\bar{c}_0} = \exists_{-\mathit{Var}(\mathsf{C_I[q(\tilde{v})],c_I})}\mathsf{c_F}. \tag{44}$$

Since $\mathcal{O}(\mathsf{D_0.A'}) = \mathcal{O}(\mathsf{D_i.A'})$ holds by hypothesis for any agent $\mathsf{A'}$, there exists a derivation

$$\langle \mathsf{D_i.C_I[C[H] \parallel tell(\tilde{v} = \tilde{r})], c_I} \rangle \rightarrow^* \langle \mathsf{D_i.B', c'_F} \rangle$$

where

$$\exists_{-\mathit{Var}(\mathsf{C_I[C[H]} \parallel \mathsf{tell(\tilde{v}=\tilde{r})],c_I})}\mathsf{c'_F} = \exists_{-\mathit{Var}(\mathsf{C_I[C[H]} \parallel \mathsf{tell(\tilde{v}=\tilde{r})],c_I})}\mathsf{\bar{c}_0}$$

and $\mathsf{m(B', c'_F) = m(\bar{B}_0, \bar{c}_0)}$. From (44) and since $\mathit{Var}(\mathsf{C_I[q(\tilde{v})], c_I}) \subseteq \mathit{Var}(\mathsf{C_I[C[H] \parallel tell\ (\tilde{v} = \tilde{r})], c_I})$, we obtain

$$\mathsf{m(B', c'_F) = m(B, c_F)} \text{ and } \exists_{-\mathit{Var}(\mathsf{C_I[q(\tilde{v})],c_I})}\mathsf{c'_F} = \exists_{-\mathit{Var}(\mathsf{C_I[q(\tilde{v})],c_I})}\mathsf{c_F}. \tag{45}$$

Finally, since $\mathsf{d\colon q(\tilde{r}) \leftarrow C[H] \in D_i}$, there exists a derivation

$$\xi' = \langle \mathsf{D_i.C_I[q(\tilde{v})], c_I} \rangle \rightarrow \langle \mathsf{D_i.C_I[C[H] \parallel tell(\tilde{v} = \tilde{r})], c_I} \rangle \rightarrow^* \langle \mathsf{D_i.B', c'_F} \rangle,$$

and the thesis follows from (45). □

Before proving the total correctness result, we need some technical lemmas. Here and in the following, we use the notation $\mathsf{w_t}$ (with $\mathsf{t \in \{ss, dd, ff\}}$) as shorthand for indicating the success weight $\mathsf{w_{ss}}$, the deadlock weight $\mathsf{w_{dd}}$, and the failure weight $\mathsf{w_{ff}}$.

LEMMA A.3. *Let* $\mathsf{q(\tilde{r}) \leftarrow H \in D_0}$, $\mathsf{t \in \{ss, dd, ff\}}$, *and let* $\mathsf{C[\ ]}$ *be a context. For any satisfiable constraint* c *and for any constraint* $\mathsf{c'}$ *such that* $\mathit{Var}(\mathsf{C[q(\tilde{t})], c}) \cap \mathit{Var}(\mathsf{\tilde{r}}) = \emptyset$ *and* $\mathsf{w_t(C[q(\tilde{t})], c, c')}$ *is defined, there exists a constraint* $\mathsf{d'}$ *such that* $\mathsf{w_t(C[q(\tilde{r}) \parallel tell(\tilde{t} = \tilde{r})], c, d') \leq w_t(C[q(\tilde{t})], c, c')}$ *and* $\exists_{-\mathit{Var}(\mathsf{C[q(\tilde{t})],c})}\mathsf{d'} = \exists_{-\mathit{Var}(\mathsf{C[q(\tilde{t})],c})}\mathsf{c'}$.

PROOF. Immediate. □

LEMMA A.4. *Let* $\mathsf{q(\tilde{r}) \leftarrow H \in D_0}$ *and* $\mathsf{t \in \{ss, dd, ff\}}$. *For any context* $\mathsf{C_I[\ ]}$, *any satisfiable constraint* c *and for any constraint* $\mathsf{c'}$, *the following holds*:

(1) *If* $\mathit{Var}(\mathsf{H}) \cap \mathit{Var}(\mathsf{C_I, c}) \subseteq \mathit{Var}(\mathsf{\tilde{r}})$ *and* $\mathsf{w_t(C_I[q(\tilde{r})], c, c')}$ *is defined, then there exists a constraint* $\mathsf{d'}$ *such that* $\mathit{Var}(\mathsf{d'}) \subseteq \mathit{Var}(\mathsf{C_I[H], c})$, $\mathsf{w_t(C_I[H], c, d') \leq w_t(C_I[q(\tilde{r})], c, c')}$ *and* $\exists_{-\mathit{Var}(\mathsf{C_I[q(\tilde{r})],c})}\mathsf{d'} = \exists_{-\mathit{Var}(\mathsf{C_I[q(\tilde{r})],c})}\mathsf{c'}$.
(2) *If* $\mathit{Var}(\mathsf{H}) \cap \mathit{Var}(\mathsf{C_I, c}) \subseteq \mathit{Var}(\mathsf{\tilde{r}})$, $\mathit{Var}(\mathsf{c'}) \cap \mathit{Var}(\mathsf{\tilde{r}}) \subseteq \mathit{Var}(\mathsf{C_I[H], c})$, *and* $\mathsf{w_t(C_I[H], c, c')}$ *is defined, then there exists a constraint* $\mathsf{d'}$ *such that* $\mathsf{w_t(C_I[q(\tilde{r})], c, d') \leq w_t(C_I[H], c, c')}$ *and* $\exists_{-\mathit{Var}(\mathsf{C_I[q(\tilde{r})],c})}\mathsf{d'} = \exists_{-\mathit{Var}(\mathsf{C_I[q(\tilde{r})],c})}\mathsf{c'}$.

PROOF. Immediate. □

The following Lemma is crucial in the proof of completeness.

LEMMA A.5. *Let* $0 \leq i \leq n$, $\mathsf{t \in \{ss, dd, ff\}}$, $\mathsf{cl\colon q(\tilde{r}) \leftarrow H \in D_i}$, *and let* $\mathsf{cl'\colon q(\tilde{r}) \leftarrow H'}$ *be the corresponding declaration in* $\mathsf{D_{i+1}}$ (*in the case* $i < n$). *For any context* $\mathsf{C_I[\ ]}$, *any satisfiable constraint* c, *and for any constraint* $\mathsf{c'}$, *the following holds:*

(1) *If* $\mathit{Var}(\mathsf{H}) \cap \mathit{Var}(\mathsf{C_l}, \mathsf{c}) \subseteq \mathit{Var}(\tilde{\mathsf{r}})$ *and* $\mathsf{w_t}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})], \mathsf{c}, \mathsf{c}')$ *is defined, then there exists a constraint* d' *such that* $\mathit{Var}(\mathsf{d}') \subseteq \mathit{Var}(\mathsf{C_l}[\mathsf{H}], \mathsf{c})$*,* $\mathsf{w_t}(\mathsf{C_l}[\mathsf{H}], \mathsf{c}, \mathsf{d}') \leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})], \mathsf{c}, \mathsf{c}')$*, and* $\exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{d}' = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{c}'$.

(2) *If* $\mathit{Var}(\mathsf{H}, \mathsf{H}') \cap \mathit{Var}(\mathsf{C_l}, \mathsf{c}) \subseteq \mathit{Var}(\tilde{\mathsf{r}})$*,* $\mathit{Var}(\mathsf{c}') \cap \mathit{Var}(\tilde{\mathsf{r}}) \subseteq \mathit{Var}(\mathsf{C_l}[\mathsf{H}], \mathsf{c})$*, and* $\mathsf{w_t}(\mathsf{C_l}[\mathsf{H}], \mathsf{c}, \mathsf{c}')$ *is defined, then there exists a constraint* d' *such that* $\mathit{Var}(\mathsf{d}') \subseteq \mathit{Var}(\mathsf{C_l}[\mathsf{H}'], \mathsf{c})$*,* $\mathsf{w_t}(\mathsf{C_l}[\mathsf{H}'], \mathsf{c}, \mathsf{d}') \leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{H}], \mathsf{c}, \mathsf{c}')$*, and* $\exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{d}' = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{c}'$.

PROOF. Observe that, for $i = 0$, the proof of (1) follows from the first part of Lemma A.4. We prove here that, for each $i \geq 0$,

(a) if 1 holds for i, then 2 holds for i;
(b) if 1 and 2 hold for i, then *1* holds for $i + 1$.

The proof of the Lemma follows from a straightforward inductive argument.

(a) If cl is not affected by the transformation step from $\mathsf{D_i}$ to $\mathsf{D_{i+1}}$, then the result is obvious by choosing $\mathsf{d}' = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{H}],\mathsf{c})}\mathsf{c}'$. Assume then that cl is affected when transforming $\mathsf{D_i}$ to $\mathsf{D_{i+1}}$, and let us distinguish the various cases.

Case 1. $\mathsf{cl}' \in \mathsf{D_{i+1}}$ was obtained from $\mathsf{D_i}$ by unfolding. In this case, the situation is the following:

—cl: $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})] \in \mathsf{D_i}$,
—u: $\mathsf{p}(\tilde{\mathsf{s}}) \leftarrow \mathsf{B} \in \mathsf{D_i}$,
—cl′: $\mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})] \in \mathsf{D_{i+1}}$,

where cl and u are assumed to be renamed so that they do not share variables. Let $\mathsf{n} = \mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]], \mathsf{c}, \mathsf{c}')$. By the definition of a transformation sequence, there exists a declaration $\mathsf{p}(\tilde{\mathsf{s}}) \leftarrow \mathsf{B_0} \in \mathsf{D_0}$. Moreover, by the hypothesis on the variables, $\mathit{Var}(\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})], \mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]) \cap \mathit{Var}(\mathsf{C_l}, \mathsf{c}) \subseteq \mathit{Var}(\tilde{\mathsf{r}})$, and then $\mathit{Var}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]], \mathsf{c}) \cap \mathit{Var}(\tilde{\mathsf{s}}) = \emptyset$. Therefore, by Lemma A.3, there exists a constraint $\mathsf{d_1}$ such that

$$\mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{s}}) \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]], \mathsf{c}, \mathsf{d_1}) \leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]], \mathsf{c}, \mathsf{c}') = \mathsf{n} \tag{46}$$

and

$$\exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]],\mathsf{c})}\mathsf{d_1} = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]],\mathsf{c})}\mathsf{c}'. \tag{47}$$

By the hypothesis on the variables and since u is renamed apart from cl, $\mathit{Var}(\mathsf{B}) \cap \mathit{Var}(\mathsf{C_l}, \mathsf{C}, \tilde{\mathsf{t}}, \mathsf{c}) = \emptyset$, and therefore $\mathit{Var}(\mathsf{B}) \cap \mathit{Var}(\mathsf{C_l}[\mathsf{C}[\,] \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})], \mathsf{c}) \subseteq \mathit{Var}(\tilde{\mathsf{s}})$. Then, by point (1), there exists a constraint d' such that $\mathit{Var}(\mathsf{d}') \subseteq \mathit{Var}(\mathsf{C_l}[\mathsf{C}[\mathsf{B} \parallel \mathsf{tell}\ (\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]], \mathsf{c})$, $\mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]], \mathsf{c}, \mathsf{d}') \leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{s}}) \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]], \mathsf{c}, \mathsf{d_1})$, and

$$\exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{s}})\ \parallel\ \mathsf{tell}(\tilde{\mathsf{t}}=\tilde{\mathsf{s}})]],\mathsf{c})}\mathsf{d}' = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{s}})\ \parallel\ \mathsf{tell}(\tilde{\mathsf{t}}=\tilde{\mathsf{s}})]],\mathsf{c})}\mathsf{d_1}.$$

By (46), $\mathsf{w_t}(\mathsf{C_l}[\mathsf{C}[\mathsf{B} \parallel \mathsf{tell}(\tilde{\mathsf{t}} = \tilde{\mathsf{s}})]], \mathsf{c}, \mathsf{d}') \leq \mathsf{n}$.

Furthermore, by hypothesis and construction, $\mathit{Var}(\mathsf{c}', \mathsf{d}') \cap \mathit{Var}(\tilde{\mathsf{r}}) \subseteq \mathit{Var}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]], \mathsf{c})$ and, without loss of generality, we can assume that $\mathit{Var}(\mathsf{d_1}) \cap \mathit{Var}(\tilde{\mathsf{r}}) \subseteq \mathit{Var}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{t}})]], \mathsf{c})$.

Then, by (47) and since $Var(\mathsf{C_l[C[p(\tilde{t})]], c}) \subseteq Var(\mathsf{C_l[C[p(\tilde{s}) \parallel tell(\tilde{t} = \tilde{s})]], c})$, we have that $\exists_{-Var(\mathsf{C_l[q(\tilde{r})],c})}\mathsf{d}' = \exists_{-Var(\mathsf{C_l[q(\tilde{r})],c})}\mathsf{c}'$, and this completes the proof.

Case 2. cl' is the result of a tell elimination or introduction. The proof is analogous to that for Case 2 of Proposition 4.5, and is omitted.

Case 3. cl' is the result of a backward instantiation. Let cl be the corresponding declaration in $\mathsf{D_i}$. The situation is then as follows:

—cl: $\mathsf{q(\tilde{r}) \leftarrow C[p(\tilde{t})]}$,
—cl′: $\mathsf{q(\tilde{r}) \leftarrow C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})]}$,

where f: $\mathsf{p(\tilde{s}) \leftarrow tell(b) \parallel H \in D_i}$ has no variable in common with cl (the case cl': $\mathsf{q(\tilde{r}) \leftarrow C[p(\tilde{t}) \parallel tell(\tilde{t}=\tilde{s})]}$ is analogous, and hence omitted). By the hypothesis, $Var(\mathsf{C[p(\tilde{t})], C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t}=\tilde{s})]}) \cap Var(\mathsf{C_l, c}) \subseteq Var(\tilde{\mathsf{r}})$, $Var(\mathsf{c}'), \cap Var(\tilde{\mathsf{r}}) \subseteq Var(\mathsf{C_l[C[p(\tilde{t})]], c})$, and there exists n such that $\mathsf{w_t(C_l[C[p(\tilde{t})]], c, c') = n}$. Then, $Var(\mathsf{C_l[C[p(\tilde{t})]], c}) \cap Var(\tilde{\mathsf{s}}) = \emptyset$ and, without loss of generality, we can assume that $Var(\mathsf{H}) \cap Var(\mathsf{C_l, c}) = \emptyset$.

Moreover, by the definition of the transformation sequence, there exists a declaration $\mathsf{p(\tilde{s}) \leftarrow B_0 \in D_0}$, and then, by Lemma A.3, there exists a constraint $\mathsf{d_1}$ such that

$$\mathsf{w_t(C_l[C[p(\tilde{s}) \parallel tell(\tilde{t} = \tilde{s})]], c, d_1) \leq w_t(C_l[C[p(\tilde{t})]], c, c') = n} \tag{48}$$

and

$$\exists_{-Var(\mathsf{C_l[C[p(\tilde{t})]],c})}\mathsf{d_1} = \exists_{-Var(\mathsf{C_l[C[p(\tilde{t})]],c})}\mathsf{c}'. \tag{49}$$

Using the hypothesis on the variables and since f is renamed separately from $Var(\tilde{\mathsf{r}})$, we have that

$$Var(\mathsf{tell(b) \parallel H}) \cap Var(\mathsf{C_l[C[\ \parallel tell(\tilde{t} = \tilde{s})]], c}) \subseteq Var(\tilde{\mathsf{s}}).$$

Then, from point (1) of the Lemma (assumed as a hypothesis) and (48), it follows that there exists a constraint $\mathsf{d_2}$ such that

$$\mathsf{w_t(C_l[C[tell(b) \parallel H \parallel tell(\tilde{t} = \tilde{s})]], c, d_2) \leq w_t(C_l[C[p(\tilde{s}) \parallel tell(\tilde{t} = \tilde{s})]], c, d_1) \leq n} \tag{50}$$

and

$$\exists_{-Var(\mathsf{C_l[C[p(\tilde{s}) \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{d_2} = \exists_{-Var(\mathsf{C_l[C[p(\tilde{s}) \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{d_1} \tag{51}$$

hold. By the definition of weight, we can assume that $Var(\mathsf{d_1}) \subseteq Var(\mathsf{C_l[C[p(\tilde{s}) \parallel tell(\tilde{t}=\tilde{s})]], c})$, and hence, we have that $Var(\mathsf{b}) \cap Var(\mathsf{C_l[C[p(\tilde{s}) \parallel tell(\tilde{t} = \tilde{s})]], c, d_1}) \subseteq Var(\tilde{\mathsf{s}})$.

We now have two cases:

(1) $\mathcal{D} \models \exists_{-Var(\tilde{\mathsf{s}})}\mathsf{d_1} \rightarrow \exists_{-Var(\tilde{\mathsf{s}})}\mathsf{b}$. In this case, by (48), there exists a derivation

$$\xi = \langle \mathsf{D_0.C_l[C[p(\tilde{s}) \parallel tell(\tilde{t} = \tilde{s})]], c} \rangle \rightarrow^* \langle \mathsf{D_0.B_F, c_F} \rangle$$

such that $\mathsf{m(B_F, c_F) = t}$, $wh(\xi) \leq \mathsf{n}$, and

$$\exists_{-Var(\mathsf{C_l[C[p(\tilde{s}) \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{c_F} = \exists_{-Var(\mathsf{C_l[C[p(\tilde{s}) \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{d_1}.$$

By the hypothesis on the variables, we can build a derivation

$$\chi = \langle \mathsf{D_0.C_l[C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})]], c} \rangle \rightarrow^* \langle \mathsf{D_0.B'_F, d_3} \rangle,$$

which performs exactly the same steps of ξ, plus possibly a tell action such that $wh(\chi) \leq \mathsf{n}$, $\mathsf{m}(\mathsf{B}'_\mathsf{F}, \mathsf{d}_3) = \mathsf{m}(\mathsf{B}_\mathsf{F}, \mathsf{c}_\mathsf{F})$, and

$$\exists_{-Var(\mathsf{C_I[C[p(\tilde{s}) \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{d}_3 = \exists_{-Var(\mathsf{C_I[C[p(\tilde{s}) \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{d}_1. \tag{52}$$

Let $\mathsf{d}' = \exists_{-Var(\mathsf{C_I[C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{d}_3$. By the previous result and by the definition of weight, $\mathsf{w_t(C_I[C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})]], c, d') \leq n}$.

Moreover, by hypothesis, $Var(\mathsf{c}', \mathsf{d}') \cap Var(\tilde{\mathsf{r}}) \subseteq Var(\mathsf{C_I[C[p(\tilde{t})]], c})$, and we can assume, without loss of generality, that $Var(\mathsf{d}_1, \mathsf{d}_2) \cap Var(\tilde{\mathsf{r}}) \subseteq Var(\mathsf{C_I[C[p(\tilde{t})]], c})$. Then, by (49), (52), and by the definition of d', it follows that $\exists_{-Var(\mathsf{C_I[q(\tilde{r})],c})}\mathsf{d}' = \exists_{-Var(\mathsf{C_I[q(\tilde{r})],c})}\mathsf{c}'$, and the thesis holds.

(2) $\mathcal{D} \not\models \exists_{-Var(\tilde{\mathsf{s}})}\mathsf{d}_1 \rightarrow \exists_{-Var(\tilde{\mathsf{s}})}\mathsf{b}$. In this case, by (51), $\mathcal{D} \not\models \exists_{-Var(\tilde{\mathsf{s}})}\mathsf{d}_2 \rightarrow \exists_{-Var(\tilde{\mathsf{s}})}\mathsf{b}$. By (50), this means that there exists a derivation

$$\xi = \langle \mathsf{D_0.C_I[C[tell(b) \parallel H \parallel tell(\tilde{t} = \tilde{s})]], c} \rangle \rightarrow^* \langle \mathsf{D_0.B_F, c_F} \rangle \not\rightarrow$$

such that $\mathsf{tell(b) \parallel H \parallel tell(\tilde{t} = \tilde{s})}$ is not evaluated in ξ, $\mathsf{m(B_F, c_F) = t}$, $wh(\xi) \leq \mathsf{n}$ and $\exists_{-Var(\mathsf{C_I[C[tell(b) \parallel H \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{c_F} = \exists_{-Var(\mathsf{C_I[C[tell(b) \parallel H \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{d}_2$. By definition, we can construct another derivation

$$\chi = \langle \mathsf{D_0.C_I[C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})]], c} \rangle \rightarrow^* \langle \mathsf{D_0.B'_F, c_F} \rangle \not\rightarrow$$

that performs exactly the same steps of ξ (and therefore $wh(\chi) \leq \mathsf{n}$), and such that $\mathsf{m(B_F, c_F) = m(B'_F, c_F)}$. Let $\mathsf{d}' = \exists_{-Var(\mathsf{C_I[C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{c_F}$. By the definition of derivation,

$$Var(\mathsf{c_F}) \cap Var(\mathsf{C_I[C[tell(b) \parallel H \parallel tell(\tilde{t} = \tilde{s})]], c}) \subseteq Var(\mathsf{C_I, C, c}),$$

and therefore $\exists_{-Var(\mathsf{C_I[C[tell(b) \parallel H \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{d}' = \exists_{-Var(\mathsf{C_I[C[tell(b) \parallel H \parallel tell(\tilde{t}=\tilde{s})]],c})}\mathsf{d}_2$. The remainder of the proof is now analogous to that of the previous case.

Case 4. Either cl' is the result of an ask simplification or cl'' is the result of a tell simplification. The proof is analogous to that for Case 4 of Proposition 4.5, and hence is omitted.

Case 5. cl' is the result of a branch elimination or of a conservative ask elimination. The proof is straightforward by noting that: (a) according to Definition 4.1, we also consider inconsistent stores resulting from nonterminated computations; (b) an ask action of the form ask(true) always succeeds; (c) if we delete an ask(true) action, we obtain a derivation whose weight is smaller.

Case 6. cl' is the result of a distribution.
Let

—cl: $\mathsf{q(\tilde{r}) \leftarrow C[H \parallel \sum_{j=1}^{n} ask(c_j) \rightarrow B_j] \in D_i}$,
—cl′: $\mathsf{q(\tilde{r}) \leftarrow C[\sum_{j=1}^{n} ask(c_j) \rightarrow (H \parallel B_j)] \in D_{i+1}}$,

where $\mathsf{e = pc(C[\ ])}$ and for every constraint e' such that $Var(\mathsf{e}') \cap Var(\mathsf{cl}) \subseteq Var(\mathsf{q(\tilde{r}), C})$, if $\langle \mathsf{D.H, e' \wedge e} \rangle$ is productive, then both the following conditions hold:

—there exists at least one $j \in [1, n]$ such that $\mathcal{D} \models (\mathsf{e}' \wedge \mathsf{e}) \to \mathsf{c_j}$
—for each $j \in [1, n]$, either $\mathcal{D} \models (\mathsf{e}' \wedge \mathsf{e}) \to \mathsf{c_j}$ or $\mathcal{D} \models (\mathsf{e}' \wedge \mathsf{e}) \to \neg\mathsf{c_j}$.

We prove that, for any derivation

$$\xi = \left\langle \mathsf{D_0.C_l}\left[\mathsf{C}\left[\mathsf{H} \parallel \sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \to \mathsf{B_j}\right]\right], \mathsf{c} \right\rangle \to^* \langle \mathsf{D_0.B, d} \rangle$$

with $\mathsf{m(B, d)} \in \{\mathsf{ss, dd, ff}\}$, there exists a derivation

$$\xi' = \left\langle \mathsf{D_0.C_l}\left[\mathsf{C}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \to (\mathsf{H} \parallel \mathsf{B_j})\right]\right], \mathsf{c} \right\rangle \to^* \langle \mathsf{D_0.B', d'} \rangle$$

such that

$$\exists_{-Var(\mathsf{C_l[C[H} \,\parallel\, \sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)\to B_j]],c)}} \mathsf{d}' = \exists_{-Var(\mathsf{C_l[C[H} \,\parallel\, \sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)\to B_j]],c)}} \mathsf{d},$$

where $wh(\xi') \le wh(\xi)$ and $\mathsf{m(B', d')} = \mathsf{m(B, d)}$. This, together with the definition of weight, implies the thesis.

If $\mathsf{H} \parallel \sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \to \mathsf{B_j}$ is not evaluated in ξ, then the proof is immediate. Otherwise, we have to distinguish two cases:

(1) There exists an $h \in [1, n]$ such that

$$\begin{aligned}\xi \;=\; & \left\langle \mathsf{D_0.C_l}\left[\mathsf{C}\left[\mathsf{H} \parallel \sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \to \mathsf{B_j}\right]\right], \mathsf{c} \right\rangle \to^* \left\langle \mathsf{D_0.C_m}\left[\mathsf{H} \parallel \sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \right.\right.\\ & \left.\left.\to B_j\right], d_m \right\rangle \to \langle \mathsf{D_0.C_m[H \parallel B_h], d_m} \rangle \to^* \langle \mathsf{D_0.B, d} \rangle\end{aligned}$$

and $\mathcal{D} \models \mathsf{d_m} \to \mathsf{c_h}$. In this case we can construct the derivation

$$\begin{aligned}\chi \;=\; & \left\langle \mathsf{D_0.C_l}\left[\mathsf{C}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \to (\mathsf{H} \parallel \mathsf{B_j})\right]\right], \mathsf{c} \right\rangle\\ & \to^* \left\langle \mathsf{D_0.C_m}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \to (\mathsf{H} \parallel \mathsf{B_j})\right], \mathsf{d_m} \right\rangle\\ & \to \langle \mathsf{D_0.C_m[H \parallel B_h], d_m} \rangle \to^* \langle \mathsf{D_0.B, d} \rangle,\end{aligned}$$

which performs exactly the same steps of ξ, and then the thesis holds.

(2) ξ is of the form

$$\begin{aligned}\xi \;=\; & \left\langle \mathsf{D_0.C_l}\left[\mathsf{C}\left[\mathsf{H} \parallel \sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \to \mathsf{B_j}\right]\right], \mathsf{c} \right\rangle \to^* \left\langle \mathsf{D_0.C_m}\left[\mathsf{H} \parallel \sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \right.\right.\\ & \left.\left.\to B_j\right], d_m \right\rangle \to \left\langle \mathsf{D_0.C_m}\left[\mathsf{H}' \parallel \sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask(c_j)} \to \mathsf{B_j}\right], \mathsf{d_{m+1}} \right\rangle \to^* \langle \mathsf{D_0.B, d} \rangle,\end{aligned}$$

By Lemma A.1 and by definition of pc, we can construct another derivation,

$$\chi = \left\langle D_0.C_l\left[C\left[H \parallel \sum_{j=1}^{n} ask(c_j) \to B_j\right]\right], c\right\rangle \to^* \left\langle D_0.C_m\left[H \parallel \sum_{j=1}^{n} ask(c_j)\right.\right.$$
$$\left.\left.\to B_j\right], d_m\right\rangle \to^* \left\langle D_0.C_k\left[H \parallel \sum_{j=1}^{n} ask(c_j) \to B_j\right], d_k\right\rangle \to^* \langle D_0.B, d\rangle,$$

which performs the same steps of ξ (possibly in a different order) and such that the agent H is not evaluated in the first k steps, where $Var(d_k) \cap Var(cl) \subseteq Var(q(\tilde{r}), C)$ and $\mathcal{D} \models d_k \to e(= pc(C[\ ]))$. Let $\chi_1 = \langle D_0.C_k[H \parallel \sum_{j=1}^{n} ask(c_j) \to B_j], d_k\rangle \to^* \langle D_0.B, d\rangle$. Now, if $\langle D.H, d_k\rangle$ is not productive, the proof is analogous to the one for Case 6 of Proposition 4.5, and hence is omitted. Then assume that $\langle Do, H, d_k\rangle$ is productive. By definition of distribution, there exists at least one $j \in [1, n]$ such that $\mathcal{D} \models d_k \to c_j$, and for each $j \in [1, n]$, either $\mathcal{D} \models d_k \to c_j$ or $\mathcal{D} \models d_k \to \neg c_j$. Then, by definition, there exists a derivation $\xi_1 = \langle D_0.C_k[\sum_{j=1}^{n} ask(c_j) \to (H \parallel B_j)], d_k\rangle \to^* \langle D_0.B, d\rangle$, which performs the same steps of χ_1 (possibly in a different order).

Therefore, there exists a derivation

$$\xi' = \left\langle D_0.C_l\left[C\left[\sum_{j=1}^{n} ask(c_j) \to (H \parallel B_j)\right]\right], c\right\rangle \to^* \left\langle D_0.C_m\left[\sum_{j=1}^{n} ask(c_j)\right.\right.$$
$$\left.\left.\to (H \parallel B_j)\right], d_m\right\rangle \to^* \left\langle D_0.C_k\left[\sum_{j=1}^{n} ask(c_j) \to (H \parallel B_j)\right], d_k\right\rangle \to^* \langle D_0.B, d\rangle,$$

which performs the same steps of χ (in a different order). By construction, $wh(\xi') = wh(\chi) = wh(\xi)$, and the thesis holds.

Case 7. cl′ is the result of a folding.
Let

—cl: $q(\tilde{r}) \leftarrow C[B]$ be the folded declaration ($\in D_i$),
—f: $p(\tilde{X}) \leftarrow B$ be the folding declaration ($\in D_0$),
—cl′: $q(\tilde{r}) \leftarrow C[p(\tilde{X})]$ be the result of the folding operation ($\in D_{i+1}$),

where, by hypothesis, $Var(cl) \cap Var(\tilde{X}) \subseteq Var(B)$, $Var(B) \cap Var(\tilde{r}, C) \subseteq Var(\tilde{X})$, $Var(C[B], C[p(\tilde{X})]) \cap Var(C_l, c) \subseteq Var(\tilde{r})$, $Var(c') \cap Var(\tilde{r}) \subseteq Var(C_l[C[B]], c)$, and there exists n such that $w_t(C_l[C[B]], c, c') = n$. Then,

$$Var(B) \cap Var(C_l[C[\ ]], c) \subseteq Var(B) \cap Var(\tilde{r}, C) \subseteq Var(\tilde{X}) \tag{53}$$

and

$$Var(c') \cap Var(\tilde{r}) \subseteq Var(C_l[C[B]], c) \cap Var(\tilde{r}) \subseteq Var(C_l[C[p(\tilde{X})]], c) \tag{54}$$

hold. Moreover, we can assume, without loss of generality, that $Var(c') \cap Var(\tilde{X}) \subseteq Var(C_l[C[B]], c)$.

Since $f \in D_0$, from (53) and point (2) of Lemma A.4, it follows that there exists a constraint d′ such that $w_t(C_l[C[p(\tilde{X})]], c, d') \leq w_t(C_l[C[B]], c, c')$ and

$$\exists_{-Var(C_l[C[p(\tilde{X})]], c)} d' = \exists_{-Var(C_l[C[p(\tilde{X})]], c)} c'. \tag{55}$$

We can assume, without loss of generality, that $\mathit{Var}(\mathsf{d}') \subseteq \mathit{Var}(\mathsf{C_l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{X}})]], \mathsf{c})$. Then by using (54) and (55), we obtain that $\exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{d}' = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{c}'$, which concludes the proof of **(a)**.

(b) Assume that parts 1 and 2 of this lemma hold for $i \geq 0$. We prove that 1 holds for $i + 1 > 0$.

Let $\mathsf{cl}\colon \mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{H} \in \mathsf{D_{i+1}}$, and let $\overline{\mathsf{cl}}\colon \mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \overline{\mathsf{H}}$ be the corresponding declaration in $\mathsf{D_i}$. Moreover, let $\mathsf{C_l}[\ ]$ be a context, c a satisfiable constraint, and let c' be a constraint such that $\mathit{Var}(\mathsf{H}) \cap \mathit{Var}(\mathsf{C_l}, \mathsf{c}) \subseteq \mathit{Var}(\tilde{\mathsf{r}})$, and $\mathsf{w_t}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})], \mathsf{c}, \mathsf{c}')$ is defined. Without loss of generality, we can assume that $\mathit{Var}(\overline{\mathsf{H}}) \cap \mathit{Var}(\mathsf{C_l}, \mathsf{c}) \subseteq \mathit{Var}(\tilde{\mathsf{r}})$. Then, since by inductive hypothesis, part (1) holds for i, there exists a constraint $\mathsf{d_1}$ such that $\mathit{Var}(\mathsf{d_1}) \subseteq \mathit{Var}(\mathsf{C_l}[\overline{\mathsf{H}}], \mathsf{c})$,

$$\mathsf{w_t}(\mathsf{C_l}[\overline{\mathsf{H}}], \mathsf{c}, \mathsf{d_1}) \leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})], \mathsf{c}, \mathsf{c}') \text{ and } \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{d_1} = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{c}'. \quad (56)$$

Since, by inductive hypothesis, part (2) holds for i, there exists a constraint d' such that $\mathit{Var}(\mathsf{d}') \subseteq \mathit{Var}(\mathsf{C_l}[\mathsf{H}], \mathsf{c})$, $\mathsf{w_t}(\mathsf{C_l}[\mathsf{H}], \mathsf{c}, \mathsf{d}') \leq \mathsf{w_t}(\mathsf{C_l}[\overline{\mathsf{H}}], \mathsf{c}, \mathsf{d_1})$, and $\exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{d}' = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{d_1}$.

By (56), we obtain $\mathsf{w_t}(\mathsf{C_l}[\mathsf{H}], \mathsf{c}, \mathsf{d}') \leq \mathsf{w_t}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})], \mathsf{c}, \mathsf{c}')$ and

$$\exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{d}' = \exists_{-\mathit{Var}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{r}})],\mathsf{c})}\mathsf{c}'$$

and then the thesis holds. □

Lemma A.6. *Let $0 \leq i \leq n$, $\mathsf{c_1}, \mathsf{c_m}$ be satisfiable constraints, $\mathsf{c_k}$ a constraint, and assume that there exists a derivation ξ: $\langle \mathsf{D_i.A_1}, \mathsf{c_1}\rangle \rightarrow^* \langle \mathsf{D_i.A_m}, \mathsf{c_m}\rangle \rightarrow^* \langle \mathsf{D_i.A_k}, \mathsf{c_k}\rangle$ such that*

(i) *in the first $m - 1$ steps of ξ, rule* **R2** *is only used for evaluating agents of the form $\mathsf{ask(c)} \rightarrow \mathsf{B}$;*
(ii) $\mathsf{w_t}(\mathsf{A_1}, \mathsf{c_1}, \mathsf{c_k})$ *is defined (for* $\mathsf{t} = \mathsf{m}(\mathsf{A_k}, \mathsf{c_k}) \in \{\mathsf{ss}, \mathsf{dd}, \mathsf{ff}\}$).

Then there exists a constraint c' such that $\mathit{Var}(\mathsf{c}') \subseteq \mathit{Var}(\mathsf{A_m}, \mathsf{c_m})$, $\exists_{-\mathit{Var}(\mathsf{A_1},\mathsf{c_1})}\mathsf{c_k} = \exists_{-\mathit{Var}(\mathsf{A_1},\mathsf{c_1})}\mathsf{c}'$, and $\mathsf{w_t}(\mathsf{A_m}, \mathsf{c_m}, \mathsf{c}') \leq \mathsf{w_t}(\mathsf{A_1}, \mathsf{c_1}, \mathsf{c_k})$.

Proof. We prove the thesis for one derivation step. Then the proof of the lemma follows by using a straightforward inductive argument. Assume that $\mathsf{c_1}, \mathsf{c_2}$ are satisfiable constraints, $\mathsf{c_k}$ is a constraint, and that there exists a derivation

$$\langle \mathsf{D_i.A_1}, \mathsf{c_1}\rangle \rightarrow \langle \mathsf{D_i.A_2}, \mathsf{c_2}\rangle \rightarrow^* \langle \mathsf{D_i.A_k}, \mathsf{c_k}\rangle$$

such that $\mathsf{m}(\mathsf{A_k}, \mathsf{c_k}) \in \{\mathsf{ss}, \mathsf{dd}, \mathsf{ff}\}$, and the first step can use rule **R2** only for evaluating agents of the form $\mathsf{ask(c)} \rightarrow \mathsf{B}$. By the definition, of derivation we have $\mathsf{A_1} = \mathsf{C_1}[\mathsf{A}]$, where $\mathsf{C_1}[\ ]$ is not a guarding context. We now have three cases:

(1) $\mathsf{A} = \mathsf{tell(c)}$. In this case,

$$\langle \mathsf{D_i.C_1}[\mathsf{tell(c)}], \mathsf{c_1}\rangle \rightarrow \langle \mathsf{D_i.C_1}[\mathsf{Stop}], \mathsf{c_1} \wedge \mathsf{c}\rangle \rightarrow^* \langle \mathsf{D_i.A_k}, \mathsf{c_k}\rangle.$$

Since $\mathsf{C_1}[\ ]$ is not a guarding context, the definition of weight implies that

$$\mathsf{w_t}(\mathsf{C_1}[\mathsf{Stop}], \mathsf{c_1} \wedge \mathsf{c}, \exists_{-\mathit{Var}(\mathsf{C_1}[\mathsf{Stop}],\mathsf{c_1}\wedge\mathsf{c})}\mathsf{c_k}) = \mathsf{w_t}(\mathsf{C_1}[\mathsf{tell(c)}], \mathsf{c_1}, \mathsf{c_k})$$

where $\mathsf{t} = \mathsf{m}(\mathsf{A_k}, \mathsf{c_k})$. Then the thesis holds.

(2) $\mathsf{A} = \mathsf{q}(\tilde{\mathsf{v}})$, and there exists a declaration $\mathsf{cl}\colon \mathsf{q}(\tilde{\mathsf{r}}) \leftarrow \mathsf{B} \in \mathsf{D_i}$. In this case,

$$\langle \mathsf{D_i.C_1[q(\tilde{v})], c_1} \rangle \rightarrow \langle \mathsf{D_i.C_1[B} \parallel \mathsf{tell(\tilde{v} = \tilde{r})], c_1} \rangle \rightarrow^* \langle \mathsf{D_i.A_k, c_k} \rangle.$$

From the definition of derivation it follows that $Var(\mathsf{C_1[q(\tilde{v})], c_1}) \cap Var(\mathsf{q(\tilde{r})}) = \emptyset$. Furthermore, by definition of a transformation sequence, there exists a declaration $\mathsf{q(\tilde{r}) \leftarrow H \in D_0}$. Since $\mathsf{w_t(C_1[q(\tilde{v})], c_1, c_k)}$ is defined by hypothesis (where $\mathsf{t = m(A_k, c_k)}$), from Lemma A.3 it follows that there exists a constraint $\mathsf{d'}$ such that $\mathsf{w_t(C_1[q(\tilde{r})} \parallel \mathsf{tell(\tilde{v} = \tilde{r})], c_1, d')} \leq \mathsf{w_t(C_1[q(\tilde{v})], c_1, c_k)}$ and $\exists_{-Var(\mathsf{C_1[q(\tilde{v})],c_1})}\mathsf{d'} = \exists_{-Var(\mathsf{C_1[q(\tilde{v})],c_1})}\mathsf{c_k}$.

From the definition of derivation it follows that $Var(\mathsf{B}) \cap Var(\mathsf{C_1[} \parallel \mathsf{tell(\tilde{v} = \tilde{r})], c_1}) \subseteq Var(\mathsf{\tilde{r}})$. Part A.5 of Lemma A.5 implies that there exists a constraint $\mathsf{c'}$ such that $Var(\mathsf{c'}) \subseteq Var(\mathsf{C_1[B} \parallel \mathsf{tell(\tilde{v} = \tilde{r})], c_1})$, $\mathsf{w_t(C_1[B} \parallel \mathsf{tell(\tilde{v} = \tilde{r})], c_1, c') \leq w_t(C_1[q(\tilde{r})} \parallel \mathsf{tell(\tilde{v} = \tilde{r})], c_1, d')}$, and

$$\exists_{-Var(\mathsf{C_1[q(\tilde{r})} \parallel \mathsf{tell(\tilde{v}=\tilde{r})],c_1})}\mathsf{c'} = \exists_{-Var(\mathsf{C_1[q(\tilde{r})} \parallel \mathsf{tell(\tilde{v}=\tilde{r})],c_1})}\mathsf{d'}$$

These results, together with the inclusion $Var(\mathsf{C_1[q(\tilde{v})], c_1}) \subseteq Var(\mathsf{C_1[q(\tilde{r})} \parallel \mathsf{tell(\tilde{v} = \tilde{r})], c_1})$, imply that $\mathsf{w_t(C_1[B} \parallel \mathsf{tell(\tilde{v} = \tilde{r})], c_1, c') \leq w_t(C_1[q(\tilde{v})], c_1, c_k)}$ and

$$\exists_{-Var(\mathsf{C_1[q(\tilde{v})],c_1})}\mathsf{c'} = \exists_{-Var(\mathsf{C_1[q(\tilde{v})],c_1})}\mathsf{c_k},$$

thus concluding the proof for this case.

(3) $\mathsf{A = ask(c) \rightarrow B}$ and $\mathcal{D} \models \mathsf{c_1 \rightarrow c}$. In this case,

$$\langle \mathsf{D_i.C_1[ask(c) \rightarrow B], c_1} \rangle \rightarrow \langle \mathsf{D_i.C_1[B], c_1} \rangle \rightarrow^* \langle \mathsf{D_i.A_k, c_k} \rangle.$$

Since $\mathsf{C_1[\ ]}$ is not a guarding context and $\mathcal{D} \models \mathsf{c_1 \rightarrow c}$, we obtain

$$\mathsf{w_t(C_1[B], c_1, \exists_{-\mathit{Var}(C_1[B],c_1)}c_k) \leq w_t(C_1[ask(c) \rightarrow B], c_1, c_k)}$$

where $\mathsf{t = m(A_k, c_k)}$, which concludes the proof. □

We need one last lemma.

LEMMA A.7. *Let* c *be a satisfiable constraint,* A *be the agent* $\mathsf{A_1} \parallel \ldots \parallel \mathsf{A_l}$, *where for any* $j \in [1, l]$, *either* $\mathsf{A_j}$ *is a choice agent or* $\mathsf{A_j = Stop}$, *and assume there exists a split derivation* ν *in* $\mathsf{D_0}$,

$$\nu = \langle \mathsf{D_0.A, c} \rangle \rightarrow \langle \mathsf{D_0.A', c'} \rangle \rightarrow^* \langle \mathsf{D_0.B, d} \rangle,$$

where $\mathsf{m(B, d) \in \{ss, dd, ff\}}$. *Then* $\langle \mathsf{D_i.A, c} \rangle \rightarrow \langle \mathsf{D_0.A', c'} \rangle \rightarrow^* \langle \mathsf{D_0.B, d} \rangle$ *is a split derivation in* $\mathsf{D_i \cup D_0}$.

PROOF. The proof is straightforward, by observing that by the hypothesis on A the first step of ν uses the rule **R2** (where such a step exists), and therefore, by definition of split derivation, $\mathsf{w_t(A, c, d) > w_t(A', c', d)}$ where $\mathsf{t = m(B, d)}$. Then, by definition, $\langle \mathsf{D_i.A, c} \rangle \rightarrow \langle \mathsf{D_0.A', c'} \rangle \rightarrow^* \langle \mathsf{D_0.B, d} \rangle$ is a split derivation in $\mathsf{D_i \cup D_0}$. □

We can now prove our main theorem.

THEOREM 4.13 (Total Correctness). *Let* $\mathsf{D_0, \ldots, D_n}$ *be a transformation sequence. Then, for any agent* A,

—$\mathcal{O}(\mathsf{D_0.A}) = \mathcal{O}(\mathsf{D_n.A})$.

PROOF. The proof proceeds by showing simultaneously, by induction on i, for $i \in [0, n]$:

(1) for any agent A, $\mathcal{O}(\mathsf{D_0.A}) = \mathcal{O}(\mathsf{D_i.A})$;

(2) $\mathsf{D_i}$ is weight-complete.

Base case. We just need to prove that $\mathsf{D_0}$ is weight-complete. Assume that there exists a derivation $\langle \mathsf{D_0.A, c_I} \rangle \rightarrow^* \langle \mathsf{D_0.B, c_F} \rangle$, where $\mathsf{c_I}$ is a satisfiable constraint and $\mathsf{m(B, c_F)} \in \{\mathsf{ss, dd, ff}\}$. Then there exists a derivation ξ: $\langle \mathsf{D_0.A, c_1} \rangle \rightarrow^* \langle \mathsf{D_0.B', c'_F} \rangle$ such that $\mathsf{m(B', c'_F)} = \mathsf{m(B, c_F)}$, whose weight is minimal and where $\exists_{-\mathit{Var}(\mathsf{A,c_I})}\mathsf{c'_F} = \exists_{-\mathit{Var}(\mathsf{A,c_I})}\mathsf{c_F}$. It follows from Definition 4.7 that ξ is a split derivation.

Induction step. By the inductive hypothesis for any agent A, $\mathcal{O}(\mathsf{D_0.A}) = \mathcal{O}(\mathsf{D_{i-1}.A})$, and $\mathsf{D_{i-1}}$ is weight-complete. From Propositions 4.5 and 4.9 it follows that if $\mathsf{D_i}$ is weight-complete, then, for any agent A, $\mathcal{O}(\mathsf{D_0.A}) = \mathcal{O}(\mathsf{D_i.A})$. So, in order to prove parts 1 and 2, we only have to show that $\mathsf{D_i}$ is weight-complete.

Assume then that there exists a derivation $\langle \mathsf{D_0.A, c_I} \rangle \rightarrow^* \langle \mathsf{D_0.B, c_F} \rangle$ such that $\mathsf{c_I}$ is a satisfiable constraint and $\mathsf{m(B, c_F)} \in \{\mathsf{ss, dd, ff}\}$. From the inductive hypothesis it follows that there exists a split derivation

$$\chi = \langle \mathsf{D_{i-1}.A, c_I} \rangle \rightarrow^* \langle \mathsf{D_{i-1}.A_m, c_m} \rangle \rightarrow \langle \mathsf{D_0.A_{m+1}, c_{m+1}} \rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F} \rangle,$$

where

$$\exists_{-\mathit{Var}(\mathsf{A,c_I})}\mathsf{c''_F} = \exists_{-\mathit{Var}(\mathsf{A,c_I})}\mathsf{c_F} \text{ and } \mathsf{m(B'', c''_F)} = \mathsf{m(B, c_F)}. \tag{57}$$

Let $\mathsf{d} \in \mathsf{D_{i-1}} \backslash \mathsf{D_i}$ be the modified clause in the transformation step from $\mathsf{D_{i-1}}$ to $\mathsf{D_i}$.

If in the first m steps of χ there is no procedure call that uses d, then clearly there exists a split derivation ξ in $\mathsf{D_i} \cup \mathsf{D_0}$,

$$\xi = \langle \mathsf{D_i.A, c_I} \rangle \rightarrow^* \langle \mathsf{D_i.A_m, c_m} \rangle \rightarrow \langle \mathsf{D_0.A_{m+1}, c_{m+1}} \rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F} \rangle,$$

which performs the same steps of χ, and then the thesis holds.

Otherwise, assume without loss of generality that **R4** is the rule used in the first step of derivation χ and that d is the clause employed in the first step of χ. We also assume that the declaration d is used only once in χ, since the extension to the general case is immediate.

We have to distinguish various cases according to what happens to clause d when moving from $\mathsf{D_{i-1}}$ to $\mathsf{D_i}$.

Case 1. d is unfolded.

Let $\mathsf{d'}$ be the corresponding declaration in $\mathsf{D_i}$. The situation is the following:

—d: $\mathsf{q(\tilde{r}) \leftarrow C[p(\tilde{t})]} \in \mathsf{D_{i-1}}$,

—u: $\mathsf{p(\tilde{s}) \leftarrow H} \in \mathsf{D_{i-1}}$, and ,

—$\mathsf{d'}$: $\mathsf{q(\tilde{r}) \leftarrow C[H \parallel tell(\tilde{t} = \tilde{s})]} \in \mathsf{D_i}$, where d and u are assumed to be renamed apart. By the definition of split derivation, χ has the form

$$\begin{aligned}&\langle \mathsf{D_{i-1}.C_l[q(\tilde{v})], c_l}\rangle \rightarrow \langle \mathsf{D_{i-1}.C_l[C[p(\tilde{t})] \parallel tell(\tilde{v} = \tilde{r})], c_l}\rangle \rightarrow^* \langle \mathsf{D_{i-1}.A_m, c_m}\rangle\\ &\quad\rightarrow \langle \mathsf{D_0.A_{m+1}, c_{m+1}}\rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F}\rangle.\end{aligned}$$

Without loss of generality, we can assume that $Var(\chi) \cap Var(\mathsf{u}) \neq \emptyset$ if and only if $\mathsf{p(\tilde{t})}$ is evaluated in the first m steps of χ, in which case u is used to evaluate it. We have to distinguish two cases.

(1) There exists $k < m$ such that the k-th derivation step of χ is the procedure call $\mathsf{p(\tilde{t})}$. In this case, χ has the form

$$\begin{aligned}&\langle \mathsf{D_{i-1}.C_l[q(\tilde{v})], c_l}\rangle \rightarrow \langle \mathsf{D_{i-1}.C_l[C[p(\tilde{t})] \parallel tell(\tilde{v} = \tilde{r})], c_l}\rangle \rightarrow^* \langle \mathsf{D_{i-1}.C_k[p(\tilde{t})], c_k}\rangle\\ &\quad\rightarrow \langle \mathsf{D_{i-1}.C_k[H \parallel tell(\tilde{t} = \tilde{s})], c_k}\rangle \rightarrow^* \langle \mathsf{D_{i-1}.A_m, c_m}\rangle \rightarrow \langle \mathsf{D_0.A_{m+1}, c_{m+1}}\rangle\\ &\quad\rightarrow^* \langle D_0.B'', c''_F\rangle.\end{aligned}$$

Then there exists a corresponding derivation in $\mathsf{D_i} \cup \mathsf{D_0}$:

$$\begin{aligned}\xi = &\langle \mathsf{D_i.C_l[q(\tilde{v})], c_l}\rangle \rightarrow \langle \mathsf{D_i.C_l[C[H \parallel tell(\tilde{t} = \tilde{s})] \parallel tell(\tilde{v} = \tilde{r})], c_l}\rangle\\ &\rightarrow^* \langle \mathsf{D_i.C_k[H \parallel tell(\tilde{t} = \tilde{s})], c_k}\rangle\\ &\rightarrow^* \langle \mathsf{D_i.A_m, c_m}\rangle \rightarrow \langle \mathsf{D_0.A_{m+1}, c_{m+1}}\rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F}\rangle,\end{aligned}$$

which performs exactly the same steps of χ except for a procedure call to $\mathsf{p(\tilde{t})}$. In this case, the proof follows by observing that, since by the inductive hypothesis χ is a split derivation, the same holds for ξ.

(2) There is no procedure call to $\mathsf{p(\tilde{t})}$ in the first m steps. Therefore, χ has the form

$$\begin{aligned}&\langle \mathsf{D_{i-1}.C_l[q(\tilde{v})], c_l}\rangle \rightarrow \langle \mathsf{D_{i-1}.C_l[C[p(\tilde{t})] \parallel tell(\tilde{v} = \tilde{r})], c_l}\rangle \rightarrow^* \langle \mathsf{D_{i-1}.C_m[p(\tilde{t})], c_m}\rangle\\ &\quad\rightarrow \langle \mathsf{D_0.C_{m+1}[p(\tilde{t})], c_m}\rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F}\rangle.\end{aligned}$$

Then, by the definition of $\mathsf{D_i}$, there exists a derivation

$$\begin{aligned}\xi_0 = &\ \langle \mathsf{D_i.C_l[q(\tilde{v})], c_l}\rangle \rightarrow \langle \mathsf{D_i.C_l[C[H \parallel tell(\tilde{t} = \tilde{s})] \parallel tell(\tilde{v} = \tilde{r})], c_l}\rangle\\ &\rightarrow^* \langle \mathsf{D_i.C_m[H \parallel tell(\tilde{t} = \tilde{s})], c_m}\rangle \rightarrow \langle \mathsf{D_0.C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m}\rangle.\end{aligned}$$

Observe that from the derivation $\langle \mathsf{D_0.C_{m+1}[p(\tilde{t})], c_m}\rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F}\rangle$ and (57), it follows that

$$\mathsf{w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F)} \text{ is defined, where } \mathsf{t = m(B, c_F)}. \tag{58}$$

The hypothesis on the variables implies that $Var(\mathsf{C_{m+1}[p(\tilde{t})], c_m}) \cap Var(\mathsf{u}) = \emptyset$. Then, by the definition of a transformation sequence and since $\mathsf{u} \in \mathsf{D_{i-1}}$, there exists a declaration $\mathsf{p(\tilde{s}) \leftarrow H_0 \in D_0}$. By Lemma A.3 and part 1 of Lemma A.5, it follows that there exists a constraint $\mathsf{d_F}$ such that

$$\mathsf{w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, d_F) \leq w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F)} \tag{59}$$

and

$$\exists_{-Var(\mathsf{C_{m+1}[p(\tilde{t})], c_m})}\mathsf{d_F} = \exists_{-Var(\mathsf{C_{m+1}[p(\tilde{t})], c_m})}\mathsf{c''_F}. \tag{60}$$

Therefore, by the definition of $\mathsf{w_t}$, by (59), and since $\mathsf{w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F)}$ is defined, there exists a derivation

$$\xi_1 = \langle \mathsf{D_0.C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m} \rangle \to^* \langle \mathsf{D_0.B', c'_F} \rangle,$$

where $\exists_{-Var(\mathsf{C_{m+1}[H \parallel tell(\tilde{t}=\tilde{s})], c_m})} \mathsf{c'_F} = \exists_{-Var(\mathsf{C_{m+1}[H \parallel tell(\tilde{t}=\tilde{s})], c_m})} \mathsf{d_F}$ and, by (58),

$$\mathsf{m(B', c'_F) = m(B, c_F)}. \tag{61}$$

By (60),

$$\exists_{-Var(\mathsf{C_{m+1}, c_m})} \mathsf{c'_F} = \exists_{-Var(\mathsf{C_{m+1}, c_m})} \mathsf{c''_F} \tag{62}$$

holds and, by definition of weight, we obtain

$$\mathsf{w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, c'_F) = w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, d_F)}. \tag{63}$$

Moreover, we can assume without loss of generality that $Var(\xi_0) \cap Var(\xi_1) = Var(\mathsf{C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m})$. Then, by the definition of procedure call

$$Var(\mathsf{C_l[q(\tilde{v})], c_l}) \cap (Var(\mathsf{c'_F}) \cup Var(\mathsf{c''_F})) \subseteq Var(\mathsf{C_{m+1}, c_m}), \tag{64}$$

and there exists a derivation

$$\begin{aligned} \xi = {} & \langle \mathsf{D_i.C_l[q(\tilde{v})], c_l} \rangle \to \langle \mathsf{D_i.C_l[C[H \parallel tell(\tilde{t} = \tilde{s})] \parallel tell(\tilde{v} = \tilde{r})], c_l} \rangle \\ & \to^* \langle \mathsf{D_i.C_m[H \parallel tell(\tilde{t} = \tilde{s})], c_m} \rangle \to \langle \mathsf{D_0.C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m} \rangle \\ & \to^* \langle D_0.B', c'_F \rangle \end{aligned}$$

such that the first $m - 1$ derivation steps do not use rule **R2** and the m-th derivation step uses rule **R2**. We now have the following equalities:

$$\begin{aligned} & \exists_{-Var(\mathsf{C_l[q(\tilde{v})], c_l})} \mathsf{c'_F} && = \text{by (64) and by construction} \\ & \exists_{-Var(\mathsf{C_l[q(\tilde{v})], c_l})} (\mathsf{c_m} \wedge \exists_{-Var(\mathsf{C_{m+1}, c_m})} \mathsf{c'_F}) && = \text{by (62)} \\ & \exists_{-Var(\mathsf{C_l[q(\tilde{v})], c_l})} (\mathsf{c_m} \wedge \exists_{-Var(\mathsf{C_{m+1}, c_m})} \mathsf{c''_F}) && = \text{by (64) and by construction} \\ & \exists_{-Var(\mathsf{C_l[q(\tilde{v})], c_l})} \mathsf{c''_F} && = \text{by the first statement in (57)} \\ & \exists_{-Var(\mathsf{C_l[q(\tilde{v})], c_l})} \mathsf{c_F}. \end{aligned}$$

By the definition of weight, $\mathsf{w_t(C_l[q(\tilde{v})], c_l, c'_F) = w_t(C_l[q(\tilde{v})], c_l, c''_F)}$, by (63) and (59), $\mathsf{w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, c'_F) \le w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F)}$ and $\mathsf{w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F) < w_t(C_l[q(\tilde{v})], c_l, c''_F)}$, since χ is a split derivation. Therefore, $\mathsf{w_t(C_{m+1}[H \parallel tell(\tilde{t} = \tilde{s})], c_m, c'_F) < w_t(C_l[q(\tilde{v})], c_l, c'_F)}$, and then, by definition, ξ is a split derivation in $\mathsf{D_i \cup D_0}$. This, together with (61), implies the thesis.

Case 2. A tell constraint in d is eliminated or introduced. In the first case, let $\mathsf{d'}$ be the corresponding declaration in $\mathsf{D_i}$. Hence the situation is the following:

—$\mathsf{d: q(\tilde{r}) \leftarrow C[tell(\tilde{s} = \tilde{t}) \parallel H]}$,
—$\mathsf{d': q(\tilde{r}) \leftarrow C[H\sigma]}$,

where σ is a relevant most general unifier of s and t, and the variables in the domain of σ do not occur in either $\mathsf{C[\ ]}$ or $\mathsf{q(\tilde{r})}$. Observe that for any derivation that uses the declaration d, we can construct another derivation such that the agent $\mathsf{tell(\tilde{s} = \tilde{t})}$ is evaluated before H. Then the thesis follows from Lemma A.5 and from the argument used in the proof of Case 2 of Proposition 4.5. The proof for the tell introduction is analogous, and hence is omitted.

Case 3. d is backward instantiated.

Let d' be the corresponding declaration in D_i. The situation is the following:

—d: $q(\tilde{r}) \leftarrow C[p(\tilde{t})] \in D_{i-1}$,
—d': $q(\tilde{r}) \leftarrow C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})] \in D_i$,

where c: $p(\tilde{s}) \leftarrow tell(b) \parallel B \in D_{i-1}$ has no variable in common with d (the case d': $q(\tilde{r}) \leftarrow C[p(\tilde{t}) \parallel tell(\tilde{t} = \tilde{s})]$ is analogous, and hence omitted). We distinguish two cases:

(1) There is no procedure call to $p(\tilde{t})$ in the first m steps. Therefore, χ has the form

$$\begin{aligned}&\langle D_{i-1}.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_{i-1}.C_l[C[p(\tilde{t})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle\\ &\quad\rightarrow^* \langle D_{i-1}.C_m[p(\tilde{t})], c_m\rangle \rightarrow \langle D_0.C_{m+1}[p(\tilde{t})], c_m\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle.\end{aligned}$$

Without loss of generality, we can assume that $Var(\chi)2 \cap Var(p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})) = Var(\tilde{t})$. Then, by the definition of D_i, there exists a derivation corresponding to χ,

$$\begin{aligned}\xi_0 = &\langle D_i.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_i.C_l[C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle\\ &\rightarrow^* \langle D_i.C_m[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})], c_m\rangle\\ &\rightarrow \langle D_0.C_{m+1}[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})], c_m\rangle.\end{aligned}$$

Following the same reasoning as in Case 3 of Lemma A.5, we can prove that there exists a constraint d_F such that

$$w_t(C_{m+1}[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})], c_m, d_F) \leq w_t(C_{m+1}[p(\tilde{t})], c_m, c''_F),$$

where $\exists_{-Var(C_{m+1}[p(\tilde{t})],c_m)} d_F = \exists_{-Var(C_{m+1}[p(\tilde{t})],c_m)} c''_F$ and $t = m(B'', c''_F)$. The rest of the proof is analogous to Case 1 (unfolding), and hence is omitted.

(2) There exists $k < m$ such that the k-th derivation step of χ is the procedure call $p(\tilde{t})$. We distinguish two more cases:

(2*a*) $p \neq q$. In this case we can assume, without loss of generality, that χ has the form

$$\begin{aligned}&\langle D_{i-1}.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_{i-1}.C_l[C[p(\tilde{t})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle\\ &\quad\rightarrow^* \langle D_{i-1}.C_k[p(\tilde{t})], c_k\rangle \rightarrow \langle D_{i-1}.C_k[tell(\overline{b}) \parallel \overline{B} \parallel tell(\tilde{t} = \tilde{s}')], c_k\rangle\\ &\quad\rightarrow^* \langle D_{i-1}.A_m, c_m\rangle \rightarrow \langle D_0.A_{m+1}, c_m\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle,\end{aligned}$$

where $c' = p(\tilde{s}') \leftarrow tell(\overline{b}) \parallel \overline{B}$ is a renaming of c such that $Var(c') \cap Var(d') = \emptyset$. In this case there exists a derivation

$$\begin{aligned}&\langle D_i.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_i.C_l[C[p(\tilde{t}) \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})] \parallel tell(\tilde{v} = \tilde{r})], c_l\rangle\\ &\quad\rightarrow^* \langle D_i.C_k[tell(\overline{b}) \parallel \overline{B} \parallel tell(\tilde{t} = \tilde{s}') \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})], c_k\rangle.\end{aligned}$$

Now observe that, given any set of declarations, if there exists a derivation χ' for the configuration $\langle C'[tell(\overline{b}) \parallel \overline{B} \parallel tell(\tilde{t} = \tilde{s}')], c'\rangle$, where c' is satisfiable and $Var(C', c') \cap Var(b, \tilde{s}) = \emptyset$, then there exists a derivation for $\langle C'[tell(\overline{b}) \parallel \overline{B} \parallel tell(\tilde{t} = \tilde{s}') \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})], c'\rangle$ which performs the same steps of χ' plus (possibly) two steps corresponding to the evaluation of $tell(b)$ and $tell(\tilde{t} = \tilde{s})$. Since $(\tilde{t} = \tilde{s}') \wedge (\tilde{t} = \tilde{s})$ is logically equivalent to $(\tilde{t} = \tilde{s}') \wedge (\tilde{s}' = \tilde{s})$, we can substitute $tell(\tilde{t} = \tilde{s}') \parallel tell(\tilde{t} = \tilde{s})$ for $tell(\tilde{t} = \tilde{s}') \parallel tell(\tilde{s}' = \tilde{s})$. Moreover, since $p(\tilde{s}') \leftarrow tell(\overline{b}) \parallel \overline{B}$ is a renaming of c and therefore $\mathcal{D} \models (\overline{b} \wedge (\tilde{s}' = \tilde{s})) \rightarrow b$ holds, we can drop the agent $tell(b)$.

Finally, observe that $\tilde{s}' = \tilde{s}$ can be reduced to a conjunction of equations of the form $\tilde{X} = \tilde{Y}$, where $\tilde{X} \subseteq Var(\tilde{s})$ and $\tilde{Y} \subseteq Var(\tilde{s}')$ are distinct variables. Therefore, we can drop the constraint $\mathsf{tell}(\tilde{s}' = \tilde{s})$, since the declarations used in the derivation are renamed separately and $Var(C'[\mathsf{tell}(\bar{b}) \parallel \bar{B} \parallel \mathsf{tell}(\tilde{t} = \tilde{s}')], c') \cap Var(\tilde{s}) = \emptyset$. Then the thesis holds for this case.

(2b) $p = q$. In this case, the situation is the following:

—d: $p(\tilde{r}) \leftarrow \mathsf{tell}(b') \parallel C''[p(\tilde{t})] \in D_{i-1}$,
—d': $p(\tilde{r}) \leftarrow \mathsf{tell}(b') \parallel C''[p(\tilde{t}) \parallel \mathsf{tell}(b) \parallel \mathsf{tell}(\tilde{t} = \tilde{s})] \in D_i$,

where c: $p(\tilde{s}) \leftarrow \mathsf{tell}(b) \parallel C'[p(\tilde{u})]$ is a renaming of d that has no variables in common with d. Let $c' = p(\tilde{s}') \leftarrow \mathsf{tell}(\bar{b}) \parallel \bar{C}[p(\tilde{u}')]$ be a renaming of c such that $Var(c') \cap Var(d') = \emptyset$. Now the proof is analogous to the previous one by observing that, for any set of declarations, if there exists a derivation χ' for $\langle \bar{C}'[\mathsf{tell}(\bar{b}) \parallel \bar{C}[p(\tilde{u}')] \parallel \mathsf{tell}(\tilde{t} = \tilde{s}')], c' \rangle$, where c' is satisfiable and $Var(\bar{C}', c') \cap Var(b, \tilde{s}) = \emptyset$, then there exists a derivation for $\langle \bar{C}'[\mathsf{tell}(\bar{b}) \parallel \bar{C}[p(\tilde{u}')] \parallel \mathsf{tell}(\tilde{t} = \tilde{s}') \parallel \mathsf{tell}(b) \parallel \mathsf{tell}(\tilde{t} = \tilde{s})], c' \rangle$ that performs the same steps of χ', plus some tell actions (analogously to the previous case, we can drop the tell agents $\mathsf{tell}(b)$ and $\mathsf{tell}(\tilde{t} = \tilde{s})$). This concludes the proof of this case.

Case 4. An ask guard in d is simplified. Let

—d: $q(\tilde{r}) \leftarrow C[\sum_{j=1}^{n} \mathsf{ask}(c_j) \rightarrow B_j]$,
—d': $q(\tilde{r}) \leftarrow C[\sum_{j=1}^{n} \mathsf{ask}(c'_j) \rightarrow B_j] \in D_i$,

where for $j \in [1, n]$, $\mathcal{D} \models \exists_{-Var(q(\tilde{r}),C,B_j)} (pc(C[\ ]) \wedge c_j) \leftrightarrow (pc(C[\ ]) \wedge c'_j)$, and $d \in D_{i-1}$ is the declaration to which the guard simplification was applied.

By the definition of split derivation, χ has the form

$$\chi = \langle D_{i-1}.C_l[q(\tilde{v})], c_l \rangle \rightarrow \left\langle D_{i-1}.C_l \left[C \left[\sum_{j=1}^{n} \mathsf{ask}(c_j) \rightarrow B_j \right] \parallel \mathsf{tell}(\tilde{v} = \tilde{r}) \right], c_l \right\rangle$$
$$\rightarrow^* \left\langle D_{i-1}.C_m \left[\sum_{j=1}^{n} \mathsf{ask}(c_j) \rightarrow B_j \right], c_m \right\rangle \rightarrow \langle D_0.A_{m+1}, c_m \rangle \rightarrow^* \langle D_0.B'', c''_F \rangle.$$

Since by the inductive hypothesis for any agent A, $\mathcal{O}(D_0.A) = \mathcal{O}(D_{i-1}.A)$, it is easy to check that there exists a derivation

$$\chi' = \langle D_{i-1}.C_l[q(\tilde{v})], c_l \rangle \rightarrow \left\langle D_{i-1}.C_l \left[C \left[\sum_{j=1}^{n} \mathsf{ask}(c_j) \rightarrow B_j \right] \parallel \mathsf{tell}(\tilde{v} = \tilde{r}) \right], c_l \right\rangle$$
$$\rightarrow^* \left\langle D_{i-1}.C_m \left[\sum_{j=1}^{n} \mathsf{ask}(c_j) \rightarrow B_j \right], c_m \right\rangle \rightarrow^* \left\langle D_{i-1}.C_{m+h} \left[\sum_{j=1}^{n} \mathsf{ask}(c_j) \right.\right.$$
$$\left.\left. \rightarrow B_j \right], c_{m+h} \right\rangle \rightarrow^* \langle D_{i-1}.\bar{B}, \bar{c}_F \rangle$$

such that $\exists_{-Var(\mathsf{C_I[q(\tilde{v})],c_I})}\bar{\mathsf{c}}_\mathsf{F} = \exists_{-Var(\mathsf{C_I[q(\tilde{v})],c_I})}\mathsf{c}''_\mathsf{F}$ and $\mathsf{m}(\bar{\mathsf{B}}, \bar{\mathsf{c}}_\mathsf{F}) = \mathsf{m}(\mathsf{B}'', \mathsf{c}''_\mathsf{F})$. From (57) it follows that

$$\exists_{-Var(\mathsf{C_I[q(\tilde{v})],c_I})}\bar{\mathsf{c}}_\mathsf{F} = \exists_{-Var(\mathsf{C_I[q(\tilde{v})],c_I})}\mathsf{c}_\mathsf{F} \text{ and } \mathsf{m}(\bar{\mathsf{B}}, \bar{\mathsf{c}}_\mathsf{F}) = \mathsf{m}(\mathsf{B}, \mathsf{c}_\mathsf{F}). \tag{65}$$

Without loss of generality, we can assume that χ' is chosen in such a way that the first $m+h$ steps of χ' do not use rule **R2** and that h is maximal, in the sense that either $\mathsf{c}_{\mathsf{m+h}}$ is not satisfiable, or in the $m+h+1$-th step, we can only use rule **R2**.

In the first case, let $\mathsf{C}'_{\mathsf{m+h}}$ be the context obtained from $\mathsf{C}_{\mathsf{m+h}}$, as follows: any (renamed) occurrence of the agent $\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c_j}) \rightarrow \mathsf{B_j}$ in $\mathsf{C}_{\mathsf{m+h}}[\ ]$, introduced in χ_0 by a procedure call of the form $\mathsf{q}(\tilde{\mathsf{s}})$, is replaced by a (suitably renamed) occurrence of the agent $\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c'_j}) \rightarrow \mathsf{B_j}$. Then, by definition of $\mathsf{D_i}$, we have that

$$\begin{aligned}\xi \;=\; &\langle \mathsf{D_i.C_I[q(\tilde{v})], c_I}\rangle \rightarrow \left\langle \mathsf{D_i.C_I}\left[\mathsf{C}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c'_j}) \rightarrow \mathsf{B_j}\right] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})\right], \mathsf{c_I}\right\rangle \\ &\rightarrow^* \left\langle \mathsf{D_i.C'_{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c'_j}) \rightarrow \mathsf{B_j}\right], \mathsf{c_{m+h}}\right\rangle\end{aligned}$$

is a derivation in $\mathsf{D_i}$ which does not use rule **R2** and such that

$$\mathsf{m}\left(\mathsf{C'_{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c'_j}) \rightarrow \mathsf{B_j}\right], \mathsf{c_{m+h}}\right) = \mathsf{m}(\mathsf{B}, \mathsf{c_F}) = \mathsf{ff}.$$

Then the thesis follows by definition of a split derivation.

Now assume that $\mathsf{c_{m+h}}$ is satisfiable. By Lemma A.6 and (65), there exists a constraint $\bar{\mathsf{d}}$ such that $Var(\bar{\mathsf{d}}) \subseteq Var(\mathsf{C_{m+h}}[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c_j}) \rightarrow \mathsf{B_j}], \mathsf{c_{m+h}})$ and

$$\mathsf{w_t}\left(\mathsf{C_{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c_j}) \rightarrow \mathsf{B_j}\right], \mathsf{c_{m+h}}, \bar{\mathsf{d}}\right) \leq \mathsf{w_t}(\mathsf{C_I[q(\tilde{v})], c_I, c_F}), \tag{66}$$

where

$$\exists_{-Var(\mathsf{C_I[q(\tilde{v})],c_I})}\bar{\mathsf{d}} = \exists_{-Var(\mathsf{C_I[q(\tilde{v})],c_I})}\mathsf{c_F} \text{ and } \mathsf{t} = \mathsf{m}(\mathsf{B}, \mathsf{c_F}). \tag{67}$$

By definition of weight, by (66), and since $Var(\bar{\mathsf{d}}) \subseteq Var(\mathsf{C_{m+h}}[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c_j}) \rightarrow \mathsf{B_j}], \mathsf{c_{m+h}})$, there exists a derivation

$$\left\langle \mathsf{D_0.C_{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c_j}) \rightarrow \mathsf{B_j}\right], \mathsf{c_{m+h}}\right\rangle \rightarrow^* \langle \mathsf{D_0}.\bar{\mathsf{B}}', \bar{\mathsf{d}}'\rangle$$

such that $\exists_{-Var(\mathsf{C_{m+h}}[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c_j}) \rightarrow \mathsf{B_j}], \mathsf{c_{m+h}})}\bar{\mathsf{d}}' = \bar{\mathsf{d}}$ and $\mathsf{m}(\bar{\mathsf{B}}', \bar{\mathsf{d}}') = \mathsf{t}$. Then, by the definition of weight and by (66),

$$\mathsf{w_t}\left(\mathsf{C_{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c_j}) \rightarrow \mathsf{B_j}\right], \mathsf{c_{m+h}}, \bar{\mathsf{d}}'\right) \leq \mathsf{w_t}(\mathsf{C_I[q(\tilde{v})], c_I, c_F}) \tag{68}$$

holds. Without loss of generality, we can assume that $Var(\bar{d}') \cap Var(C_l[q(\tilde{v})], c_l) \subseteq Var(C_{m+h}, c_{m+h})$. Therefore, from (67) it follows that

$$\exists_{-Var(C_l[q(\tilde{v})],c_l)}\bar{d}' = \exists_{-Var(C_l[q(\tilde{v})],c_l)}c_F \text{ and } m(\bar{B}', \bar{d}') = m(B, c_F). \tag{69}$$

Let $\tilde{B}' = C'_{m+h}[\sum_{j=1}^{n} ask(c'_j) \to B_j]$ be the agent obtained from

$$\tilde{B} = C_{m+h}\left[\sum_{j=1}^{n} ask(c_j) \to B_j\right]$$

as follows: any (renamed) occurrence of the agent $\sum_{j=1}^{n} ask(c_j) \to B_j$ in $C_{m+h}[\ ]$, introduced in χ_0 by a procedure call of the form $q(\tilde{s})$, is replaced by a (suitably renamed) occurrence of the agent $\sum_{j=1}^{n} ask(c'_j) \to B_j$. By the definition of D_i and since $\langle D_{i-1}.C_l[q(\tilde{v})], c_l\rangle \to^* \langle D_{i-1}.C_{m+h}[\sum_{j=1}^{n} ask(c_j) \to B_j], c_{m+h}\rangle$, there exists a derivation

$$\xi_0 = \langle D_i.C_l[q(\tilde{v})], c_l\rangle \to^* \left\langle D_i.C'_{m+h}\left[\sum_{j=1}^{n} ask(c'_j) \to B_j\right], c_{m+h}\right\rangle,$$

which does not use rule **R2**. Observe that, by construction, $\tilde{B}$, has the form $A_1 \parallel \ldots \parallel A_l$, where A_j is either a choice agent or Stop for each $j \in [1, l]$. Moreover, since the first $m+h$ steps of χ_0 do not use rule **R2** (and, therefore, it is not possible to evaluate a procedure call of the form $q(\tilde{s})$ inside a guarding context), $\tilde{B}'$ has the form $A'_1 \parallel \ldots \parallel A'_l$, where either $A'_j = A_j$ or A_j is a (renamed) occurrence of the agent $\sum_{j=1}^{n} ask(c_j) \to B_j$ while A'_j is a (suitably renamed) occurrence of the agent $\sum_{j=1}^{n} ask(c'_j) \to B_j$. By Lemma A.1, $\mathcal{D} \models c_{m+h} \to pc(C'[\ ])$, where $C'[\ ]$ is a renamed version of the context $C[\ ]$ in $\tilde{B}$, which was introduced in χ_0 by a procedure call of the form $q(\tilde{s})$.

Now, from the definition of derivation and of ask simplification, it follows that, if $ask(\tilde{c}_j) \to \tilde{B}_j$ is a choice branch in $\tilde{B}$ and $ask(\tilde{c}'_j) \to \tilde{B}_j$ is the corresponding choice branch in $\tilde{B}'$, then

$$\mathcal{D} \models \exists_{-Var(\tilde{B}_j, c_{m+h})}(c_{m+h} \wedge \tilde{c}_j) \leftrightarrow (c_{m+h} \wedge \tilde{c}'_j)$$

holds. Therefore, by using the same arguments as in Case 4 of Proposition 4.5, since (by inductive hypothesis) D_0 is weight-complete, and $\langle D_0.C_{m+h}[\sum_{j=1}^{n} ask(c_j) \to B_j], c_{m+h}\rangle \to^* \langle D_0.\bar{B}', \bar{d}'\rangle$, we obtain that there exists a split derivation in D_0 of the form

$$\nu = \left\langle D_0.C'_{m+h}\left[\sum_{j=1}^{n} ask(c'_j) \to B_j\right], c_{m+h}\right\rangle \to \langle D_0.B_{m+h+1}, c_{m+h}\rangle \to^* \langle D_0.B', c'_F\rangle$$

such that $\exists_{-Var(C'_{m+h}[\sum_{j=1}^{n} ask(c'_j) \to B_j], c_{m+h})}c'_F = \exists_{-Var(C'_{m+h}[\sum_{j=1}^{n} ask(c'_j) \to B_j], c_{m+h})}\bar{d}'$ and $m(B', c'_F) = m(\bar{B}', \bar{d}')$.

Then, by using the same arguments as in Case 4 of Proposition 4.5, from the definition of weight and from (68), it follows that

$$\begin{aligned} & \mathsf{w_t}\left(\mathsf{C}'_{\mathsf{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c}'_{\mathsf{j}}) \to \mathsf{B_j}\right], \mathsf{c_{m+h}}, \mathsf{c}'_{\mathsf{F}}\right) \\ & = \mathsf{w_t}\left(\mathsf{C}'_{\mathsf{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c}'_{\mathsf{j}}) \to \mathsf{B_j}\right], \mathsf{c_{m+h}}, \overline{\mathsf{d}'}\right) \qquad (70) \\ & = \mathsf{w_t}\left(\mathsf{C_{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c_j}) \to \mathsf{B_j}\right], \mathsf{c_{m+h}}, \overline{\mathsf{d}'}\right) \le \mathsf{w_t}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}, \mathsf{c_F}), \end{aligned}$$

where $\mathsf{t} = \mathsf{m}(\mathsf{B}', \mathsf{c}'_{\mathsf{F}})$. Moreover, we can assume without loss of generality that

$$Var(\xi_0) \cap Var(\nu) = Var\left(\mathsf{C}'_{\mathsf{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c}'_{\mathsf{j}}) \to \mathsf{B_j}\right], \mathsf{c_{m+h}}\right).$$

Then, by (69), we obtain

$$\exists_{-Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c}'_{\mathsf{F}} = \exists_{-Var(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})],\mathsf{c_I})}\mathsf{c_F} \text{ and } \mathsf{m}(\mathsf{B}', \mathsf{c}'_{\mathsf{F}}) = \mathsf{m}(\mathsf{B}, \mathsf{c_F}), \qquad (71)$$

and therefore, by the definition of weight,

$$\mathsf{w_t}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}, \mathsf{c}'_{\mathsf{F}}) = \mathsf{w_t}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}, \mathsf{c_F}) \qquad (72)$$

holds. By Lemma A.7 and by construction of $\mathsf{C}'_{\mathsf{m+h}}[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c}'_{\mathsf{j}}) \to \mathsf{B_j}]$,

$$\xi_1 = \left\langle \mathsf{D_i}.\mathsf{C}'_{\mathsf{m+h}}\left[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c}'_{\mathsf{j}}) \to \mathsf{B_j}\right], \mathsf{c_{m+h}}\right\rangle \to \langle \mathsf{D_0}.\mathsf{B_{m+h+1}}, \mathsf{c_{m+h}}\rangle \to^* \langle \mathsf{D_0}.\mathsf{B}', \mathsf{c}'_{\mathsf{F}}\rangle$$

is a split derivation in $\mathsf{D_i} \cup \mathsf{D_0}$. By the definition of split derivation $\mathsf{w_t}(\mathsf{B_{m+h+1}}, \mathsf{c_{m+h}}, \mathsf{c}'_{\mathsf{F}}) < \mathsf{w_t}(\mathsf{C}'_{\mathsf{m+h}}[\sum_{\mathsf{j=1}}^{\mathsf{n}} \mathsf{ask}(\mathsf{c}'_{\mathsf{j}}) \to \mathsf{B_j}], \mathsf{c_{m+h}}, \mathsf{c}'_{\mathsf{F}})$, where $\mathsf{t} = \mathsf{m}(\mathsf{B}', \mathsf{c}'_{\mathsf{F}})$. Then, by (72) and (70), we have that

$$\mathsf{w_t}(\mathsf{B_{m+h+1}}, \mathsf{c_{m+h}}, \mathsf{c}'_{\mathsf{F}}) < \mathsf{w_t}(\mathsf{C_I}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_I}, \mathsf{c}'_{\mathsf{F}}). \qquad (73)$$

Finally,

$$\begin{aligned} \xi \; = \; & \langle D_i.C_I[q(\tilde{v})], c_I\rangle \to^* \left\langle D_i.C'_{m+h}\left[\sum_{j=1}^{n} ask(c'_j) \to B_j\right], c_{m+h}\right\rangle \\ & \to \langle \mathsf{D_0}.\mathsf{B_{m+h+1}}, \mathsf{c_{m+h}}\rangle \to^* \langle \mathsf{D_0}.\mathsf{B}', \mathsf{c}'_{\mathsf{F}}\rangle \end{aligned}$$

is a derivation in $\mathsf{D_i} \cup \mathsf{D_0}$. By construction, the first $m + h$ steps of ξ do not use rule **R2** and the $m + h + 1$-th step uses rule **R2**. Thus the thesis follows from (73) and (71).

Case 5. d is the declaration to which either a branch elimination or an ask elimination was applied. In the case of branch elimination, the proof follows immediately from the fact that we also consider the inconsistent results of nonterminated computations. As for the ask elimination case, let us assume that

—d: $q(\tilde{r}) \leftarrow C[ask(true) \rightarrow H] \in D_{i-1}$and
—d′: $q(\tilde{r}) \leftarrow C[H] \in D_i$.

We show, by induction on the weight $w_t(C_I[q(\tilde{v})], c_I, c''_F)$, where $t = m(B, c_F)$, that there exists a split derivation $\xi = \langle D_i.C_I[q(\tilde{v})], c_I\rangle \rightarrow^* \langle D_0.B', c'_F\rangle$ in $D_i \cup D_0$, such that $\exists_{-Var(C_I[q(\tilde{v})],c_I)} c'_F = \exists_{-Var(C_I[q(\tilde{v})],c_I)} c''_F$ and $m(B', c'_F) = m(B'', c''_F)$. The proof follows by (57).

Base Case. In this case, $w_t(C_I[q(\tilde{v})], c_I, c''_F) = 0$, and, by definition of split derivation, $B'' = C_k[ask(true) \rightarrow H]$, χ has the form

$$\chi = \langle D_{i-1}.C_I[q(\tilde{v})], c_I\rangle \rightarrow \langle D_{i-1}.C_I[C[ask(true) \rightarrow H] \parallel tell(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \\ \langle D_{i-1}.C_k[ask(true) \rightarrow H], c''_F\rangle.$$

rule **R2** is not used, and hence each derivation step is done in D_{i-1}. Moreover, observe that since $t \in \{ss, dd, ff\}$, if c''_F is satisfiable, then C_k is a guarding context. It is then easy to check that

$$\xi = \langle D_i.C_I[q(\tilde{v})], c_I\rangle \rightarrow \langle D_i.C_I[C[H] \parallel tell(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_i.C_k[H], c''_F\rangle$$

is a split derivation in $D_i \cup D_0$, such that $m(C_k[H], c''_F) = m(C_k[ask(true) \rightarrow H], c''_F) \in \{dd, ff\}$; the thesis follows by the previous observation.

Induction Step. Assume that $w_t(C_I[q(\tilde{v})], c_I, c''_F) = n > 0$ and that χ has the form

$$\chi = \langle D_{i-1}.C_I[q(\tilde{v})], c_I\rangle \rightarrow \langle D_{i-1}.C_I[C[ask(true) \rightarrow H] \parallel tell(\tilde{v} = \tilde{r})], c_I\rangle \\ \rightarrow^* \langle D_{i-1}.C_m[ask(true) \rightarrow H], c_m\rangle \rightarrow \langle D_0.C_m[H], c_m\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle,$$

since the other case is immediate. By the definition of D_i and since χ is a split derivation, there exists a derivation

$$\xi_0 = \langle D_i.C_I[q(\tilde{v})], c_I\rangle \rightarrow \langle D_i.C_I[C[H] \parallel tell(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_i.C_m[H], c_m\rangle$$

which does not use rule **R2**. Moreover, by definition of split derivation

$$w_t(C_m[H], c_m, c''_F) < w_t(C_I[q(\tilde{v})], c_I, c''_F),$$

and therefore, by inductive hypothesis, there exists a split derivation in $D_i \cup D_0$,

$$\xi_1 = \langle D_i.C_m[H], c_m\rangle \rightarrow^* \langle D_0.B', c'_F\rangle$$

such that

$$m(B', c'_F) = m(B'', c''_F) = t \text{ and } \exists_{-Var(C_m[H],c_m)} c'_F = \exists_{-Var(C_m[H],c_m)} c''_F. \tag{74}$$

Without loss of generality, we can assume that $Var(\xi_0) \cap Var(\xi_1) \subseteq Var(C_m[H], c_m)$. Therefore, by (74) and by definition of c'_F and c''_F,

$$\begin{aligned} &\exists_{-Var(C_I[q(\tilde{v})],c_I)} c'_F \\ &\quad = \exists_{-Var(C_I[q(\tilde{v})],c_I)}(c_m \wedge \exists_{-Var(C_m[H],c_m)} c'_F) \\ &\quad = \exists_{-Var(C_I[q(\tilde{v})],c_I)}(c_m \wedge \exists_{-Var(C_m[H],c_m)} c''_F) \\ &\quad = \exists_{-Var(C_I[q(\tilde{v})],c_I)} c''_F. \end{aligned} \tag{75}$$

Then, by definition of weight, since $\mathsf{w_t}(\mathsf{C_m[H]}, \mathsf{c_m}, \mathsf{c''_F}) < \mathsf{w_t}(\mathsf{C_l[q(\tilde{v})]}, \mathsf{c_l}, \mathsf{c''_F})$ and by (74),

$$\mathsf{w_t}(\mathsf{C_m[H]}, \mathsf{c_m}, \mathsf{c'_F}) < \mathsf{w_t}(\mathsf{C_l[q(\tilde{v})]}, \mathsf{c_l}, \mathsf{c'_F}). \tag{76}$$

Moreover, by our hypothesis on the variables of ξ_0 and of ξ_1, there exists a derivation $\mathsf{D_i} \cup \mathsf{D_0}$,

$$\begin{aligned}\xi \;=\; & \langle \mathsf{D_i.C_l[q(\tilde{v})], c_l} \rangle \rightarrow \langle \mathsf{D_i.C_l[C[H] \parallel tell(\tilde{v} = \tilde{r})], c_l} \rangle \\ & \rightarrow^* \langle \mathsf{D_i.C_m[H], c_m} \rangle \rightarrow^* \langle \mathsf{D_0.B', c'_F} \rangle.\end{aligned}$$

By (74) and (76), since ξ_0 do not use Rule **R2** and ξ_1 is a split derivation in $\mathsf{D_i} \cup \mathsf{D_0}$, we have that ξ is a split derivation in $\mathsf{D_i} \cup \mathsf{D_0}$, such that $\mathsf{m(B', c'_F)} = \mathsf{m(B'', c''_F)}$. The thesis now follows by (75).

Case 6. An ask guard in d is distributed. Let

—d: $\mathsf{q(\tilde{r}) \leftarrow C[H \parallel \sum_{j=1}^{n} ask(c_j) \rightarrow B_j] \in D_{i-1}}$,
—$\mathsf{d'}$: $\mathsf{q(\tilde{r}) \leftarrow C[\sum_{j=1}^{n} ask(c_j) \rightarrow (H \parallel B_j)] \in D_i}$,

where, for every constraint $\mathsf{e'}$ such that $Var(\mathsf{e'}) \cap Var(\mathsf{d}) \subseteq Var(\mathsf{q(\tilde{r}), C})$, if $\langle \mathsf{D_{i-1}.H, e' \wedge pc(C[\,])} \rangle$ is productive, then there exists at least one $j \in [1, n]$ such that $\mathcal{D} \models (\mathsf{e'} \wedge \mathsf{pc(C[\,])}) \rightarrow \mathsf{c_j}$ and, for each $j \in [1, n]$, either $\mathcal{D} \models (\mathsf{e'} \wedge \mathsf{pc(C[\,])}) \rightarrow \mathsf{c_j}$ or $\mathcal{D} \models (\mathsf{e'} \wedge \mathsf{pc(C[\,])}) \rightarrow \neg \mathsf{c_j}$.

By the definition of split derivation, χ has the form

$$\begin{aligned}\chi \;=\; & \langle \mathsf{D_{i-1}.C_l[q(\tilde{v})], c_l} \rangle \rightarrow \left\langle \mathsf{D_{i-1}.C_l} \left[\mathsf{C} \left[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask(c_j)} \rightarrow \mathsf{B_j} \right] \parallel \mathsf{tell(\tilde{v} = \tilde{r})} \right], \mathsf{c_l} \right\rangle \\ & \rightarrow^* \left\langle \mathsf{D_{i-1}.C_m} \left[\sum_{j=1}^{n} \mathsf{ask(c_j)} \rightarrow \mathsf{B_j} \right], \mathsf{c_m} \right\rangle \rightarrow \langle \mathsf{D_0.A_{m+1}, c_m} \rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F} \rangle.\end{aligned}$$

If the first $m-1$ steps of χ do not evaluate the agent H, then the proof is analogous to that of Case 6 of Lemma A.5. Otherwise, let us assume that

$$\begin{aligned}\chi \;=\; & \langle \mathsf{D_{i-1}.C_l[q(\tilde{v})], c_l} \rangle \rightarrow \left\langle \mathsf{D_{i-1}.C_l} \left[\mathsf{C} \left[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask(c_j)} \rightarrow \mathsf{B_j} \right] \parallel \mathsf{tell(\tilde{v} = \tilde{r})} \right], \mathsf{c_l} \right\rangle \\ & \rightarrow^* \left\langle \mathsf{D_{i-1}.C_m} \left[\mathsf{H'} \parallel \sum_{j=1}^{n} \mathsf{ask(c_j)} \rightarrow \mathsf{B_j} \right], \mathsf{c_m} \right\rangle \rightarrow \langle \mathsf{D_0.A_{m+1}, c_m} \rangle \rightarrow^* \langle \mathsf{D_0.B'', c''_F} \rangle.\end{aligned}$$

Since, by the inductive hypothesis, for any agent A, $\mathcal{O}(\mathsf{D_0.A}) = \mathcal{O}(\mathsf{D_{i-1}.A})$, there exists a derivation

$$\begin{aligned}\chi' \;=\; & \langle \mathsf{D_{i-1}.C_l[q(\tilde{v})], c_l} \rangle \rightarrow \left\langle \mathsf{D_{i-1}.C_l} \left[\mathsf{C} \left[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask(c_j)} \rightarrow \mathsf{B_j} \right] \parallel \mathsf{tell(\tilde{v} = \tilde{r})} \right], \mathsf{c_l} \right\rangle \\ & \rightarrow^* \left\langle \mathsf{D_{i-1}.C_k} \left[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask(c_j)} \rightarrow \mathsf{B_j} \right], \mathsf{c_k} \right\rangle \rightarrow^* \langle \mathsf{D_{i-1}.\overline{B}, \overline{c}_F} \rangle,\end{aligned}$$

where $\exists_{-Var(\mathsf{C_l[q(\tilde{v})],c_l})}\overline{\mathsf{c}}_\mathsf{F} = \exists_{-Var(\mathsf{C_l[q(\tilde{v})],c_l})}\mathsf{c}''_\mathsf{F}$ and $\mathsf{m}(\overline{\mathsf{B}}, \overline{\mathsf{c}}_\mathsf{F}) = \mathsf{m}(\mathsf{B}'', \mathsf{c}''_\mathsf{F})$. By (57),

$$\exists_{-Var(\mathsf{C_l[q(\tilde{v})],c_l})}\overline{\mathsf{c}}_\mathsf{F} = \exists_{-Var(\mathsf{C_l[q(\tilde{v})],c_l})}\mathsf{c}_\mathsf{F} \text{ and } \mathsf{m}(\overline{\mathsf{B}}, \overline{\mathsf{c}}_\mathsf{F}) = \mathsf{m}(\mathsf{B}, \mathsf{c}_\mathsf{F}). \tag{77}$$

Without loss of generality, we can assume that the first k steps of $\underline{\chi}'$ neither use rule **R2** nor contain the evaluation of any (renamed) occurrence $\overline{\mathsf{H}}$ of the agent H, where $\mathsf{q}(\overline{\tilde{\mathsf{r}}}') \leftarrow \overline{\mathsf{C}}[\overline{\mathsf{H}} \parallel \sum_{j=1}^{n} \mathsf{ask}(\overline{\mathsf{c}}_\mathsf{j}) \to \overline{\mathsf{B}}_\mathsf{j}]$ is a renamed version of the declaration d and $\overline{\mathsf{C}}[\overline{\mathsf{H}} \parallel \sum_{j=1}^{n} \mathsf{ask}(\overline{\mathsf{c}}_\mathsf{j}) \to \overline{\mathsf{B}}_\mathsf{j}]$ has been introduced by the evaluation of a procedure call of the form $\mathsf{q}(\tilde{\mathsf{s}})$. Moreover, we can assume that k is maximal, either $\mathsf{c_k}$ is not satisfiable or in the sense that the $k+1$-th step can only use either rule **R2** or evaluate a (renamed) occurrence of H introduced by a procedure call of the form $\mathsf{q}(\tilde{\mathsf{s}})$. If $\mathsf{c_k}$ is not satisfiable, then the proof is analogous to that in Case 4.

Assume then that $\mathsf{c_k}$ is satisfiable. By Lemma A.6 and (77), there exists a constraint $\overline{\mathsf{d}}$ such that $Var(\overline{\mathsf{d}}) \subseteq Var(\mathsf{C_k}[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to \mathsf{B_j}], \mathsf{c_k})$, and

$$\mathsf{w_t}\left(\mathsf{C_k}\left[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to \mathsf{B_j}\right], \mathsf{c_k}, \overline{\mathsf{d}}\right) \le \mathsf{w_t}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_l}, \mathsf{c_F}), \tag{78}$$

where

$$\exists_{-Var(\mathsf{C_l[q(\tilde{v})],c_l})}\overline{\mathsf{d}} = \exists_{-Var(\mathsf{C_l[q(\tilde{v})],c_l})}\mathsf{c}_\mathsf{F} \text{ and } \mathsf{t} = \mathsf{m}(\mathsf{B}, \mathsf{c}_\mathsf{F}). \tag{79}$$

By definition of weight, by (78), and since $Var(\overline{\mathsf{d}}) \subseteq Var(\mathsf{C_k}[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to \mathsf{B_j}], \mathsf{c_k})$, there exists a derivation

$$\left\langle \mathsf{D_0.C_k}\left[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to \mathsf{B_j}\right], \mathsf{c_k} \right\rangle \to^* \langle \mathsf{D_0}.\overline{\mathsf{B}}', \overline{\mathsf{d}}' \rangle$$

such that $\exists_{-Var(\mathsf{C_k[H} \parallel \sum_{j=1}^{n} \mathsf{ask(c_j) \to B_j],c_k})}\overline{\mathsf{d}}' = \overline{\mathsf{d}}$ and $\mathsf{m}(\overline{\mathsf{B}}', \overline{\mathsf{d}}') = \mathsf{t}$. Then, by the definition of weight and by (78),

$$\mathsf{w_t}\left(\mathsf{C_k}\left[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to \mathsf{B_j}\right], \mathsf{c_k}, \overline{\mathsf{d}}'\right) \le \mathsf{w_t}(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_l}, \mathsf{c_F}). \tag{80}$$

Without loss of generality, we can assume that $Var(\overline{\mathsf{d}}') \cap Var(\mathsf{C_l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_l}) \subseteq Var(\mathsf{C_k}, \mathsf{c_k})$. Hence, from (79) it follows that

$$\exists_{-Var(\mathsf{C_l[q(\tilde{v})],c_l})}\overline{\mathsf{d}}' = \exists_{-Var(\mathsf{C_l[q(\tilde{v})],c_l})}\mathsf{c}_\mathsf{F} \text{ and } \mathsf{m}(\overline{\mathsf{B}}', \overline{\mathsf{d}}') = \mathsf{m}(\mathsf{B}, \mathsf{c}_\mathsf{F}). \tag{81}$$

Let $\mathsf{C}'_\mathsf{k}[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to (\mathsf{H} \parallel \mathsf{B_j})]$ be the agent obtained from $\mathsf{C_k}[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to \mathsf{B_j}]$ as follows: any (renamed) occurrence of the agent $\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to \mathsf{B_j}$ in $\mathsf{C_k}[\ ]$, introduced by a procedure call of the form $\mathsf{q}(\tilde{\mathsf{s}})$ is replaced by a (suitably) renamed occurrence of the agent $\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to (\mathsf{H} \parallel \mathsf{B_j})$.

By the definition of $\mathsf{D_i}$ and since $\langle \mathsf{D_{i-1}.C_l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_l} \rangle \to^* \langle \mathsf{D_{i-1}.C_k}[\mathsf{H} \parallel \sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to \mathsf{B_j}], \mathsf{c_k} \rangle$, there exists a derivation

$$\xi_0 = \langle \mathsf{D_i.C_l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c_l} \rangle \to^* \left\langle \mathsf{D_i.C'_k}\left[\sum_{j=1}^{n} \mathsf{ask}(\mathsf{c_j}) \to (\mathsf{H} \parallel \mathsf{B_j})\right], \mathsf{c_k} \right\rangle,$$

which does not use rule **R2**.

Now, by construction, $C'_k[\sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow (H \parallel B_j)]$ has the form $A_1 \parallel \ldots \parallel A_l$, where A_j is either a choice agent or Stop.

Moreover, since D_0 is weight-complete, $\langle D_0.C_k[H \parallel \sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow B_j], c_k\rangle \rightarrow^* \langle D_0.\overline{B}', \overline{d}'\rangle$, and analogously to Case 6 of Lemma A.5, there exists a split derivation

$$\xi_1 = \left\langle D_0.C'_k\left[\sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow (H \parallel B_j)\right], c_k\right\rangle \rightarrow^* \langle D_0.B', c'_F\rangle$$

such that $\exists_{-Var(C'_k[\sum_{j=1}^n \mathsf{ask}(c_j)\rightarrow(H \parallel B_j)],c_k)} c'_F) = \exists_{-Var(C'_k[\sum_{j=1}^n \mathsf{ask}(c_j)\rightarrow(H \parallel B_j)],c_k)} \overline{d}')$ and $m(B', c'_F) = m(\overline{B}', \overline{d}')$.

Then, by using the same arguments as in Case 6 of Lemma A.5, from the definition of weight and (80), it follows that

$$\begin{aligned} & w_t\left(C'_k\left[\sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow (H \parallel B_j)\right], c_k, c'_F\right) \\ &= w_t\left(C'_k\left[\sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow (H \parallel B_j)\right], c_k, \overline{d}'\right) \\ &\leq w_t\left(C_k\left[H \parallel \sum_{j=1}^n \mathsf{ask}(c_j) \rightarrow B_j\right], c_k, \overline{d}'\right) \leq w_t(C_l[q(\tilde{v})], c_l, c_F), \end{aligned}$$

where $t = m(B', c'_F)$.

From this point, the proof proceeds exactly as in Case 4 by using Lemma A.7, and is therefore omitted.

Case 7. Finally, assume that d is folded. Let

—d: $q(\tilde{r}) \leftarrow C[H]$ be the folded declaration ($\in D_{i-1}$),
—f: $p(\tilde{X}) \leftarrow H$ be the folding declaration ($\in D_0$),
—d': $q(\tilde{r}) \leftarrow C[p(\tilde{X})]$ be the result of the folding operation ($\in D_i$),

where, by definition of folding, $Var(d) \cap Var(\tilde{X}) \subseteq Var(H)$ and $Var(H) \cap (Var(\tilde{r}) \cup Var(C)) \subseteq Var(\tilde{X})$. Since C[ ] is a guarding context, the agent H in C[H] appears in the scope of an ask guard. By definition of split derivation, χ has the form

$$\begin{aligned} &\langle D_{i-1}.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_{i-1}.C_l[C[H] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_l\rangle \rightarrow^* \langle D_{i-1}.C_m[H], c_m\rangle \\ &\quad \rightarrow \langle D_0.C_{m+1}[H], c_m\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle, \end{aligned}$$

where $C_m[\ ]$ is a guarding context. Without loss of generality, we can assume that $Var(\chi) \cap Var(\tilde{X}) \subseteq Var(H)$. Then, from the definition of D_i, it follows that there exists a derivation

$$\begin{aligned} \xi_0 \;=\; &\langle D_i.C_l[q(\tilde{v})], c_l\rangle \rightarrow \langle D_i.C_l[C[p(\tilde{X})] \parallel \mathsf{tell}(\tilde{v} = \tilde{r})], c_l\rangle \\ &\rightarrow^* \langle D_i.C_m[p(\tilde{X})], c_m\rangle \rightarrow \langle D_0.C_{m+1}[p(\tilde{X})], c_m\rangle, \end{aligned}$$

which performs exactly the first *m* steps as χ. Since $\langle D_0.C_{m+1}[H], c_m\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle$, the definition of weight implies that $w_t(C_{m+1}[H], c_m, c''_F)$ is defined, where $\mathsf{t} = \mathsf{m}(\mathsf{B}'', \mathsf{c}''_\mathsf{F})$. Then, by (57), we have that

$$\mathsf{t} = \mathsf{m}(\mathsf{B}, \mathsf{c}_\mathsf{F}). \tag{82}$$

The definitions of derivation and folding imply that $Var(\mathsf{H}) \cap Var(\mathsf{C}_{\mathsf{m}+1}, \mathsf{c}_\mathsf{m}) \subseteq Var(\mathsf{H}) \cap (Var(\mathsf{C}, \tilde{\mathsf{r}})) \subseteq Var(\tilde{\mathsf{x}})$ holds. Moreover, from the assumptions on the variables, we obtain that $Var(\mathsf{c}''_\mathsf{F}) \cap Var(\tilde{\mathsf{x}}) \subseteq Var(\mathsf{H})$. Thus, from part 2 of Lemma A.4, it follows that there exists a constraint d' such that

$$\begin{aligned} w_\mathsf{t}(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m}, \mathsf{d}') &\leq w_\mathsf{t}(\mathsf{C}_{\mathsf{m}+1}[\mathsf{H}], \mathsf{c}_\mathsf{m}, \mathsf{c}''_\mathsf{F}) \text{ and} \\ \exists_{-Var(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m})}\mathsf{d}' &= \exists_{-Var(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m})}\mathsf{c}''_\mathsf{F}. \end{aligned} \tag{83}$$

From the definition of weight and the fact that $w_\mathsf{t}(\mathsf{C}_{\mathsf{m}+1}[\mathsf{H}], \mathsf{c}_\mathsf{m}, \mathsf{c}''_\mathsf{F})$ is defined, it follows that there exists a derivation $\xi_1 = \langle \mathsf{D}_0.\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m}\rangle \rightarrow^* \langle \mathsf{D}_0.\mathsf{B}', \mathsf{c}'_\mathsf{F}\rangle$, where $\mathsf{m}(\mathsf{B}', \mathsf{c}'_\mathsf{F}) = \mathsf{t}$ and $\exists_{-Var(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m})}\mathsf{c}'_\mathsf{F} = \exists_{-Var(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m})}\mathsf{d}'$. Then, by the definition of weight, $w_\mathsf{t}(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m}, \mathsf{c}'_\mathsf{F}) = w_\mathsf{t}(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m}, \mathsf{d}')$ and, therefore, by (83),

$$\begin{aligned} \exists_{-Var(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m})}\mathsf{c}'_\mathsf{F} &= \exists_{-Var(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m})}\mathsf{c}''_\mathsf{F} \text{ and} \\ w_\mathsf{t}(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m}, \mathsf{c}'_\mathsf{F}) &\leq w_\mathsf{t}(\mathsf{C}_{\mathsf{m}+1}[\mathsf{H}], \mathsf{c}_\mathsf{m}, \mathsf{c}''_\mathsf{F}) \end{aligned} \tag{84}$$

hold. Moreover, from (82), we obtain

$$\mathsf{m}(\mathsf{B}', \mathsf{c}'_\mathsf{F}) = \mathsf{m}(\mathsf{B}, \mathsf{c}_\mathsf{F}). \tag{85}$$

Without loss of generality, we can now assume that $Var(\xi_0) \cap Var(\xi_1) = Var(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m})$. Then, by (84) and (57), it follows that

$$\begin{aligned} &\exists_{-Var(\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l})}\mathsf{c}'_\mathsf{F} = \exists_{-Var(\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l})}(\mathsf{c}_\mathsf{m} \wedge \exists_{-Var(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m})}\mathsf{c}'_\mathsf{F}) \\ &\quad = \exists_{-Var(\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l})}(\mathsf{c}_\mathsf{m} \wedge \exists_{-Var(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m})}\mathsf{c}''_\mathsf{F}) = \exists_{-Var(\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l})}\mathsf{c}''_\mathsf{F} \\ &\quad = \exists_{-Var(\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l})}\mathsf{c}_\mathsf{F}. \end{aligned} \tag{86}$$

From the definition of weight $w_\mathsf{t}(\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l}, \mathsf{c}'_\mathsf{F}) = w_\mathsf{t}(\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l}, \mathsf{c}''_\mathsf{F})$ and since χ is a split derivation, we obtain $w_\mathsf{t}(\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l}, \mathsf{c}''_\mathsf{F}) > w_\mathsf{t}(\mathsf{C}_{\mathsf{m}+1}[\mathsf{H}], \mathsf{c}_\mathsf{m}, \mathsf{c}''_\mathsf{F})$. Then, from (84) and (86), it follows that

$$w_\mathsf{t}(\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l}, \mathsf{c}'_\mathsf{F}) > w_\mathsf{t}(\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{x}})], \mathsf{c}_\mathsf{m}, \mathsf{c}'_\mathsf{F}), \tag{87}$$

and, therefore, by construction

$$\begin{aligned} \xi \;=\; &\langle \mathsf{D}_\mathsf{i}.\mathsf{C}_\mathsf{l}[\mathsf{q}(\tilde{\mathsf{v}})], \mathsf{c}_\mathsf{l}\rangle \rightarrow \langle \mathsf{D}_\mathsf{i}.\mathsf{C}_\mathsf{l}[\mathsf{C}[\mathsf{p}(\tilde{\mathsf{X}})] \parallel \mathsf{tell}(\tilde{\mathsf{v}} = \tilde{\mathsf{r}})], \mathsf{c}_\mathsf{l}\rangle \rightarrow^* \langle \mathsf{D}_\mathsf{i}.\mathsf{C}_\mathsf{m}[\mathsf{p}(\tilde{\mathsf{X}})], \mathsf{c}_\mathsf{m}\rangle \\ &\rightarrow \langle \mathsf{D}_0.\mathsf{C}_{\mathsf{m}+1}[\mathsf{p}(\tilde{\mathsf{X}})], \mathsf{c}_\mathsf{m}\rangle \rightarrow^* \langle \mathsf{D}_0.\mathsf{B}', \mathsf{c}'_\mathsf{F}\rangle \end{aligned}$$

is a derivation in $\mathsf{D}_\mathsf{i} \cup \mathsf{D}_0$ such that: (a) rule **R2** is not used in the first $m - 1$ steps; (b) rule **R2** is used in the m-th step. The thesis then follows from (86), (85), and (87), thus concluding the proof. □

A.1 Proof of Correctness for Intermediate Results and Traces

In this section we show how the previous proofs can be adapted when considering intermediate results and traces as observables. We first consider Theorem 5.1. Since its proof is essentially the same as the one given for the total correctness theorem, here we provide only the intuition illustrating the (minor) modifications needed.

THEOREM 5.1 (Total Correctness 2). *Let* $D_0, \dots, D_n$ *be a transformation sequence, and* A *be an agent.*

—*If there exists a derivation* $\langle D_0.A, c\rangle \rightarrow^* \langle D_0.B, d\rangle$*, then there exists a derivation* $\langle D_n.A, c\rangle \rightarrow^* \langle D_n.B', d'\rangle$ *such that* $\mathcal{D} \models \exists_{-Var(A,c)} d' \rightarrow \exists_{-Var(A,c)} d$.

—*Conversely, if there exists a derivation* $\langle D_n.A, c\rangle \rightarrow^* \langle D_n.B, d\rangle$*, then there exists a derivation* $\langle D_0.A, c\rangle \rightarrow^* \langle D_0.B', d'\rangle$ *with* $\mathcal{D} \models \exists_{-Var(A,c)} d' \rightarrow \exists_{-Var(A,c)} d$.

PROOF. The proof of this result is essentially the same as that one of the total correctness Theorem 4.13 provided that in such a proof, as well as in the proofs of the related preliminary results, we perform the following changes:

(1) Rather than considering terminating derivations, we consider any (possibly non-maximal) finite derivation.
(2) Whenever in a proof we write, that given a derivation ξ, a derivation ξ' is constructed which performs the same steps of ξ, possibly in a different order, we now write that a derivation ξ'' is constructed which performs the same step as ξ (possibly in a different order) plus some other additional steps. Since the store grows monotonically in ccp derivations, clearly if a constraint c is the result of the derivation ξ, then a constraint c'' is the result of ξ'' such that $\mathcal{D} \models c" \rightarrow c$ holds. For example, for case 2 in the proof of Proposition 4.5 (in the Appendix), when considering a (non-maximal) derivation ξ which uses the declaration $H \leftarrow C[\mathsf{tell}(\tilde{s} = \tilde{t})] \parallel B]$ we can always construct a derivation ξ'' which performs all the steps of ξ (possibly plus others) and such that the $\mathsf{tell}(\tilde{s} = \tilde{t})$ agent is evaluated before B. Differently from the previous proof, now we are not ensured that the result of ξ is the same as that one of ξ'', since ξ is non-maximal (thus, ξ could also avoid the evaluation of $\mathsf{tell}(\tilde{s} = \tilde{t})$). However, we are ensured that the result of ξ'' is stronger (i.e., implies) that one of ξ. □

We now consider the correctness results for the restricted transformation system with respect to the traces. Also, in this case, the proofs follow the guidelines already presented in Section 4 and in the previous part of this Appendix. We then sketch the proofs by showing the relevant new notions and differences with respect to the previous ones.

In the remainder of this section we always refer to the restricted transformation system and to a given restricted transformation sequence $D_0, \dots, D_n$.

We start with the following definition.

Definition A.10 . Let D be a set of declarations and let ξ be the derivation

$$\langle D.A_1, c_1\rangle \rightarrow^* \langle D.A_m, c_m\rangle \rightarrow^* \langle D.A_n, c_n\rangle.$$

We define $\mathsf{tr}(\xi) =$

$$\exists_{-Var(A_1,c_1)}(c_1; c_2; \dots; c_n) = (c_1; (\exists_{-Var(A_1,c_1)} c_2); \dots; (\exists_{-Var(A_1,c_1)} c_n)).$$

The function *mode* $(m(A, d))$ is extended to also consider nonterminated derivations in the obvious way. We then extend the notion of weight split derivation, and weight-complete programs to the case of traces. Here, and in the following, the subscript t denotes a generic termination mode, that is, we assume

$t \in \{ss, dd, pp, ff\}$. We also say that a trace starts with c in case c is the first constraint appearing in that trace.

Definition A.11 (*Weight for Traces*). Given an agent A, a satisfiable constraint c, and a trace s starting with c, we define the *weight* of the agent A w.r.t. the trace s, notation $w_t(A, s)$, as follows:

$$w_t(A, s) = \min\{n \mid n = wh(\xi) \text{ and } \xi \text{ is a derivation } \langle D_0.A, c\rangle \rightarrow^* \langle D_0.B, d\rangle \text{ such that } \exists_{-Var(A,c)} s \preceq tr(\xi) \text{ and } t = m(B, d)\}.$$

Definition A.12 (*Split derivation for traces*). Let $D_0, \ldots, D_i$ be a transformation sequence. We call a derivation in $D_i \cup D_0$ a *split derivation for traces* if it has the form

$$\langle D_i.A_1, c_1\rangle \rightarrow^* \langle D_i.A_m, c_m\rangle \rightarrow \langle D_0.A_{m+1}, c_{m+1}\rangle \rightarrow^* \langle D_0.A_n, c_n\rangle,$$

where $m \in [1, n]$ and the following conditions hold:

(a) the first $m - 1$ derivation steps do not use rule **R2**;
(b) the m-th derivation step $\langle D_i.A_m, c_m\rangle \rightarrow \langle D_0.A_{m+1}, c_{m+1}\rangle$ uses rule **R2**;
(c) $w_t(A_1, (c_1; c_2; \ldots; c_n)) > w_t(A_{m+1}, (c_{m+1}; \ldots; c_n))$, where $t = m(A_n, c_n)$.

Definition A.13 . We call the program D_i *weight-complete for traces* iff, for any agent A and any satisfiable constraint c, the following hold: If there exists a derivation

$$\chi = \langle D_0.A, c\rangle \rightarrow^* \langle D_0.B, d\rangle \tag{88}$$

such that $m(B, d) \in \{ss, dd, pp, ff\}$, then there exists a split derivation in $D_i \cup D_0$,

$$\xi = \langle D_i.A, c\rangle \rightarrow^* \langle D_0.B', d'\rangle, \tag{89}$$

where $tr(\chi) \preceq tr(\xi)$ and $m(B', d') = m(B, d)$.

Proposition 4.9 also holds when considering as observables $\mathcal{O}_t$ rather than $\mathcal{O}$, and its proof is essentially the same, so we omit it.

The following lemma is obtained from Lemma A.1 by considering the weakest produced constraint wpc, rather than the produced constraint. The proof is analogous to that for Lemma A.1, and hence is omitted.

LEMMA A.14 . *Assume that there exists a derivation* $\langle D.C[A], c\rangle \rightarrow^* \langle D.C'[A], c'\rangle$*, where* c *is a satisfiable constraint and the context* $C'[\ ]$ *has the form*

$$A_1 \parallel \ldots \parallel \bar{C}[\ ] \parallel \ldots \parallel A_n.$$

Then $\mathcal{D} \models (wpc(\bar{C}[\ ]) \wedge c') \rightarrow wpc(C[\ ])$ *holds, and in case* $\bar{C}[\ ]$ *is the empty context,* $\mathcal{D} \models c' \rightarrow wpc(C[\ ])$ *also holds.*

In the following, we extend to a set of observables the (pre-order) relation $\preceq$ in the expected way: Given two sets of observables $\mathcal{O}_t(D_i.A)$ and $\mathcal{O}_t(D_j.A)$, we say that $\mathcal{O}_t(D_i.A) \preceq \mathcal{O}_t(D_j.A)$ iff, for any $\langle s, x\rangle \in \mathcal{O}_t(D_i.A)$, (with $x \in \{ss, dd, pp, ff\}$), there exists $\langle s', x\rangle \in \mathcal{O}_t(D_j.A)$ such that $s \preceq s'$. We denote by $\equiv$ the equivalence relation induced by $\preceq$ on sets of observables, that is, $\mathcal{O}_t(D_i.A) \equiv \mathcal{O}_t(D_j.A)$ iff $\mathcal{O}_t(D_i.A) \preceq \mathcal{O}_t(D_j.A)$ and $\mathcal{O}_t(D_j.A) \preceq \mathcal{O}_t(D_i.A)$.

The following is analogous to Proposition 4.5 for traces.

PROPOSITION A.15 (Partial Correctness for traces). *If, for each agent* A, $\mathcal{O}_t(\mathsf{D_0.A}) \equiv \mathcal{O}_t(\mathsf{D_i.A})$*, then, for each agent* A, $\mathcal{O}_t(\mathsf{D_{i+1}.A}) \preceq \mathcal{O}_t(\mathsf{D_i.A})$.

PROOF. We have to show that, given an agent A and a satisfiable constraint $\mathsf{c_I}$, if there exists a derivation $\xi = \langle \mathsf{D_{i+1}.A, c_I} \rangle \rightarrow^* \langle \mathsf{D_{i+1}.B, c_F} \rangle$, then there also exists a derivation $\xi' = \langle \mathsf{D_i.A, c_I} \rangle \rightarrow^* \langle \mathsf{D_i.B', c'_F} \rangle$ such that $\mathsf{tr}(\xi) \preceq \mathsf{tr}(\xi')$ and $\mathsf{m(B', c'_F) = m(B, c_F)}$.

The proof is analogous to that for Proposition 4.5, hence we provide only the modifications needed to adapt such a proof.

Assume that the first step of derivation ξ uses rule **R4** and let $\mathsf{d'} \in \mathsf{D_{i+1}}$ be the declaration used in the first step of ξ. Assume also that $\mathsf{d'} \notin \mathsf{D_i}$ and that $\mathsf{d'}$ is the result of the transformation operation applied to obtain $\mathsf{D_{i+1}}$. As usual, we distinguish various cases according to the kind of operation performed. Here we consider only those cases whose proof is different from that of Proposition 4.5, due to the fact that here we consider traces (consisting of intermediate results) rather than the final constraints.

Case 2. In this case, $\mathsf{d: H \leftarrow C[tell(\tilde{s} = \tilde{t}) \parallel B] \in D_i}$, $\mathsf{d' : H \leftarrow C[B\sigma] \in D_{i+1}}$, where σ is a relevant most general unifier of $\tilde{s}$ and $\tilde{t}$ (or a renaming, in case $\tilde{\mathsf{s}}$ and $\tilde{\mathsf{t}}$ consist of distinct variables). From the definition of the operation, we know that the variables in the domain of σ do not occur in either $\mathsf{C[\ ]}$ or H and, unlike the case of Proposition 4.5, $\mathsf{Var(B) \cap Var(H, C)} = \emptyset$.

For any derivation that uses a declaration $\mathsf{H \leftarrow C[tell(\tilde{s} = \tilde{t}) \parallel B]}$, if the agent $\mathsf{tell(\tilde{s} = \tilde{t})}$ is evaluated before B, then the proof is analogous to that, for Case 2 of Proposition 4.5. Otherwise, if the agent $\mathsf{tell(\tilde{s} = \tilde{t})}$ is not evaluated before B, then by using condition $\mathsf{Var(B) \cap Var(H, C)} = \emptyset$, we obtain that evaluation of the agent B can add to the store only constraints on variables that do not occur in either in the global store (before the evaluation of B) or in $\mathit{Var}(\mathsf{A, c_I})$. Therefore, the contribution to the global store of the agent B (before the evaluation of the agent $\mathsf{tell(\tilde{s} = \tilde{t})}$), when restricted to $\mathit{Var}(\mathsf{A, c_I})$, is equivalent to either the constraint true or to the constraint false.

In the first case, the global store is the same as the one existing before the evaluation of B. In the second case, we can obtain the constraint false by evaluating the same agents evaluated in B as in $\mathsf{B}\sigma$.

Case 3. In this case the proof is analogous to that for Case 3 of Proposition 4.5 by observing the following: If, in the derivation χ in $\mathsf{D_i}$, either the agent $\mathsf{tell(b)}$ or the agent $\mathsf{tell(\tilde{t} = \tilde{s})}$ are evaluated, then in the derivation χ' the agent $\mathsf{p(\tilde{t})}$ can be evaluated, and then we perform exactly the same steps of χ, except for the evaluation of a renamed version of the agents $\mathsf{tell(b)}$ and $\mathsf{tell(\tilde{t} = \tilde{s})}$.

Case 4. For ask simplification, the proof of Case 4 of Proposition 4.5 is simplified using Lemma A.14 and observing that, for any derivation, when the choice agent inside $\mathsf{C[]}$ is evaluated, the current store certainly implies $\mathsf{wpc(C[])}$. Hence we do not need to construct the new derivation χ'. The same holds for the tell simplification.

Case 7. In this case the proof is analogous to that for the previous Case 2, by observing that in the derivation

$$\beta = \langle D_0.C_I[C[H' \parallel tell(\tilde{X} = \tilde{X}')] \parallel tell(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_0.B'_0, c_0\rangle,$$

$Var(H') \cap Var(C_I, C, c_I, \tilde{x}, \tilde{v}, \tilde{r}) = \emptyset$. Therefore, we can construct a derivation

$$\chi_0 = \langle D_0.C_I[C[H \parallel tell(\tilde{X}' = \tilde{X})] \parallel tell(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_0.B''_0, c'_0\rangle,$$

where $tr(\beta) \preceq tr(\chi_0)$ and $m(B''_0, c'_0) = m(B'_0, c_0)$. Moreover, we can drop the constraint $tell(\tilde{x}' = \tilde{x})$, since the declarations in the derivation are renamed separately and, by construction, $Var(C_I[C[H] \parallel tell(\tilde{r} = \tilde{v})], c_I) \cap Var(\tilde{t}') = \emptyset$. We then obtain that there exists a derivation $\beta' = \langle D_0.C_I[C[H] \parallel tell(\tilde{v} = \tilde{r})], c_I\rangle \rightarrow^* \langle D_0.\overline{B}_0, \overline{c}_0\rangle$, which performs exactly the same steps of χ_0, except for (possibly) the evaluation of $tell(\tilde{t}' = \tilde{t})$ and such that $\exists_{-Var(C_I[C[H] \parallel tell(\tilde{v}=\tilde{r})], c_I)} tr(\chi_0) \preceq tr(\beta')$ and $m(\overline{B}_0, \overline{c}_0) = m(B''_0, c'_0)$. Now, the proof is the same as that for Case 7 of Proposition 4.5, since the evaluation of $tell(\tilde{x}' = \tilde{x})$ does not modify the current store with respect to the variables not in $Var(\tilde{x}')$. □

The following Lemmas are the counterpart of previous Lemmas A.3 and A.4, when considering the observable $\mathcal{O}_t(D_i.A)$.

LEMMA A.16 . *Let* $q(\tilde{r}) \leftarrow H \in D_0$ *and let* $C[\,]$ *be context. For any satisfiable constraint* c *and for any trace* s *starting with* c*, such that* $Var(C[q(\tilde{t})], c) \cap Var(\tilde{r}) = \emptyset$ *and* $w_t(C[q(\tilde{t})], s)$ *is defined, there exists a trace* s' *such that* $w_t(C[q(\tilde{r}) \parallel tell(\tilde{t} = \tilde{r})], s') \leq w_t(C[q(\tilde{t})], s)$ *and* $\exists_{-Var(C[q(\tilde{t})], c)} s \preceq \exists_{-Var(C[q(\tilde{t})], c)} s'$.

PROOF. Immediate. □

LEMMA A.17 . *Let* $q(\tilde{r}) \leftarrow H \in D_0$*. For any context* $C_I[\,]$*, any satisfiable constraint* c *and for any sequence* s *starting in* c*, the following holds:*

(1) *If* $Var(H) \cap Var(C_I, c) \subseteq Var(\tilde{r})$ *and* $w_t(C_I[q(\tilde{r})], s)$ *is defined, then there exists a sequence* s' *such that* $Var(s') \subseteq Var(C_I[H], c)$*,* $w_t(C_I[H], s') \leq w_t(C_I[q(\tilde{r})], s)$*, and* $\exists_{-Var(C_I[q(\tilde{r})], c)} s \preceq \exists_{-Var(C_I[q(\tilde{r})], c)} s'$.
(2) *If* $Var(H) \cap Var(C_I, c) \subseteq Var(\tilde{r})$*,* $Var(s) \cap Var(\tilde{r}) \subseteq Var(C_I[H], c)$ *and* $w_t(C_I[H], s)$ *is defined, then there exists a sequence* s'*, such that* $w_t(C_I[q(\tilde{r})], s') \leq w_t(C_I[H], s)$*, and* $\exists_{-Var(C_I[q(\tilde{r})], c)} s \preceq \exists_{-Var(C_I[q(\tilde{r})], c)} s'$.

PROOF. Immediate. □

Analogously to the case of the previous results, the following lemma is crucial in the proof of completeness for traces.

LEMMA A.18 . *Let* $0 \leq i \leq n$*,* $cl:\ q(\tilde{r}) \leftarrow H$ *be a declaration in* D_i *and let* cl'*:* $q(\tilde{r}) \leftarrow H'$ *be the corresponding declaration in* D_{i+1} *(in case* $i < n$*). For any context* $C_I[\,]$*, any satisfiable constraint* c *and for any sequence* s *starting in* c *the following holds*:

(1) *If* $Var(H) \cap Var(C_I, c) \subseteq Var(\tilde{r})$ *and* $w_t(C_I[q(\tilde{r})], s)$ *is defined, then there exists a sequence* s' *such that* $Var(s') \subseteq Var(C_I[H], c)$*,* $w_t(C_I[H], s') \leq w_t(C_I[q(\tilde{r})], s)$ *and* $\exists_{-Var(C_I[q(\tilde{r})], c)} s \preceq \exists_{-Var(C_I[q(\tilde{r})], c)} s'$;
(2) *If* $Var(H, H') \cap Var(C_I, c) \subseteq Var(\tilde{r})$*,* $Var(c') \cap Var(\tilde{r}) \subseteq Var(C_I[H], c)$ *and* $w_t(C_I[H], s)$ *is defined, then there exists a sequence* s' *such that* $Var(s') \subseteq Var(C_I[H'], c)$*,* $w_t(C_I[H'], s') \leq w_t(C_I[H], s)$*, and* $\exists_{-Var(C_I[q(\tilde{r})], c)} s \preceq \exists_{-Var(C_I[q(\tilde{r})], c)} s'$.

Proof. The proof is analogous to that for Lemma A.5, by using Lemma A.17 and A.16 instead of Lemma A.4 and A.3, respectively. We only have to observe the following facts:

For Case 3, Point (1), we can evaluate the agent $\mathsf{tell(b)}$ after the global store implies $\exists_{-Var(\tilde{s})}\mathsf{b}$. In this way, the new derivation has the same sequence of intermediate results.

For Case 6, Point (2), by using Lemma A.14 , if there exists a derivation

$$\xi = \langle \mathsf{D_0.C_l[C[H \parallel \textstyle\sum_{j=1}^{n} ask(c_j) \to B_j]], c} \rangle \to^* \langle \mathsf{D_0.C_m[H \parallel \textstyle\sum_{j=1}^{n} ask(c_j)} \\ \to B_j], d_m\rangle \to \langle \mathsf{D_0.C_m[H' \parallel \textstyle\sum_{j=1}^{n} ask(c_j) \to B_j], d_{m+1}} \rangle \to^* \langle \mathsf{D_0.B, d} \rangle,$$

then $\mathcal{D} \models \mathsf{d_m} \to \mathsf{e}(= \mathsf{wpc(C[\,])})$. If $\langle \mathsf{D_0.H, d_m} \rangle$ is not productive, then the proof is straightforward. Otherwise, assume that $\langle \mathsf{D_0.H, d_m} \rangle$ is productive. By definition of distribution, there exists at least one $j \in [1, n]$ such that $\mathcal{D} \models \mathsf{d_m} \to \mathsf{c_j}$ and, for each $j \in [1, n]$, either $\mathcal{D} \models \mathsf{d_m} \to \mathsf{c_j}$ or $\mathcal{D} \models \mathsf{d_m} \to \neg\mathsf{c_j}$. Then, by definition, there exists a derivation $\xi_1 = \langle \mathsf{D_0.C_m[\sum_{j=1}^{n} ask(c_j) \to (H \parallel B_j)], d_m} \rangle \to^* \langle \mathsf{D_0.B, d} \rangle$ that performs the same steps of χ_1 in the same order, except for one step in evaluating the agent $\sum_{j=1}^{n} \mathsf{ask(c_j) \to B_j}$, which is performed before evaluating the agent H. Then the thesis follows by definition of the relation $\preceq$. □

The proof of the following Lemma is also analogous to that of its previous counterpart (Lemma A.6), and hence is omitted.

Lemma A.19 . *Let* $0 \le i \le n$, $\mathsf{c_1}$ *be a satisfiable constraint and assume that there exists a derivation* ξ: $\langle \mathsf{D_i.A_1, c_1} \rangle \to^* \langle \mathsf{D_i.A_m, c_m} \rangle \to^* \langle \mathsf{D_i.A_k, c_k} \rangle$ *such that* $\mathsf{c_m}$ *is satisfiable. If*

(i) *in the first* $m - 1$ *steps of* ξ *rule* **R2** *is only used for evaluating agents of the form* $\mathsf{ask(c) \to B}$*;*

(ii) $\mathsf{w_t(A_1, tr(\xi))}$ *is defined* (*for* $\mathsf{t = m(A_k, c_k) \in \{ss, dd, pp, ff\}}$).

Then there exists a sequence $\mathsf{s'}$*, starting in* $\mathsf{c_m}$ *such that* $Var(\mathsf{s'}) \subseteq Var(\mathsf{A_m, c_m})$*,* $\exists_{-Var(\mathsf{A_1,c_1})}(\mathsf{c_m};\ldots;\mathsf{c_k}) \preceq \exists_{-Var(\mathsf{A_1,c_1})}\mathsf{s'}$ *and* $\mathsf{w_t(A_m, s') \le w_t(A_1, tr(\xi))}$.

Finally, we have the following.

Theorem 5.12 (Strong Total Correctness). *Let* $\mathsf{D_0}, \ldots, \mathsf{D_n}$ *be a restricted transformation sequence and* A *be an agent.*

—*If* $\langle \mathsf{s, x} \rangle \in \mathcal{O}_\mathsf{t}(\mathsf{D_0.A})$ *with* $\mathsf{x \in \{ss, dd, pp, ff\}}$ *then there exists* $\langle \mathsf{s', x} \rangle \in \mathcal{O}_\mathsf{t}(\mathsf{D_n.A})$*, such that* $\mathsf{s \preceq s'}$.

—*Conversely, if* $\langle \mathsf{s, x} \rangle \in \mathcal{O}_\mathsf{t}(\mathsf{D_n.A})$*, then there exists* $\langle \mathsf{s', x} \rangle \in \mathcal{O}_\mathsf{t}(\mathsf{D_0.A})$ *such that* $\mathsf{s \preceq s'}$.

Proof. The proof is analogous to that for Theorem 4.13, and proceeds by showing simultaneously, by induction on i, that for $i \in [0, n]$ and for any agent A:

(1) $\mathcal{O}_\mathsf{t}(\mathsf{D_0.A}) \equiv \mathcal{O}_\mathsf{t}(\mathsf{D_i.A})$;

(2) $\mathsf{D_i}$ is weight-complete for the traces.

The proof of the base case is analogous to that for the base case of Theorem 4.13, and hence is omitted. For the induction step, we have that, by induction hypothesis, for any agent A, $\mathcal{O}_t(D_0.A) \equiv \mathcal{O}_t(D_{i-1}.A)$ and D_{i-1} is weight-complete for the traces. Proposition 4.9 also holds when considering $\mathcal{O}_t$ rather than $\mathcal{O}$. From Proposition A.15 and (the counterpart for traces of) Proposition 4.9, it follows that if D_i is weight-complete for traces then, for any agent A, $\mathcal{O}_t(D_0.A) = \mathcal{O}_t(D_i.A)$. So in order to prove parts 1 and 2, we have only to show that, for any derivation $\beta = \langle D_0.A, c_I\rangle \rightarrow^* \langle D_0.B, c_F\rangle$ such that c_I is a satisfiable constraint and $m(B, c_F) \in \{ss, dd, pp, ff\}$, there exists a split derivation in $D_i \cup D_0$, $\xi = \langle D_i.A, c_I\rangle \rightarrow^* \langle D_0.B', c'_F\rangle$ such that $tr(\beta) \preceq tr(\xi)$ and $m(B', c'_F) = m(B, c_F)$.

From the inductive hypothesis, it follows that there exists a split derivation

$$\chi = \langle D_{i-1}.A, c_I\rangle \rightarrow^* \langle D_0.B'', c''_F\rangle,$$

where $tr(\beta) \preceq tr(\chi)$ and $m(B'', c''_F) = m(B, c_F)$. Now let $d \in D_{i-1} \backslash D_i$ be the modified clause in the transformation step from D_{i-1} to D_i. The rest of the proof is essentially analogous to that for Theorem 4.13. The only points that require some explanation are the following:

Case 2. In this case, the proof is analogous to that for Case 2 of Proposition A.15 .

Case 3. In this case the proof is analogous to that for Case 3 of Proposition A.15 , provided we observe the following fact for case (2a):

Given any set of declarations, if there exists a derivation χ' for the configuration $\langle C'[tell(\bar{b}) \parallel \bar{B} \parallel tell(\tilde{t} = \tilde{s}')], c'\rangle$ where c' is satisfiable and $Var(C'[tell(\bar{b}) \parallel \bar{B} \parallel tell(\tilde{t} = \tilde{s}')], c') \cap Var(b, \tilde{s}) = \emptyset$, then there exists a derivation for $\langle C'[tell(\bar{b}) \parallel \bar{B} \parallel tell(\tilde{t} = \tilde{s}') \parallel tell(b) \parallel tell(\tilde{t} = \tilde{s})], c'\rangle$, which performs the same steps of χ' plus (possibly) two steps corresponding to the evaluation of $tell(\tilde{t} = \tilde{s})$ and $tell(b)$, after the evaluation of $tell(\bar{b})$ and $tell(\tilde{t} = \tilde{s}')$.

Case 4. Analogously to the proof of Case 4 of Proposition A.15 , it is sufficient to observe the following. From Lemma A.14 it follows that, for any derivation, when the choice agent inside a context C[ ] is evaluated, the current store implies wpc(C[ ]). Then, by definition of ask simplification, the constraints c_j and c'_j are equivalent with respect to the current store (and therefore we do not need to construct the new derivation χ'). The same reasoning applies to the case of tell simplification.

Case 6. The proof is analogous to that for Case 6 of Lemma A.18 . □

ACKNOWLEDGMENTS

We would like to thank the reviewers for their precise and helpful comments.

REFERENCES

Bensaou, N. and Guessarian, I. 1998. Transforming constraint logic programs. *Theor. Comput. Sci. 206,* 1–2, 81–125.

Boer, F. S. D., Gabbrielli, M., Marchiori, E., and Palamidessi, C. 1997. Proving concurrent constraint programs correct. *ACM Trans. Program. Lang. Syst. 19,* 5, 685–725.

Burstall, R. and Darlington, J. 1977. A transformation system for developing recursive programs. *J. ACM 24,* 1 (Jan.), 44–67.

Clark, K. and Sickel, S. 1977. Predicate logic: a calculus for deriving programs. In *Proceedings of IJCAI'77*. 419–120.

Codish, M., Falaschi, M., and Marriott, K. 1994. Suspension analyses for concurrent logic programs. *ACM Trans. Program. Lang. Syst. 16,* 3, 649–686.

de Boer, F. and Palamidessi, C. 1991. A fully abstract model for concurrent constraint programming. In *TAPSOFT/CAAP*. S. Abramsky and T. Maibaum, eds. LNCS 493, Springer Verlag.

de Boer, F. and Palamidessi, C. 1992. On the semantics of concurrent constraint programming. In *ALPUK 92, Workshops in Computing*, Springer-Verlag, 145–173.

Etalle, S. and Gabbrieli, M. 1998. Partial evaluation of concurrent constraint languages. *ACM Comput. Surv. 30,* (Sept.).

Etalle, S. and Gabbrielli, M. 1996. Transformations of CLP modules. *Theor. Comput. Sci. 166,* 1, 101–146.

Etalle, S., Gabbrielli, M., and Meo, M. C. 1998. Unfold/fold transformations of CCP programs. In *CONCUR98—1998 International Conference on Concurrency Theory*, D. Sangiorgi and R. de Simone, eds. LNCS 1466, Springer Verlag, 348–363.

Francesco, N. D. and Santone, A. 1996. Unfold/fold transformation of concurrent processes. In *Proceeding of the. 8th International Symposium. on Programming Languages: Implementations, Logics and Programs*, H. Kuchen and S. Swierstra, eds. Vol. 1140, Springer Verlag, 167–181.

Gengler, M. and Martel, M. 1997. Self-applicable partial evaluation for pi-calculus. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM-97). ACM SIGPLAN Not. 32*, 12. ACM Press, New York, 36–46.

Hogger, C. 1981. Derivation of logic programs. *J. ACM 28,* 2 (April), 372–392.

Hosoya, H., Kobayashi, N., and Yonezawa, A. 1996. Partial evaluation scheme for concurrent languages and its correctness. In *Euro-Par'96*, LNCS 1123, Springer Verlag, 625–632.

Jaffar, J. and Maher, M. J. 1994. Constraint logic programming: A survey. *J. Logic Program. 19/20*, 503–581.

Jørgensen, N., Marriott, K., and Michaylov, S. 1991. Some global compile-time optimizations for CLP($\mathcal{R}$). In *Proceeding of the International Logic Programming Symposium*, V. Saraswat and K. Ueda, eds. MIT Press, Cambridge, MA, 420–434.

Kawamura, T. and Kanamori, T. 1988. Preservation of stronger equivalence in unfold/fold logic programming transformation. In *Proceeding of the Internatinal Conference on Fifth Generation Computer Systems*. Institute for New Generation Computer Technology, Tokyo, 413–422.

Komorowski, H. 1982. Partial evaluation as a means for inferencing data structures in an applicative language: A theory and implementation in the case of Prolog. In *Proceedings of the Ninth ACM Symposium on Principles of Programming Languages* (Albuquerque, NM). ACM Press, New York, NY, 255–267.

Lloyd, J. W. 1987. *Foundations of Logic Programming*. 2dn ed. Springer Verlag, Berlin.

Maher, M. 1993. A transformation system for deductive databases with perfect model semantics. *Theor. Comput. Sci. 110,* 2 (March), 377–403.

Marinescu, M. and Goldberg, B. 1997. Partial-evaluation techniques for concurrent programs. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM-97). ACM SIGPLAN Not. 32*, 12. ACM Press, New York, NY, 47–62.

Milner, R. 1989. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, NJ.

Mogensen, T. and Sestoft, P. 1997. Partial evaluation. In *Encyclopedia of Computer Science and Technology*, A. Kent and J. Williams, eds. Vol. 37. Marcel. Dekker, 247–279.

Pettorossi, A. and Proietti, M. 1994. Transformation of logic programs: Foundations and techniques. *J. Logic Program. 19,* 20, 261–320.

Sahlin, D. 1995. Partial evaluation of AKL. In *Proceedings of the First International Conference on Concurrent Constraint Programming*.

Saraswat, V. and Rinard, M. 1990. Concurrent constraint programming. In *Proceeding of the 17th ACM Symposium on Principles of Programming Languages*. ACM Press, New York, NY, 232–245.

Saraswat, V., Rinard, M., and Panangaden, P. 1991. Semantics foundations of concurrent constraint programming. In *Proceeding of the 18th Annual ACM Symposium on Principles of Programming Languages*. ACM Press, New York, NY.

SARASWAT, V. A. 1989. Concurrent constraint programming languages. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA.

SMOLKA, G. 1995. The Oz programming model. In *Computer Science Today*, J. van Leeuwen, ed. LNCS 1000, Springer Verlag; see www.ps.uni-sb.de/oz/.

TAMAKI, H., AND SATO, T. 1984. Unfold/fold transformations of logic programs. In *Proceedings of the Second International Conference on Logic Programming*, S.-A. ke Tärnlund, ed. 127–139.

UEDA, K. 1986. Guarded Horn clauses. In *Logic Programming '85*, E. Wada, ed. LNCS 221, Springer Verlag, Berlin, 168–179.

UEDA, K. AND FURUKAWA, K. 1988. Transformation rules for GHC programs. In *Proceedings of the Internatinal Conference on Fifth Generation Computer Systems*. Institute for New Generation Computer Technology, Tokyo, 582–591.

Received January 1999; revised December 2000; accepted April 2001