Elsevier Editorial System(tm) for Journal of Logical and Algebraic Methods in Programming
Manuscript Draft

Manuscript Number: JLAMP-D-16-00036R1

Title: Observational and Behavioural Equivalences for Soft Concurrent Constraint Programming

Article Type: SI: MeMo

Keywords: Soft Concurrent Constraint Programming
Fair Computations
Obervational Semantics
Behavioural Equivalences

Corresponding Author: Professor Fabio Gadducci, Ph.D.

Corresponding Author's Institution: University of Pisa

First Author: Fabio Gadducci, Ph.D.

Order of Authors: Fabio Gadducci, Ph.D.; Francesco Santini, Ph.D.; Luis Pino, Ph.D.; Frank Valencia, Ph.D.

Abstract: We present a labelled semantics for Soft Concurrent Constraint Programming (SCCP), a meta-language where concurrent agents may synchronise on a shared store by either posting or checking the satisfaction of (soft) constraints. SCCP generalises the classical formalism by parametrising the constraint system over an order-enriched monoid, thus abstractly representing the store with an element of the monoid, and the standard unlabelled semantics just observes store updates. The novel operational rules are shown to offer a sound and complete co-inductive technique to prove the original equivalence over the unlabelled semantics. Based on this characterisation, we provide an axiomatisation for finite agents.

GENERAL REPLY

Dear Marino Miculan,

Following your encouragement to resubmit our manuscript, we are happy to send you a revised version of the article "*Observational and Behavioural Equivalences for Soft Concurrent Constraint Programming*" submitted to *Journal of Logical and Algebraic Methods in Programming* (JLAMP), originally submitted for a special issue of the *2nd International Workshop on Meta Models for Process Languages*.

We received two reviews that globally agree on the interest of studying observational and behavioural equivalence relations for the deterministic fragment of Soft Concurrent Constraint Programming. However, the reviewers have raised different concerns we here summarise: the first reviewer basically asks to improve the presentation, yet s/he finds that the technical results in the paper are interesting and sound. Also the main concern of the second reviewer is related to the presentation, in particular to the need of more relevant and better placed examples, and we hope that the new version improves the situation.

In the rest of this document we reply to each and every issue raised by the two reviewers, going into the details of their comments and suggestions. We hope that you will find this revision suitable for publication in JLAMP. We are confident that we properly answered to the questions and doubts legitimately raised by the reviewers.

We would really like to thank the reviewers for their thoughtful and very useful feedback on the submitted version of our article. Please let us know if you need any further information.

Sincerely yours,

Fabio Gadducci, Francesco Santini, Luis F. Pino, Frank Valencia

REVIEW #2

The paper presents a labeled operational semantics for the Soft Concurrent Constraint Programming Language (SCCP), a variant of the Concurrent Constraint Paradigm that can distinguish how many times a piece of information occurs in the store (no idempotency). The authors also introduce an axiomatization for a finite fragment of the language and define the notions of weak and strong bisimilarity to reason about the behavior of SCCP programs.

This is the journal version of a conference paper of the same authors, presented in COORDINATION 2015.

In my opinion, the paper needs some major changes to be considered for publication in JLAMP.
First of all, I found it difficult to follow and with several imprecisions, inconsistencies, and errors.
Moreover, the new contributions are minimal with respect to the conference paper, which is much better written and organized.

I don't have major concerns about the soundness of the results, but I think that the presentation should be seriously improved to make the paper acceptable.
In the following, I give some suggestions to improve the readability of the paper.

- page 9, definition 12: give an intuition of what each agent does, since it is not clear for people not familiar with the CCP paradigm
***
We added the an explanatory paragraph below Definition 12, in order to introduce (S)CCP.
***

- page 10, definition 14: give an intuition about what \Delta is for (what role these variables have in the transition?)
***
We added an explanatory paragraph below Definition 14.
***

- page 17: it is not clear the connection between definition 17 and definition 18. Maybe here an example would help.
***
We added an example, as well as tried to explain better the rational for paths.
***

There is also a little bit of confusion between the notions of active path and active agent that should be clarified.
***
Indeed, and we apologize for that. We hope that with the example and an extended discussion, things are clearer now.
***

- page 18, line 28: explain why the weak bisimilarity is unsatisfactory.
***

In a nutshell, any proof of the equivalence of two agents should consider all the possible stores, since R is a bisimulation for agents A and B if all the configurations <A,c> and <B,c> are related by R. This forces a cumbersome proof technique. Mutatis mutandis, it is like requiring that an equivalence on e.g. nets or processes has to be a congruence, instead of proving that it is so.
***

- page 23, example 3: give an intuition on the reason why the agents considered are weakly bisimilar.
***

We tried and gave an intuitive explanation before moving to the formal proof.
***

- page 25, figure 3, axiom 3: I think that the right agent should be ask(c \wedge d) -> A since both have to be entailed by the store for A to be executed.
***

Indeed, but this is why we need to require that their union is entailed by the store.
***

- page 25, section 7: maybe this part would be more useful near the definition of the language SCCP
***

Indeed, we reformulated and simplified the former section 7, which is now subsection 2.4. This way, it fleshes out some examples about CLIM, cylindrification, etc.
***

- page 28, section 8: why don't move these examples near tables 1 and 2? I think that at page 28 these examples are a little bit out of context and they are not useful anymore. On the contrary, moving them close to the tables would clarify to the reader the meaning of the rules and make them more understandable.
***

We followed the advice of the reviewer: now the examples in the former section 8 have been inserted inside section 3. We also believe that the calculus is easier to make sense of now.
***

- the examples at pages 28-29 are not peculiar to the Soft version of CCP, it would be nice to have an example of a program that behaves differently in SCCP w.r.t CCP.
***

CCP is just SCCP with a specific kind of CLIM, namely, one where the monoidal operator is idempotent, such as the positive real numbers with min. In order to differentiate, in the examples we used the CLIM of real numbers in the interval [0,1] with multiplication and the one of positive real numbers with addition. We could provide specific examples of agents and compare them in the different stores, but we believe that it would not be too informative. It is the fact that our treatment works for not necessarily idempotent CLIMs that is relevant.
***

The idea to consider local stores in the operational semantics would be an interesting contribution to make the paper more interesting and complete, and to add a substantial contribution w.r.t. the conference version, leaving the handle of non-determinism (much more challenging) to a future paper.
***
Unfortunately, the introduction of local variables is much less obvious than expected, since also in this case the semantics is still monotone, but the observables are not well defined even for fair computations. Instead, in this paper we wanted to show how in the classic cases we could recast all the trappings of the crisp version, and we preferred to stick to the deterministic case. However, we believe that the examples and the amount of technical insights in the current version of the paper (with respect to the conference one) represent an interesting addition to the soft CCP lore.
***

MINOR COMMENTS:

- page 4, definition 2: what is Y?
***
We added "Y \subseteq A" in the definition.
***

- the agent stop sometimes is called 0, it would be better to call it always in one way or the other (for example at page 9, 13, 24 ...)
***
We now only use stop.
***

- page 11 line 35: fn -> fv
***
We fixed it as suggested.
***

- page 12, line 13: what is \varepsilon? I don't think it has been introduced before
***
We now say before Definition 15 that \varepsilon is a (possibly infinite) computation.
***

- page 13, line 46: is -> in
***
We fixed it as suggested.
***

- page 15, definition 22: c in C^\otimes is missing
***
We added it in the first requirement (and in the one of Definitions 24, 27 and 28 as well).

***

- page 16, definition 23: what is gamma? Maybe \gamma = <A,sigma> is missing
***
We fixed it as suggested.
***

- page 17, proof 7: lines 24-30 are used for both directions of the proof
***
We apologize, but we are not sure we understood the remark.
***

- page 20, definition 27: sometimes the notation <A, sigma>/<B, rho> is used and others the \gamma notation is used instead. It would be better to be consistent. For example in the second point (line 36) it would be clearer to write <A, sigma> ->^alpha gamma_1' and <B, rho> ->^alpha gamma_2'.
***
Our motivation was to try and be as little cumbersome as possible. We now added the new definition \gamma \otimes \alpha that should improve some definitions here and there.
***

- lemmata 12, 13, 14: C -> \cal{C}
***
We fixed it as suggested.
***

- lemmata 12, 13: a point is missing before "Moreover"
***
We fixed it as suggested.
***

- theorem 3, 4: there is an additional \subseteq
***
We fixed it by removing the additional \subseteq.
***

- page 29, example 7: the arrow of the ask agent is too long (not the same as the definition at page 9)
***
We fixed it: now it is shorter.
***

REVIEW #4

The paper presents a labeled semantics for SCCP and the prove that it is equivalent to the unlabeled semantics by using a bisimulation technique.

In my opinion the paper is very technical and certainly not an easy read. Obviously, this is due to the nature of the findings of the authors. I doubt that this paper is very significant since it is dealing with an extension of a (formal) language that has not seen any real implementation, it is deterministic, and usually assumes the possibility to revert the introduction of a constraint (which in practice is often a very computationally expensive operation). Moreover, the new results, based on co-inductive proofs, can tell us something more than inductive techniques only for non terminating SCCP system, whose need is not very well motivated by the authors.
***
We agree on the relevance of our techniques for non-terminating system; as for the usefulness of such systems, there is already quite some literature on their use in crisp CCP, and we hope that the example we provided can offer a glimpse of it. In general, our paper is the first to provide a labeled semantics for soft CCP, adding many new concepts (such as \otimes-compactness) to the SCCP lore.
***

This paper builds upon a paper presented at COORDINATION and other papers in the literature. It is unclear to me if the additional contribution constitutes the usual 20/30% extension that usually justifies a journal paper. I therefore let the editor to decide if the paper presents enough new material to justify a journal publication.

Said so, even though I believe that the significance is very low, I recognize that the authors did a big formalization and proving effort that has some merit in it. For this reason, even if probably few people are interested in such a work, I believe that this paper deserve to be accepted, provided that the authors address some of my major comments detailed below.

Even though I did not go through all of the proofs, from the technical point of view I did not spot any major problem. Since I believe that few readers will be interested in going to this level of technicalities in this review I am more focusing on the presentation that, if improved, may allow the access to the paper also to non SCCP expert readers.

MAJOR COMMENTS:

One of the weaknesses of this paper is that the authors did not try to justify the usefulness of SCCP. I believe that the (part of) Section 7 should be moved earlier (even before the background) to allow a casual reader to appreciate the SCCP and allow him/her to better understand what we are dealing with. I believe that an introduction of SCCP with examples is far more easy to understand than a formal definition. In particular, since we are dealing with co-inductive techniques, it could be nice to have more significant non terminating examples. Section 3 and Section 7, if merged together, can improve the understanding of the SCCP language (right now the paper is structured to be read only by experts in the field).

***

As remarked for the other reviewer, we decided to shorten and simplify the former section 7, which is now subsection 2.4, thus using it to make a complete example about CLIMs, cylindrification, etc.

***

For the presentation sake some notations are introduced without giving an intuition or an example. It may be far better for a reader to have examples, whenever possible, to help understanding the notation. I would recommend the authors to consider every definition individually and ask themselves if they can write an example to help the reader to better understand the definition. For instance, an example of Cylindrification and Diagonalisation (Definitions 9 and 10) could be added.

***

We hope that the problem concerning the examples for section 2 is solved with the introduction of subsection 2.4. As for Section 3, we inserted the examples contained in the former section 8.

***

In Definition 18 the double arrow is introduced but not defined before. I have problems in understanding this definition. Clearly an example here could greatly improve the presentation. I am also puzzled by the definition of fairness. How this definition of fairness relates to the ones usually adopted in process algebra calculi (e.g., the paper [1] that you also cite)? Here is seems that this definition of fairness is just something you would like to have to derive confluence. Since fairness is something you require for your results I believe that a better explanation and a richer discussion need to be added here.

***

The double arrow was a mistake: we meant the arrow for the axioms in Table 1. We hope now that by correcting the mistake, reshaping the structure of the section and adding examples and a new section the whole proposal has been clarified enough. We remark that e.g. in POPL'90 the authors simply write that <<We say that a sequence S is fair if it eventually reduces every subagent that can be reduced at some stage of S. This is a common notion that one needs in defining the operational semantics of concurrent systems>>. We just tried and were more formal, identifying precisely what it means that a subagent <<can be reduced at some stage>>, by the notion of enabled path, and what it means that a sequence <<eventually reduces>> it, by the notion of active path. As for [1], the author there simply writes that <<an execution sequence S is weakly fair if no subagent remains enabled continuously from some point on. S is strongly fair if no subagent remains enabled infinitely often>>. A definition that is somehow loose, as we argued briefly in the new section, also hinting at an additional paper [2], where the issue is fully carried out.

***

I invite the authors to add a related work section. Even though you only focus on SCCP, there are other formalisms related to it, albeit not as general as SCCP. For instance it could be interesting to see the similarities of SCCP and the rule base Constraint Handling Rules language (both deal with a constraint store where agents/rules can fire in parallel), tuple based languages such as Linda, or even Distributed Constraint Programming where CSPs or COPs are solved by a network of nodes. Could be nice also to have a discussion on the limitation of SCCP (e.g., the unique global store that could

become a bottleneck when there are a lot of agents accessing it, the importance of determinism, the invertibility, fairness, ...).
***

We added a new section with related work (new Section 8). We included similar approaches related to CCP and SCPP languages, and we described how labelled tuple-based languages (ad KLAIM and Linda) could benefit from the framework we define in our paper, which is based on the minimal information to trigger an action.
***

MINOR COMMENTS:

Pag 10, line 54: typo ":" instead of "."
***

We fixed it as suggested.
***

Pag 13, line 46 "is A" -> typo
***

We fixed it as “a path in A”.
***

Pag 22, line 50 "and" -> typo
***

We removed “and” and we fixed it as “Theorem~\ref{stronEq} by”.
***

Citation 5 and 6 are the same.
***

We moved one of the two.
***

Is Lemma 14 really needed? The paper already has a lot of lemmas and definitions. If one can be avoided probably it is better to do so.
***

The lemma (now Lemma 17 due to the reshuffling of part of the material in Section 2) just proves that weak bisimilarity is a congruence. We could add the proof of Theorem 4 (concerning the equivalence of the two semantics) and insert the result directly there, but it seems to us that such modularity makes clearer to the reader the steps leading to the theorem.
***

[1] Marta Kwiatkowska. Infinite behaviour and fairness in concurrent constraint programming.
[2] Sven-Olof Nystrom and Bengt Jonsson. A Fully Abstract Semantics for Concurrent Constraint Programming.

Observational and Behavioural Equivalences for Soft Concurrent Constraint Programming

Fabio Gadducci[a,*], Francesco Santini[b,1,**], Luis F. Pino[c], Frank D. Valencia[d]

[a]*Dipartimento di Informatica, Università di Pisa, Italy*
[b]*Dipartimento di Matematica e Informatica, Università di Perugia, Italy*
[c]*Dipartimento di Matematica e Informatica, Università di Cagliari, Italy*
[d]*CNRS and LIX, École Polytechnique de Paris, France*

Abstract

We present a labelled semantics for Soft Concurrent Constraint Programming (SCCP), a meta-language where concurrent agents may synchronise on a shared store by either posting or checking the satisfaction of (soft) constraints. SCCP generalises the classical formalism by parametrising the constraint system over an order-enriched monoid, thus abstractly representing the store with an element of the monoid, and the standard unlabelled semantics just observes store updates. The novel operational rules are shown to offer a sound and complete co-inductive technique to prove the original equivalence over the unlabelled semantics. Based on this characterisation, we provide an axiomatisation for finite agents.

1. Introduction

Concurrent Constraint Programming (CCP) [1] is a language based on a shared-memory communication pattern: processes may interact by either posting or checking partial information, which is represented as constraints in a global store. CCP belongs to the larger family of process calculi, thus a syntax-driven operational semantics represents the computational steps.

*Principal corresponding author.
**Secondary corresponding author.
Email addresses: fabio.gadducci@unipi.it (Fabio Gadducci), francesco.santini@dmi.unipg.it (Francesco Santini), luis.pino@unica.it (Luis F. Pino), frank.valencia@lix.polytechnique.fr (Frank D. Valencia)

For example, the term **tell**(c) represents a process that posts c in the store, and the term **ask**(c) $\rightarrow$ P is the process that executes P if c can be derived from the information in the store.

The formalism is parametric with respect to the entailment relation. Under the name of *constraint system*, the information recorded on the store is structured as a partial order (in fact, a lattice) $\leq$, where $c \leq d$ means that c can be derived from d. Under a few requirements over such systems, CCP has been provided with (coincident) operational and denotational semantics. Recently, a labelled semantics has also been provided, and the associated weak bisimilarity proved to coincide with the original semantics [2].

A key aspect of CCP is the *idempotency* of the operator for composing constraints: adding the same information twice does not change the store. On the contrary, the soft variant of the formalism (Soft CCP, or just SCCP [3]) drops idempotency: constraint systems in SCCP may distinguish the number of occurrences of a piece of information. Dropping idempotency requires a complete reworking of the theory. Although an operational semantics for SCCP has been devised [3], hitherto neither the denotational nor the labelled one has been reintroduced. This is unfortunate since due to its generality, suitable SCCP instances has been successfully applied as a specification formalism for negotiation of Service Level Agreements [4], or the enforcement of ACL-based access control [5].

As a language, SCCP has been used as a specification formalism for agents collaborating via a shared knowledge basis, possibly with temporal features [6, 7]. Thus, on a methodological level, the development of behavioural equivalences for SCCP may result in the improvement on the analysis techniques for agents that need to reason guided by their preferences, more so if their knowledge is not complete.

In more general terms, SCCP represents, by its parametric nature, a formal meta-model where to develop different constraint-languages over different (weighted or crisp) logics, as it will clearly appear from Section 3.

The work in [1] establishes a denotational semantics for CCP and an equational theory for infinite agents. More recently, in [2] the authors prove that the axiomatisation is underlying a specific weak bisimilarity among agents, thus providing a clear operational understanding. The key ingredients are a complete lattice as the domain of the store, with least upper bound for constraint combination, and a notion of compactness such that domain equations for the parallel composition of recursive agents would be well-defined. On the contrary, the soft version [3] drops the upper

bound for combination in exchange of a more general monoidal operator. Thus, the domain is potentially just a (not necessarily complete) partial order, possibly with finite meets and a residuation operator (a kind of inverse of the monoidal one) in order to account for algorithms concerning constraint propagation. Indeed, the main use of SCCP has been in the generalisation of classical constraint satisfaction problems, hence the lack of investigation about e.g compactness and denotational semantics.

The objective of our work is the development of a general theory for the operational semantics of SCCP, via the introduction of suitable observational and behavioural equivalences. Reaching this objective is technically challenging, since most of the simplicity of CCP is based precisely on the premise that posting an information multiple times is the same as posting it only once. The first step consists in recasting the notion of compactness from crisp to soft; we then introduce a novel labelled semantics for SCCP which will allow us to give a sound and complete technique to prove the equivalence over the unlabelled semantics.

We will build our framework by supposing to have a global store, that is shared by all the agents. Such a premise is required by how we design the transition system: in fact, it is labelled with a shared set of variables (i.e., Δ) as a means to keep track of variables' name after renaming them through the hiding operator. Since variables support the constraints posted to the store, renaming is more subtle than in other algebras: the store retains a memory of former names. In the future we plan to investigate how hiding can be performed in case of stores local to each agent (see Section 8).

This paper details the work in [8] and extends it by reconnecting the presented framework with the classical work on soft constraint systems [3] (see Section 2.4). Section 2 opens the paper with the technical background, presenting also some novelty as $\otimes$-compact elements. Section 3 presents the semantics of a deterministic fragment of a constraint language, together with fundamental concepts, e.g., observables, confluence, observational equivalence, and, in Section 4, the notion of saturated bisimulation. Section 5 derives a labelled transition system for SCCP, soundness and completeness with respect to the unlabelled one, and weak/strong bisimilarity relations. Section 6 shows a sound and complete axiomatisation for a finite fragment of the language. Finally, Section 7 reports the main literature in the field, while Section 8 wraps up the paper with conclusions and future work.

2. The Algebraic Background

This section recalls the main notions we are going to need later on. First of all, we present some basic facts concerning monoids [9] enriched over complete lattices. These are used to recast the standard presentation of the soft constraints paradigm, and to generalise the classical crisp one.

2.1. Lattice-enriched Monoids

Definition 1 (Complete lattices). *A partial order (PO) is a pair $\langle A, \leq \rangle$ such that A is a set of values and $\leq \subseteq A \times A$ is a reflexive, transitive, and anti-symmetric relation. A complete lattice (CL) is a PO such that any subset of A has a least upper bound (LUB).*

We denote as $\bigvee X$ the necessarily unique LUB of a subset $X \subseteq A$, and explicitly $\bot$ and $\top$ if we are considering the empty set and the whole A, respectively: the former is the bottom and the latter is the top of the PO. Obviously, CLs also have the greatest lower bound (GLB) for any subset $Y \subseteq A$, denoted as $\bigwedge Y$. In the following we fix a CL $\mathbb{C} = \langle A, \leq \rangle$.

Definition 2 (Compact elements). *An element $a \in A$ is compact if whenever $a \leq \bigvee Y$ for some $Y \subseteq A$ there exists a finite subset $X \subseteq Y$ such that $a \leq \bigvee X$.*

Note that for complete lattices the definition of compactness given above coincides with the one using directed subsets. It will be easier to generalise it, though, to compactness with respect to the monoidal operator (see Definition 6). We let $A^C \subseteq A$ denote the set of compact elements of $\mathbb{C}$.

Example 1. *Note that A^C might be trivial. Consider for instance the CL $\langle [0, 1], \geq \rangle$ (the segment of the reals with the inverse of the usual order), used for probabilistic constraints [10]: only the bottom element 1 is compact. As we will see, the situation for the soft paradigm is more nuanced.*

Definition 3 (Monoids). *A commutative monoid with identity (IM) is a triple $\langle A, \otimes, \mathbf{1} \rangle$ such that $\otimes : A \times A \to A$ is a commutative and associative function and*

(identity) $\forall a \in A. \otimes (a, \mathbf{1}) = a$

We will often use an infix notation, such as $a \otimes b$ for $a, b \in A$. The monoidal operator can be defined for any multi-set: it is given for a family of elements $a_i \in A$ indexed over a finite, non-empty set I, and it is denoted by $\bigotimes_{i \in I} a_i$. If for an index set I the a_i's are different, we write $\bigotimes S$ instead of $\bigotimes_{i \in I} a_i$ for the set $S = \{a_i \mid i \in I\}$. Conventionally, we denote $\bigotimes \emptyset = \bot$.

We now move our attention to our choice for the domain of values.

Definition 4 (CL-enriched IMs). *A CL-enriched IM (CLIM) is a triple* $\mathbb{S} = \langle A, \leq, \otimes \rangle$ *such that* $\langle A, \leq \rangle$ *is a CL,* $\langle A, \otimes, \bot \rangle$ *is an IM, and*

(distributivity) $\forall a \in A. \forall X \subseteq A. a \otimes \bigwedge X = \bigwedge \{a \otimes x \mid x \in X\}$

Remark 1. *The reader who is familiar with the soft constraint literature may have noticed that we have basically rewritten the standard presentation using a CLIM instead of an absorptive semiring, recently popularised as c-semiring [11], where the* $a \oplus b$ *operator is replaced by the binary LUB* $a \vee b$*. Besides what we consider a streamlined presentation, the main advantage in the use of CLIMs is the easiness in defining the LUB of infinite sets and, as a consequence, the notion of* $\otimes$*-compactness given below. An alternative solution using infinite sums can be found in [12, Section 3], and a possible use is sketched in [13].*

Thanks to distributivity, we can show that $\otimes$ is monotone, and since $\bot$ is the identity of the monoid, monotonicity implies that the combination of constraints is increasing, i.e., $\forall a, b \in A. a \leq a \otimes b$ holds. Finally, we recall that by definition $\bigwedge \emptyset = \top$, so that $\forall a \in A. a \otimes \top = \top$ also holds.[2]

In the following, we fix a CLIM $\mathbb{S} = \langle A, \leq, \otimes \rangle$. The next step is to provide an infinite composition. Our definition is from [9] (see also [12, p. 42]).

Definition 5 (Infinite composition). *Let I be a (countable) set of indexes. Then, the composition* $\bigotimes_{i \in I} a_i$ *is defined as* $\bigvee_{J \subseteq I} \bigotimes_{j \in J} a_j$ *for all finite subsets J.*

Should I be finite, the definition gives back the usual multi-set composition, since $\otimes$ is monotone and increasing. As the infinitary composition is also monotone and increasing, by construction $\bigotimes A = \bigvee A = \top$. We now provide a notion of compactness with respect to the monoidal operator.

[2] A symmetric choice $\langle A, \otimes, \top \rangle$ with distributivity with respect to $\bigvee$ (and thus $a \otimes \bot = \bot$) is possible: the monoidal operator would be decreasing, so that for example $a \otimes b \leq a$. Indeed, this is the usual order in the semiring-based approach to soft constraints [13].

Definition 6 (⊗-compact elements). *Let $a \in A$. It is ⊗-compact if whenever $a \leq \bigotimes_{i\in I} a_i$ then there exists a finite subset $J \subseteq I$ such that $a \leq \bigotimes_{j\in J} a_j$.*

We let $A^{\otimes} \subseteq A$ denote the set of ⊗-compact elements of $\mathbb{S}$. It is easy to show that a compact element is also ⊗-compact, i.e., $A^{C} \subseteq A^{\otimes}$. Indeed, the latter notion is definitively more flexible.

Example 2. *Consider the CLIM $\langle [0,1], \geq, \times\rangle$ examined above, which corresponds to the segment of the reals with the inverse of the usual order and multiplication as monoidal product. Since any infinite multiplication tends to 0, then all the elements are ⊗-compact, except the top element itself, that is, precisely 0.*

Remark 2. *It is easy to show that idempotency implies that $\bigotimes$ coincides with LUBs, that is, $\bigotimes S = \bigvee S$ for all subsets $S \subseteq A$. In other words, the whole soft structure collapses to a complete distributive lattice. Indeed, requiring distributivity makes the soft paradigm not fully comparable with the crisp one. We are going to discuss it again in the concluding remarks.*

2.2. Residuation

Our first operator on CL-enriched IMs is a simple construction for a weak inverse of the monoidal operator in CL-enriched monoids, known in the literature on enriched monoid as residuation [12, 14].

Definition 7 (Residuation). *Let $a, b \in A$. The residuation of a with respect to b is defined as $a \oslash b = \bigwedge\{c \in A \mid a \leq b \otimes c\}$.*

The definition conveys the intuitive meaning of a division operator: indeed, $a \leq b \otimes (a \oslash b)$, thanks to distributivity. Also, $(a \otimes b) \oslash b \leq a$ and $a \oslash (b \otimes c) = (a \oslash b) \oslash c$. Residuation is monotone on the first argument: if $a \leq b$ then $a \oslash c \leq b \oslash c$ and $a \oslash b = \bot$. For more properties of residuation we refer to [15, Table 4.1].

Most important for our formalism is the following result on ⊗-compactness.

Lemma 1. *Let $a, b \in A$. If a is ⊗-compact, so is $a \oslash b$.*

Proof 1. *If $a \oslash b \leq \bigotimes_{i\in I} a_i$, then by monotonicity $a \leq \bigotimes_{i\in I\uplus\{*\}} a_i$ for $a_* = b$. By ⊗-compactness of a there exists a finite $J \subseteq I$ such that $a \leq \bigotimes_{j\in J\uplus\{*\}} a_j$, and by the definition of division $a \oslash b \leq \bigotimes_{j\in J} a_j$, hence the result holds.*

Most standard soft instances (boolean, fuzzy, probabilistic, weighted, and so on) are described by CL-enriched monoids and are residuated: see e.g., [13]. For these instances the $\oplusminus$ operator is used to (partially) remove constraints from the store, and as such is going to be used in Section 5. In fact, in the soft literature it is usually required a tighter relation of (full) invertibility, also satisfied by all the previous CLIMs instances, stated in our framework by the definition below.

Definition 8. *$\mathbb{S}$ is invertible if $b \leq a$ implies $b \otimes (a \oplusminus b) = a$ for all $a, b \in A^{\otimes}$.*

2.3. Cylindrification

We now consider two families of operators to model variables hiding and the passing of parameters in soft CCP. They can be considered as generalised notions of existential quantifier and diagonal element [1], which are expressed in terms of operators of cylindric algebras [16]. [3]

Definition 9 (Cylindrification). *Let V be a set of variables. A cylindric operator $\exists$ over $\mathbb{S}$ and V is a family of monotone, $\otimes$-compactness preserving functions $\exists_x : A \to A$ indexed by elements in V such that for all $a, b \in A$ and $x, y \in V$*

1. $\exists_x a \leq a$;
2. $\exists_x(a \otimes \exists_x b) = \exists_x a \otimes \exists_x b$;
3. $\exists_x \exists_y a = \exists_y \exists_x a$.

Let $a \in A$. The support *of a is the set of variables $sv(a) = \{x \in V \mid \exists_x a \neq a\}$.*

For a finite $X \subseteq V$, let $\exists_X a$ denote any sequence of function applications. We now fix a set of variables V and a cylindric operator $\exists$ over $\mathbb{S}$ and V.

Definition 10 (Diagonalisation). *A diagonal operator δ for $\exists$ is a family of idempotent elements $\delta_{x,y} \in A$ indexed by pairs of elements in V such that $\delta_{x,y} = \delta_{y,x}$ and for all $a \in A$ and $x, y, z \in V$*

1. $\delta_{x,x} = \bot$;

[3] However, since we consider monoids instead of groups, the set of axiom of diagonal operators is included in the standard one for cylindric algebras.

2. *if* $z \notin \{x, y\}$ *then* $\delta_{x,y} = \exists_z(\delta_{x,z} \otimes \delta_{z,y})$;

3. *if* $x \neq y$ *then* $a \leq \delta_{x,y} \otimes \exists_x(a \otimes \delta_{x,y})$.

Axioms 1 and 2 above plus idempotency imply that $\exists_x \delta_{x,y} = \bot$, which in turn implies (by axiom 2 and idempotency of $\exists$) that $sv(\delta_{x,y}) = \{x, y\}$ for $x \neq y$. Diagonal operators are used for modelling variable substitution and parameter passing. In the following, we fix a diagonal operator δ for $\exists$.

Definition 11 (Substitution). *Let* $x, y \in V$ *and* $a \in A$. *The substitution* $a[^y/_x]$ *is defined as* a *if* $x = y$ *and as* $\exists_x(\delta_{x,y} \otimes a)$ *otherwise.*

Substitution behaves correctly with respect to $\exists$.

Lemma 2. *Let* $x, y, w \in V$ *and* $a \in A$. *Then it holds*

- $(\exists_x a)[^y/_x] = a$
- $\exists_x a = \exists_y(a[^y/_x])$ *for* $y \notin sv(a)$
- $(\exists_w a)[^y/_x] = \exists_w(a[^y/_x])$ *if* $w \notin \{x, y\}$

Proof 2. *The proofs are immediate. Consider for instance the most difficult item* 3. *If* $x = y$ *the proof is over. Now, since* $w \notin \{x, y\}$ *we have by definition that* $\delta_{x,y} = \exists_w(\delta_{x,w} \otimes \delta_{w,y})$. *Again by definition* $b[^y/_x] = \exists_x(\delta_{x,y} \otimes b)$, *so that* $\exists_w(a[^y/_x]) = \exists_w \exists_x(\delta_{x,y} \otimes a) = \exists_x \exists_w(\delta_{x,y} \otimes a) = \exists_x(\delta_{x,y} \otimes \exists_w a) = (\exists_w a)[^y/_x]$.

Finally, we rephrase some further laws of the crisp case (see [2, p.140]).

Lemma 3. *Let* $x, y \in V$ *and* $a \in A$. *Then it holds*

1. $(a[^y/_x])[^x/_y] = a$ *for* $y \notin sv(a)$
2. $a[^y/_x] \otimes b[^y/_x] = (a \otimes b)[^y/_x]$
3. $x \notin sv(a[^y/_x])$.

Proof 3. *Consider the most difficult item* 2. *By definition* $a[^y/_x] \otimes b[^y/_x] = \exists_x(\delta_{x,y} \otimes a) \otimes \exists_x(\delta_{x,y} \otimes b)$, *which in turn coincides with* $\exists_x(\delta_{x,y} \otimes a \otimes \exists_x(\delta_{x,y} \otimes b))$ *by axiom* 2 *of* $\exists$; *by axiom* 3 *of* $\delta_{x,y}$ *we have that* $(a \otimes b)[^y/_x] = \exists_x(\delta_{x,y} \otimes a \otimes b) \leq \exists_x(\delta_{x,y} \otimes a \otimes \exists_x(\delta_{x,y} \otimes b))$, *while the vice versa holds by the monotonicity of* $\exists_x$.

2.4. A Case Study: Soft Constraints

In the past sections we mentioned the CLIM $\langle [0,1], \geq, \times \rangle$ of the $[0,1]$ interval of real numbers with the inverse order and multiplication as the monoidal operator. Another example is given by the CLIM $\langle R^+, \leq, + \rangle$ of non-negative reals plus ∞ with the standard order and addition: they are often called the fuzzy and the weighted/tropical semiring in the literature.

In this section we give a main example of CLIM where the notion of cylindrification can be easily given, by means of the notion of soft constraint: indeed, our proposal follows yet generalises [3].

Definition 12 (Soft constraints). *Let V be a set of variables, D a finite domain of interpretation and $\mathbb{S} = \langle A, \leq, \otimes \rangle$ a CLIM. Then, a* (soft) constraint $c : (V \to D) \to A$ *is a function associating a value in A with each assignment $\eta : V \to D$ of the variables.*

In the following, we define $\mathbb{C}$ as the set of constraints that can be built starting from chosen $\mathbb{S}$, V and D. The application of a constraint function $c : (V \to D) \to A$ to a variable assignment $\eta : V \to D$ is denoted $c\eta$.

Even if a constraint involves all the variables in V, it may depend on the assignment of a finite subset of them, called its support. For instance, a binary constraint c with $supp(c) = \{x, y\}$ is a function $c : (V \to D) \to A$ which depends only on the assignment of variables $\{x, y\} \subseteq V$, meaning that two assignments $\eta_1, \eta_2 : V \to D$ differing only for the image of variables $z \notin \{x, y\}$ coincide (i.e., $c\eta_1 = c\eta_2$). The support corresponds to the classical notion of scope of a constraint. We often refer to a constraint with support X as c_X. Moreover, an assignment over a support X of size k is concisely represented by a tuple t in D^k and we often write $c_X(t)$ instead of $c_X\eta$.

The set of constraints forms a CLIM, with the structure lifted from $\mathbb{S}$.

Lemma 4 (The CLIM of constraints). *The CLIM of constraints $\mathbb{C}$ is defined as $\langle C, \leq, \otimes \rangle$ such that*

- $c_1 \leq c_2$ *if* $c_1\eta \leq c_2\eta$ *for all* $\eta : V \to D$
- $(c_1 \otimes c_2)\eta = c_1\eta \otimes c_2\eta$ *for all* $c_1, c_2 \in C$

Combining two constraints by the $\otimes$ operator means building a new constraint whose support involves at most the variables of the original

ones. The obtained constraint associates with each tuple of domain values for such variables the element that is obtained by multiplying those associated by the original constraints to the appropriate sub-tuples. Residuation works as expected (i.e., $(c_1 \oplusminus c_2)\eta = c_1\eta \oplusminus c_2\eta$), and top and bottom are the constant functions mapping all η to $\top$ and $\bot$, respectively.

Example 3 (A simple CLIM). *Let us consider as* $\mathbb{S}$ *the CLIM of non-negative reals, and as D a finite subset of such reals. A polynomial with variables in V and non-negative reals as coefficients such as* $ux \hat{+} vy \hat{+} z$ *(where hat operators represent corresponding arithmetic operations) can be interpreted as the soft constraint associating with a function* $\eta : V \to D$ *the value* $u \hat{\times} \eta(x) \hat{+} v \hat{\times} \eta(y) \hat{+} z$*. Clearly, the composition of such constraints is precisely the addition of polynomials. Instead, the ordering might not be the one induced by the coefficients, due to the presence of constants. For example, assuming* $D = \{1, 2, 3\}$*, then* $2x \hat{+} 1 \leq x \hat{+} 9$*: hence,* $u_1 \leq u_2$ *and* $z_1 \leq z_2$ *implies* $u_1 x \hat{+} z_1 \leq u_2 x \hat{+} z_2$*, but the vice versa does not hold. Similarly for residuation, which is just bounded subtraction, as long as there is no constant. For example,* $(2x \hat{+} 3y) \oplusminus (x \hat{+} y) = x \hat{+} 2y$*, while* $(x \hat{+} y) \oplusminus (2x \hat{+} 3y)$ *is the bottom constraint mapping all assignments to* 0*. Instead,* $(2x \hat{+} 1) \oplusminus (x \hat{+} 9)$ *is not x, but again the bottom constraint, while* $(x \hat{+} 9) \oplusminus (2x \hat{+} 1)$ *could be described as* $\hat{-}x \hat{+} 8$*, even if the latter falls outside of the polynomials we considered since it has a negative coefficient. If D is not the singleton, the support of a polynomial is precisely the set of variables occurring in it.*

The CLIM of constraints also enjoys the cylindric properties, as shown by the result below (adapted from the idempotent case, due to [3]).

Lemma 5 (cylindric and diagonal operators for constraints). *The CLIM of constraints* $\mathbb{C}$ *admits cylindric and diagonal operators, defined as*

- $(\exists_x c)\eta = \bigwedge_{d \in D} c\eta[x := d]$ *for all* $c \in C, x \in V$
- $\delta_{x,y}\eta = \begin{cases} \bot & \text{if } \eta(x) = \eta(y); \\ \top & \text{otherwise.} \end{cases}$ *for all* $x, y \in V$

Hiding means eliminating variables from the support, and indeed, $supp(\exists_x c) \subseteq supp(c) \setminus \{x\}$.[4] Finally, the diagonal element $\delta_{x,y}$ has support $\{x, y\}$ for $x \neq y$, while the support for $\delta_{x,x}$ is $\emptyset$.

[4]The operator is called *projection* in the soft framework, and $\exists_x c$ is denoted $c \Downarrow_{V-\{x\}}$.

Example 4 (Continued...). *Let us consider again the situation of Example 3, and let* $D = \{1, 2, 3\}$. *Then,* $\exists_x(2x \hat{+} 1) = 3$, *the minimum obtained for the evaluation of the polynomial with respect to the elements in D. Instead,* $\delta_{x,y}$ *can be considered as a kind of matching* $[x = y]$, *since* $[x = y]\eta$ *is either* 0 *or* ∞ *depending if either* $\eta(x) = \eta(y)$ *or* $\eta(x) = \eta(y)$.

Note also that the diagonal elements are not guaranteed to be $\otimes$-compact, even if they have finite support, since $\top$ is not necessarily so. To this end, we close the section by adding the simple result below to the soft constraint lore.

Proposition 1. *Let* $c \in \mathbb{C}$ *be a constraint. It is* $\otimes$*-compact if and only if it has finite support and* $c\eta$ *is* $\otimes$*-compact for all* η.

3. Deterministic Soft CCP

This section introduces our (meta-)language. We fix a set of variables V, ranged over by $x, y, \ldots$, and an invertible CLIM $\mathbb{S} = \langle C, \leq, \otimes \rangle$, which is cylindric over V and whose compact elements are ranged over by $c, d, \ldots$.

Definition 13 (Agents). *The set* $\mathcal{A}$ *of all agents, parametric with respect to a set* $\mathcal{P}$ *of (unary) procedure declarations* $p(x)$, *is given by the following grammar*

$$A ::= \ \mathbf{stop} \mid \mathbf{tell}(c) \mid \mathbf{ask}(c) \rightarrow A \mid A \parallel A \mid \exists_x A \mid p(x).$$

Hence, the syntax includes a termination agent **stop**, and the two typical operations in *CCP* [1] languages: **tell**(c) adds the constraint c to a common store through which all agents interact, and **ask**$(c) \rightarrow A$ continues as agent A only when c is entailed by such a store (otherwise its execution is suspended). The other operators respectively express the parallel composition between two agents (i.e., $A \parallel A$), the hiding of a variable x in the computation of A ($\exists_x A$), and, finally, the calling of a procedure $p \in \mathcal{P}$ (whose body is an agent A) with an actual parameter identified by variable x.

We denote by $fv(A)$ the set of free variables of an agent, defined in the expected way by structural induction, assuming that $fv(\mathbf{tell}(c)) = sv(c)$ and $fv(\mathbf{ask}(c) \rightarrow A) = sv(c) \cup fv(A)$. In the following, we restrict our attention to procedure declarations $p(x) = A$ such that $fv(A) = \{x\}$ and there is no unbound replication, i.e., each occurrence in A of $p(y)$ is below the scope of either an **ask** or an hiding operator (that is, A is not of the shape $p(x) \parallel B$).

We now move to consider the reduction semantics of our language. The first step is to define substitution for agents, along the lines of the operator on CLIMs given in Definition 11.

Definition 14 (Substitutions). *Let x, y be variables in V. The substitution $[^y/_x] : \mathcal{A} \to \mathcal{A}$ is a function on agents defined by structural induction as*

- $\mathbf{stop}[^y/_x] = \mathbf{stop}$
- $(\mathbf{ask}(c) \to A)[^y/_x] = \mathbf{ask}(c[^y/_x]) \to A[^y/_x]$
- $\mathbf{tell}(c)[^y/_x] = \mathbf{tell}(c[^y/_x])$
- $(A_1 \parallel A_2)[^y/_x] = (A_1[^y/_x] \parallel A_2[^y/_x])$
- $p(w)[^y/_x] = \begin{cases} p(y) & \text{if } w = x \\ p(w) & \text{otherwise} \end{cases}$
- $(\exists_w A)[^y/_x] = \begin{cases} \exists_w A & \text{if } w = x \\ \exists_z (A[^z/_y][^y/_x]) & \text{for } z \text{ fresh if } w = y \\ \exists_w (A[^y/_x]) & \text{otherwise} \end{cases}$

As we recalled, the substitution on compact elements is the one given in Definition 11. Thus, thanks to Lemma 2 the choice of the intermediate variable is immaterial. In the following we consider agents to be equivalent up-to renaming of bound variables, that is $\exists_x A = \exists_y (A[^y/_x])$ for $y \notin sv(A)$.

Even if we do not state them explicitly, and considering the parallel operator for the tensor, also the laws reported in Lemma 3 holds for substitutions on agents. We are now able to propose our reduction semantics.

Definition 15 (Reductions). *Let $\Gamma = \mathcal{A} \times C^{\otimes}$ be the set of* configurations*. The* direct reduction semantics *for SCCP is the pair $\langle \Gamma, \mapsto \rangle$ such that $\mapsto \subseteq \Gamma \times \Gamma$ is the family of binary relations indexed over sets of variables, i.e., $\mapsto = \bigcup_{\Delta \subseteq V} \mapsto_\Delta$ and $\mapsto_\Delta \subseteq \Gamma \times \Gamma$, obtained by the rules in Table 1.*

The reduction semantics *for SCCP is the pair $\langle \Gamma, \to \rangle$ such that $\to \subseteq \Gamma \times \Gamma$ is the family of binary relations indexed over sets of variables, i.e., $\to = \bigcup_{\Delta \subseteq V} \longrightarrow_\Delta$ and $\longrightarrow_\Delta \subseteq \Gamma \times \Gamma$, obtained by the rules in Table 1 and Table 2.*

The split distinguishes between axioms and rules guaranteeing the closure with respect to the parallel operator. Indeed, rules **R1** and **R2** model the interleaving of two agents in parallel. We index transitions (i.e., $\mapsto_\Delta$ and $\longrightarrow_\Delta$) with a set of variables $\Delta \subseteq V$: Δ represents the bag of variables used across different parallel computation-branches, in order to

A1	$\dfrac{sv(\sigma) \cup sv(c) \subseteq \Delta}{\langle \mathbf{tell}(c), \sigma \rangle \mapsto_\Delta \langle \mathbf{stop}, \sigma \otimes c \rangle}$	**Tell**
A2	$\dfrac{sv(\sigma) \cup sv(c) \cup fv(A) \subseteq \Delta \wedge c \leq \sigma}{\langle \mathbf{ask}(c) \rightarrow A, \sigma \rangle \mapsto_\Delta \langle A, \sigma \rangle}$	**Ask**
A3	$\dfrac{sv(\sigma) \cup \{y\} \subseteq \Delta \wedge p(x) = A \in \mathcal{P}}{\langle p(y), \sigma \rangle \mapsto_\Delta \langle A[^y/_x], \sigma \rangle}$	**Rec**
A4	$\dfrac{sv(\sigma) \cup fv(\exists_x A) \subseteq \Delta \wedge w \notin \Delta}{\langle \exists_x A, \sigma \rangle \mapsto_\Delta \langle A[^w/_x], \sigma \rangle}$	**Hide**

Table 1: Axioms of the reduction semantics for SCCP.

R1	$\dfrac{\langle A, \sigma \rangle \longrightarrow_\Delta \langle A', \sigma' \rangle \wedge fv(B) \subseteq \Delta}{\langle A \parallel B, \sigma \rangle \longrightarrow_\Delta \langle A' \parallel B, \sigma' \rangle}$	**Par1**
R2	$\dfrac{\langle A, \sigma \rangle \longrightarrow_\Delta \langle A', \sigma' \rangle \wedge fv(B) \subseteq \Delta}{\langle B \parallel A, \sigma \rangle \longrightarrow_\Delta \langle B \parallel A', \sigma' \rangle}$	**Par2**

Table 2: Contextual rules of the reduction semantics for SCCP.

provide globally new names when a fresh variable is needed, e.g., in **A4**. The set of variables Δ has to be global because renaming x with w in one branch could reuse the same w fresh in a different branch.

In **A1** a constraint c is added to the store σ. **A2** checks if c is entailed by σ: if not, the computation is blocked. Axiom **A3** replaces a procedure identifier with the associated body, renaming the formal parameter with the actual one. Axiom **A4** hides the variable x occurring in A, replacing it with a globally fresh variable, as ensured by $w \notin \Delta$. Note that the latter is more general than just requiring that $w \notin fv(\exists_x A) \cup sv(\sigma)$, since $\langle B, \rho\rangle \longrightarrow_\Delta$ implies that $fv(B) \cup sv(\rho) \subseteq \Delta$.[5]

Example 5 (Ask, tell and parallel). *We provide some examples of agent reductions to clarify the semantics in Table 1. Note that we drop the trailing* **stop** *whenever in parallel with another agent. Moreover, we consider two generic constraints c_1 (with support $sv(c_1) = \{x\}$) and c_2 (with support $sv(c_2) = \{y\}$).*

We use the querying agent $\mathbf{ask}(c_1)$*, executed in a store $c_1 \otimes c_2$: since $c_1 \leq c_1 \otimes c_2$, $sv(c_1 \otimes c_2) \subseteq \{x, y\}$, and $sv(c_1) = \{x\}$, using rule* **A1** *in Table 1 we get* $\langle \mathbf{ask}(c_1) \to \mathbf{stop}, c_1 \otimes c_2\rangle \mapsto_{\{x,y\}} \langle \mathbf{stop}, c_1 \otimes c_2\rangle$.

Now consider the agent $\mathbf{tell}(c_1) \parallel \mathbf{tell}(c_2)$*. As stated in the precondition of rules* **R1**-**R2** *in Table 1, the free variables of the suspended branch have to be in the subscript Δ as well, since Δ collects all the variables that are going to be used in all the branches. As an example, since $fv(\mathbf{tell}(c_2)) = sv(c_2) = y$, then* $\langle \mathbf{tell}(c_1) \parallel \mathbf{tell}(c_2), \bot\rangle \mapsto_{\{x,y\}} \langle \mathbf{tell}(c_2), c_1\rangle \mapsto_{\{x,y\}} \langle \mathbf{stop}, c_1 \otimes c_2\rangle$.

Example 6 (Procedure call and hiding). *In this example we adopt the CLIM $\langle \mathbb{R}^+, \leq, +\rangle$ of positive real numbers. In this setting, we use two soft constraints (see Section 2.4) $c_1 = x \hat{+} 1$ ($sv(c_1) = \{x\}$) and $c_2 = y \hat{+} 2$ ($sv(c_2) = \{y\}$).*

Rule **A3** *in Table 1 just requires the actual parameter y to be in the subscript Δ (together with the support variables of the global store, in this case $sv(c_1) = \{x\}$); then, it applies a straight renaming of the formal parameter with the actual one. An example is given below, where we suppose to have $p(x) = \exists_x \mathbf{tell}(c_1)$ among the procedures declared in $\mathcal{P}$, so that $\langle p(y), c_1\rangle \mapsto_{\{x,y\}} \langle \exists_y \mathbf{tell}(c_1[{}^y/_x]), c_1\rangle \ldots$*

Note that, by substituting x with y in c_1 (i.e., $c_1[{}^y/_x]$), the value becomes $y \hat{+} 1$ (see Definition 11, and Lemma 4 for the operator instantiations): in words, the value of x in c_1 is passed to y with a diagonal constraint ($\delta_{x,y} \otimes c_1$), and then the influence of x is removed from c_1 through the cylindric operator: $\exists_x(\delta_{x,y} \otimes c_1)$. The

[5] Our rule is reminiscent of (8) in [1, p. 342].

next step concerns the hiding operator: rule **A4** *in Table 1 requires as precondition a fresh name w ($w \notin \{x, y\}$) that is used to rename the hidden variable y. In this case, y is renamed with w in c_1 (thus, $d = w \hat{+} 1$, see Proposition 4), and w is then added to the subscript Δ because of the tell action $\ldots \langle \exists_y \mathbf{tell}(c_1[{}^{y}/_{x}]), c_1\rangle \mapsto_{\{x,y\}} \langle \mathbf{tell}((c_1[{}^{y}/_{x}])[{}^{w}/_{y}]), c_1\rangle \mapsto_{\{x,y,w\}} \langle \mathbf{stop}, c_1 \otimes d\rangle$.*

Therefore, the final store is $c_1 \otimes d$. Notice that y has been recorded as used, but it is no longer present in the global store.[6]

Our next set is to prove some monotonicity properties for the reduction semantics, which will play a role in defining the notion of computation. So, let $\gamma = \langle A, \sigma\rangle$ be a configuration. We denote by $fv(\gamma)$ the set $fv(A) \cup sv(\sigma)$ and by $\gamma[{}^{z}/_{w}]$ the component-wise application of substitution $[{}^{z}/_{w}]$.

Lemma 6 (On monotonicity). *Let $\langle A, \sigma\rangle \rightarrow_\Delta \langle B, \sigma'\rangle$ be a reduction. Then*

1. *$sv(\sigma') \subseteq sv(\sigma) \cup fv(A) \subseteq \Delta$;*
2. *$\sigma \leq \sigma'$;*
3. *$\langle A, \sigma\rangle \rightarrow_{\Delta'} \langle B, \sigma'\rangle$ if $sv(\sigma) \cup fv(A) \subseteq \Delta' \wedge fv(B) \cap \Delta' \subseteq fv(A)$;*
4. *$\langle A, \sigma \otimes \rho\rangle \rightarrow_\Delta \langle B, \sigma' \otimes \rho\rangle$ if $\rho \in C^{\otimes} \wedge sv(\rho) \subseteq \Delta$.*

Proof 4. *All statements are straightforward. Consider item 1. By construction $sv(\sigma) \cup fv(A) \subseteq \Delta$: since only the rule* **A1** *can modify the store and $sv(\sigma \otimes c) \subseteq sv(\sigma) \cup sv(c)$, then the statement holds. Similarly for item 2, since $\sigma \leq \sigma \otimes c$. Also item 3 is true by construction: since the set of variables in the rule* **A4** *can always be enlarged. As for item 4, it suffices to also note that $\sigma, \rho \in C^{\otimes}$ ensure that $\sigma \otimes \rho \in C^{\otimes}$ and clearly the rule* **A2** *will still be executable.*

Computations are just sequences of reductions, ranged over by ξ. Clearly, also infinite computations are well defined, since adding (infinitely many) compact elements to a store that is already compact will always end up in a compact store. However, in order to proper account for the set of global variables, we restrict our attention of computations such that the set of variables associated to derivations is increasing with the time.

[6] Note $sv(c \otimes d) \subseteq sv(c) \cup sv(d)$, differently from what stated in [3], where = is assumed.

Definition 16 (Increasing computations). *Let* $\gamma_0 \rightarrow_{\Delta_1} \gamma_1 \rightarrow_{\Delta_2} \gamma_2 \rightarrow_{\Delta_3} \dots$ *be a (possibly infinite) computation. It is* increasing *if* $\Delta_k \subseteq \Delta_{k+1}$ *for any* $k > 1$*, and* minimally increasing *if* $\Delta_k + fv(\gamma_k) = \Delta_{k+1}$ *for any* $k > 1$*.*

What is noteworthy is that in a minimally increasing computation the sets of variables Δ's are always uniquely identified, once Δ_1 is fixed. In fact, we could be more restrictive by imposing $fv(\gamma_0) = \Delta_1$, yet this would make some of the statements unnecessarily cumbersome. In any case, thanks to Lemma 6, for the sake of simplicity and without loss of generality in the following we restrict our attention to minimally increasing computations, dropping altogether the subscripts Δ's whenever they are irrelevant.

3.1. Observables and local confluence

The direct reduction semantics is at its heart deterministic, even of there can be some issues concerning the choice of the fresh variable introduced.

Lemma 7. *Let* $\gamma \mapsto_\Delta \gamma_i$ *be direct reductions for* $i = 1, 2$*. Then either* $\gamma_1 = \gamma_2$ *or*

- $\gamma \mapsto_\Delta \gamma'$ *with* $\gamma' = \gamma_1[^z/_{w_1}] = \gamma_2[^z/_{w_2}]$ *for* $w_i \in fv(\gamma_i) \setminus \Delta$ *and* z *fresh.*

Proof 5. *The direct reduction semantics is deterministic, and the only freedom is the choice of the fresh variable in rule* **A4***. Let us assume that different variables are chosen, let us say* w_1 *and* w_2*. However,* $\gamma_1[^z/_{w_1}] = \gamma_2[^z/_{w_2}]$ *by replacing the new variables with a globally fresh one, and the existence of* ξ *is immediate.*

Proposition 2. *Let* $\xi_i = \gamma \rightarrow_\Delta \gamma_i$ *be reductions for* $i = 1, 2$*. Then either* $\gamma_1 = \gamma_2$ *or*

1. $\xi' = \gamma \rightarrow_\Delta \gamma'$ *with* $\gamma' = \gamma_1[^z/_{w_1}] = \gamma_2[^z/_{w_2}]$ *for* $w_i \in fv(\gamma_i) \setminus \Delta$ *and* z *fresh;*
2. $\xi'_i = \gamma \rightarrow_\Delta \gamma_i \rightarrow_{\Delta_i} \gamma_3$*;*
3. $\xi'_i = \gamma \rightarrow_\Delta \gamma_i[^{z_i}/_w] \rightarrow_{\Delta \cup \{z_i\}} \gamma_3$ *for* $w \in fv(\gamma_i) \setminus (\Delta \cup fv(\gamma_3))$ *and* z_i*'s fresh.*

Proof 6. *The first item is clearly due to the choice of different free variables for the same hiding operator, as for direct reduction semantics.*

So, let us assume that the two reductions occur on the opposite sides of a parallel operator. A first relevant case is if both reductions replace an hiding operator with the same fresh variable w*. However, it suffices to replace* w *with fresh variables* z_1 *and* z_2 *in the two reductions, in order for item 3 to be verified.*

Among the remaining cases, the only relevant one is if both actions add different constraints to the store. So, let us assume that $\gamma = \langle A_1 \parallel A_2, \sigma\rangle$ *such that* $\langle A_1, \sigma\rangle \to_\Delta \langle B_1, \sigma_1\rangle$ *and* $\langle A_2, \sigma\rangle \to_\Delta \langle B_2, \sigma_2\rangle$*. Note that since reduction semantics is monotone (Lemma 6) and* σ *is* $\otimes$*-compact, also* σ_1 *is* $\otimes$*-compact and furthermore we have* $\sigma_1 = \sigma \otimes (\sigma_1 \ominus \sigma)$*. Now, monotonicity (again Lemma 6) ensures us that* $\langle B_1 \parallel A_2, \sigma \otimes (\sigma_1 \ominus \sigma)\rangle \to_\Delta \langle B_1 \parallel B_2, \sigma \otimes (\sigma_1 \ominus \sigma) \otimes (\sigma_2 \ominus \sigma)\rangle$ *and by symmetric reasoning the latter configuration is the one we were looking for.*

The result above is a local confluence theorem, which is expected, since the calculus is essentially deterministic. The complex formulation is due to the occurrence of hiding operators: as an example, different fresh variables may be chosen for replacing $\exists_x$, such as w_1 and w_2 in the first item above, and then a globally fresh variable z has to be found for replacing them.

3.2. Observables

We now need to introduce a suitable notion of observable for a computation, which will be pivotal in defining agent equivalence.

Definition 17 (Observables). *Let* $\xi = \langle A_0, \sigma_0\rangle \to \langle A_1, \sigma_1\rangle \to \dots$ *be a (possibly infinite) computation. Then its observation* $Result(\xi)$ *is* $\bigvee_i(\exists_{X_i}\sigma_i)$*, for* $X_i = fv(\langle A_i, \sigma_i\rangle) \setminus fv(\langle A_0, \sigma_0\rangle)$*.*

For finite computations the result is obviously the store of the final configuration. Furthermore, we can strengthen the local confluence results of the previous section by proving that the observable are preserved.

Lemma 8. *Let* $\xi_i = \gamma \to_\Delta \gamma_i$ *be reductions for* $i = 1, 2$ *and* ξ'*,* ξ'_i *computations as obtained in Proposition 2. Then either* $Result(\xi') = Result(\xi_1) = Result(\xi_2)$ *(item* 1*) or* $Result(\xi'_1) = Result(\xi'_2)$ *(item* 2 *and item* 3*).*

We close with a technical lemma concerning the substitution of fresh variables in a computation, which ensures that it is essentially captured by a substitution in the initial configuration: in other terms, observables are also preserved under renaming with fresh variables.

Lemma 9. *Let* $\xi = \gamma_0 \to \gamma_1 \to \dots$ *be a (possibly infinite) computation and* w*,* z *variables such that* $w \in fv(\gamma_0)$ *and* z *is fresh. Then there exists a computation* $\xi[^z/_w] = \gamma_0[^z/_w] \to \gamma_1[^z/_w] \to \dots$ *such that* $Result(\xi)[^z/_w] = Result(\xi[^z/_w])$*.*

3.3. Fairness and observational equivalence

Intuitively, a computation is fair if each sub-agents that might be executed at a certain point will surely be executed sooner or later. However, in order to properly characterise possibly different occurrences of the same sub-agent, we need to introduce a syntactical device.

Definition 18 (Paths). *Let $A \in \mathcal{A}$ be an agent. The set of paths of A is the set of strings $P(A) \subseteq \{0,1\}^*$ defined by structural induction as:*

- $P(\mathbf{stop}) = P(\mathbf{tell}(c)) = P(p(x)) = \{\epsilon\}$.
- $P(\mathbf{ask}(c) \to A) = P(\exists_x A) = \{\epsilon\} \cup \{0\} \cdot P(A)$.
- $P(A_1 \parallel A_2) = \{\epsilon\} \cup \{0\} \cdot P(A_1) \cup \{1\} \cdot P(A_2)$.

Thus, given an agent A, its set of paths can be used to characterise the occurrences of sub-agents. More precisely, for each path $\pi \in P(A)$, the expression $A|_\pi$ denotes uniquely a sub-agent of A. For example, consider the agent $A = \mathbf{tell}(c) \parallel \mathbf{ask}(d) \to \mathbf{tell}(c)$. Its set of paths $P(A) = \{\epsilon, 0, 1, 10\}$, and each of them identifies a sub-agent, namely $A|_\epsilon = A, A|_0 = A|_{10} = \mathbf{tell}(c)$, and $A|_1 = \mathbf{ask}(d) \to \mathbf{tell}(c)$.

Now, for a configuration/reduction we can introduce enabled and active paths, intuitively witnessing when the corresponding sub-agents might actually be, and indeed are, executed.

Definition 19 (Enabled/Active paths). *Let $\gamma = \langle A, \sigma\rangle$ be a configuration and π a path in A. We say that π is enabled in γ if $\langle A|_\pi, \sigma\rangle \mapsto$ and for all prefixes π' of π the sub-agent $A|_{\pi'}$ has the shape $A_1 \parallel A_2$.*

Let $\xi = \langle A, \sigma\rangle \to \langle B, \rho\rangle$ be a reduction. We say that π is active in ξ if it is enabled in $\langle A, \sigma\rangle$ and $\langle A|_\pi, \sigma\rangle \mapsto \langle B|_\pi, \rho\rangle$.

The intuition is that a path is enabled if the corresponding sub-agent may perform a reduction $\mapsto$, hence such sub-agent is of the shape $\mathbf{tell}(c)$ or $\mathbf{ask}(c) \to A$ or $p(x)$ or $\exists_x A$ and the reduction is obtained by the application of exactly one instance of one of the axioms in Table 1. And an agent of such shape is *active* in a reduction $\gamma \to \gamma'$ if it is enabled in the configuration γ, and in fact generates that transition.

Looking at the previous example with agent $A = \mathbf{tell}(c) \parallel \mathbf{ask}(d) \to \mathbf{tell}(c)$, let us consider the configuration $\gamma = \langle A, \bot\rangle$. Now, path 0 is enabled

in γ, and in fact it is active in the derivation $\langle A, \bot\rangle \to \langle \mathbf{stop} \parallel \mathbf{ask}(d) \to \mathbf{tell}(c), c\rangle$. Instead, 1 is not enabled in γ, and indeed it is not active in any reduction starting from it. It will become enabled only if $d \leq c$.

We want to exploit the notion of paths for finding a good definition of fair computation, which is somehow elusive. Consider how fairness for the crisp case is stated in [1, p.27]: it basically says that whenever a sub-agent is enabled at a certain point, it is going to be eventually executed. However, in this context *what* is a sub-agent may be ambiguous. Consider the agent $A = p(x) \parallel \mathbf{tell}(c)$ for $p(x) = (\mathbf{ask}(\bot) \to p(x)) \parallel \mathbf{tell}(c)$. The sub-agent $\mathbf{tell}(c)$ in A might never be executed, yet a different occurrence of the same sub-agent may be executed, and infinitely often at that.

We believe that such a computation should not be fair. This requires some care in identifying the transformation of the path of a sub-agent along a computation. However, we are assisted by the following result.

Lemma 10. *Let $\xi = \langle A, \sigma\rangle \to \langle B, \rho\rangle$ be a reduction and π a path is A. If π is enabled in $\langle A, \sigma\rangle$ and not active in ξ, then it is enabled in $\langle B, \rho\rangle$ and $A \mid_\pi = B \mid_\pi$.*

The proof is immediate. This result guarantees that whenever a path is enabled, it keeps on being so until the corresponding sub-agent is executed.

Definition 20 (Fair computations). *Let $\gamma_0 \to \gamma_1 \to \ldots$ be a (possibly infinite) computation. We say that it is* fair *if whenever a path π is enabled in some γ_i then π is active in $\gamma_j \to \gamma_{j+1}$ for some $j \geq i$.*

The definition above seems to precisely capture the notion of fairness for crisp programming [1, p.27]. As for the crisp case, if a finite computation is fair then it is deadlocked and its result is the store of the final configuration.

Theorem 1 (Confluence). *Let γ be a configuration and $\xi_1, \xi_2 : \gamma \to \ldots$ two (possibly infinite) fair computations. Then $Result(\xi_1) = Result(\xi_2)$.*

Proof 7. *The result is a combination of Proposition 2 and Lemmata 9 and 10. So, let us assume we have two (possibly infinite) fair computations originating from γ, and let us consider their initial reductions $\gamma \to_\Delta \gamma_i$ for $i = 1, 2$. First thing, note that assuming the same subscript Δ is not restrictive since we work with minimally increasing reductions. Now, according to Proposition 2, we have two cases, as depicted on Figure 1 and Figure 2.*

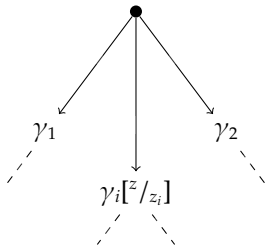

Figure 1: Proposition 2 (item 1) and Lemma 6 ensure that we may collapse the initial reductions (thus confluence).

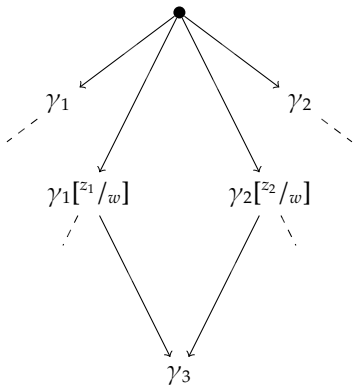

Figure 2: Proposition 2 (item 2), Lemma 6, and the restriction to fair computations ensure confluence.

Thus, fair computations originating from a configuration are either all finite or all infinite, and furthermore they have the same result.

Now, let us denote by $Result(\gamma)$ the result of a configuration γ, which is obtained as the result of any fair computation originating from it. We are now ready to propose our first semantics.

Definition 21 (Observational equivalence). *Let $A, B \in \mathcal{A}$ be agents. We say that they are observationally equivalent and we write $A \sim_o B$ if $Result(\langle A, \sigma\rangle) = Result(\langle B, \sigma\rangle)$ for all $\sigma \in C^{\otimes}$.*

It is easy to show $\sim_o$ to be preserved by all contexts, i.e., it is a *congruence*.

3.4. An example of infinite (and fair) computation

Fairness represents a class of infinitary properties, usually viewed as a sub-class of liveness ones (i.e., *"something good happens"*). Such properties restrict the set of potential computations by disallowing those infinite computations that indefinitely delay some system component [17]. Fairness has been often addressed in the literature on crisp CCP, see already [1, 17], even if a precise definition of such property is less common, see e.g., [18]. What is important is that our proposal is suitable also for the soft case.

In order to provide a larger example of an infinite and fair computation, we slightly extend our syntax by allowing procedures with multiple parameters and consider an interaction $A \parallel B$ between a sequencer agent

B that marks the rhythm of agent A, which in turn has the task to progressively consume an infinite resource by adding c_r each time. Constraints are indexed with their support variable, i.e., c_x, c_y, and c_r. A waits for B to grant permission to add c_r via a signal on c_x: when c_x is in the store, A adds c_r. When A finishes, it informs B by telling c_y, and B, hanging on the corresponding $\mathbf{ask}(c_y)$, is then unblocked. Finally, B launches the same parallel computation after changing the signalling variables x and y by renaming them (in the same way for both A and B). Two procedures $p_1(x, y, r)$ and $p_2(x, y)$ replicate the behaviour of the initial agents, so that they can be invoked infinite times.

$$A = p_1(x, y, r) : \mathbf{ask}(c_x) \rightarrow \mathbf{tell}(c_y \otimes c_r)$$

$$B = p_2(x, y) : \mathbf{tell}(c_x) \parallel (\mathbf{ask}(c_y) \rightarrow \exists_{\{x,y\}}(p_1(x, y) \parallel p_2(x, y)))$$

Looking now at the reduction semantics provided in Table 1, the first observables are described from Equation 1 to Equation 7.

$$\langle A \parallel B, \bot\rangle \longrightarrow_{\{r,x,y\}} \langle A \parallel \mathbf{ask}(c_y) \rightarrow \exists_{\{r,x,y\}}(p_1(x, y) \parallel p_2(x, y)), c_x\rangle \tag{1}$$

$$\begin{aligned}\langle \mathbf{tell}(c_y \otimes c_r) \parallel \mathbf{ask}(c_y) \rightarrow \exists_{\{r,x,y\}}(p_1(x, y) \parallel p_2(x, y)), c_x\rangle \longrightarrow_{\{r,x,y\}} \\ \langle \mathbf{tell}(c_y \otimes c_r) \parallel \mathbf{ask}(c_y) \rightarrow \exists_{\{r,x,y\}}(p_1(x, y) \parallel p_2(x, y)), c_x\rangle\end{aligned} \tag{2}$$

$$\begin{aligned}\langle \mathbf{tell}(c_y \otimes c_r) \parallel \mathbf{ask}(c_y) \rightarrow \exists_{\{r,x,y\}}(p_1(x, y) \parallel p_2(x, y)), c_x\rangle \longrightarrow_{\{r,x,y\}} \\ \langle \mathbf{ask}(c_y) \rightarrow \exists_{\{r,x,y\}}(p_1(x, y) \parallel p_2(x, y)), c_x \otimes c_y \otimes c_r\rangle\end{aligned} \tag{3}$$

$$\begin{aligned}\langle \mathbf{ask}(c_y) \rightarrow \exists_{\{r,x,y\}}(p_1(x, y) \parallel p_2(x, y)), c_x\rangle \longrightarrow_{\{r,x,y\}} \\ \langle \exists_{\{r,x,y\}}(p_1(x, y) \parallel p_2(x, y)), c_x \otimes c_y \otimes c_r\rangle\end{aligned} \tag{4}$$

Then, if we rename x with v and y with w we have

$$\begin{aligned}\langle \exists_{\{r,x,y\}}(p_1(x, y) \parallel p_2(x, y)), c_x \otimes c_r \otimes c_y \otimes c_r\rangle \longrightarrow_{\{r,x,y\}} \\ \langle p_1(x, y)[^v/_x][^w/_y] \parallel p_2(x, y)[^v/_x][^w/_y], c_x \otimes c_y \otimes c_r\rangle\end{aligned} \tag{5}$$

where $p_1(x, y)[^v/_x][^w/_y] = p_1(v, w)$ and $p_2(x, y)[^v/_x][^w/_y] = p_2(v, w)$. Then, if $p_1(v, w)$ is invoked first

$$\begin{aligned}\langle p_1(v, w) \parallel p_2(v, w), c_x \otimes c_y \otimes c_r\rangle \longrightarrow_{\{r,v,w,x,y\}} \\ \langle \mathbf{ask}(c_v) \rightarrow \mathbf{tell}(c_w \otimes c_r) \parallel p_2(v, w), c_x \otimes c_y \otimes c_r\rangle\end{aligned} \tag{6}$$

where $c_v = c_x[^v/_x]$ and $c_w = c_y \otimes c_r[^w/_y]$. finally, also $p_2(v, w)$ is invoked

$$\begin{aligned} \langle \mathbf{ask}(c_v) \to \mathbf{tell}(c_w \otimes c_r) \parallel p_2(v,w), c_x \otimes c_y \otimes c_r \rangle \longrightarrow_{\{r,v,w,x,y\}} \\ \langle \mathbf{ask}(c_v) \to \mathbf{tell}(c_w \otimes c_r) \parallel \mathbf{tell}(c_v) \parallel \\ \mathbf{ask}(c_w) \to \exists_{\{v,w\}}(p_1(v,w) \parallel p_2(v,w)), c_x \otimes c_y \otimes c_r \rangle \end{aligned} \tag{7}$$

Then the computation continues as in Equation 1, and so on.

4. Saturated Bisimulation

As proposed in [2] for crisp languages, we define a barbed equivalence between two agents [19]. Intuitively, barbs are basic observations (predicates) on the states of a system, and in our case they correspond to the compact constraints in $C^{\otimes}$.

Definition 22 (Barbs). *Let $\langle A, \sigma \rangle$ be a configuration and $c \in C^{\otimes}$ and we say that $\langle A, \sigma \rangle$ verifies c, or that $\langle A, \sigma \rangle \downarrow_c$ holds, if $c \leq \sigma$.*

However, since *barbed bisimilarity* is only an equivalence, along [2] we propose the use of *saturated bisimilarity* in order to obtain a congruence: Definition 23 and Definition 25 respectively provide the strong and weak definition of saturated bisimilarity.

For the sake of readability, in order to reduce the amount of symbols, for a configuration $\gamma = \langle A, \sigma \rangle$ and an element d we write $\gamma \otimes d$ for $\langle A, \sigma \otimes d \rangle$.

Definition 23 (Saturated bisimilarity). *A saturated bisimulation is a symmetric relation R on configurations such that whenever $(\gamma_1, \gamma_2) \in R$*

1. *if $\gamma_1 \downarrow_c$ then $\gamma_2 \downarrow_c$ for all $c \in C^{\otimes}$;*
2. *if $\gamma_1 \longrightarrow \gamma_1'$ then there exists γ_2' such that $\gamma_2 \longrightarrow \gamma_2'$ and $(\gamma_1', \gamma_2') \in R$;*
3. *$(\gamma_1 \otimes d, \gamma_2 \otimes d) \in R$ for all $d \in C^{\otimes}$.*

We say that γ_1 and γ_2 are saturated bisimilar ($\gamma_1 \sim_s \gamma_2$) if there exists a saturated bisimulation R such that $(\gamma_1, \gamma_2) \in R$. We write $A \sim_s B$ if $\langle A, \bot \rangle \sim_s \langle B, \bot \rangle$.

We now let $\longrightarrow^*$ denote the reflexive and transitive closure of $\longrightarrow$, restricted to increasing computations.

Definition 24 (Weak barbs). *Let $\gamma = \langle A, \sigma \rangle$ be a configuration and $c \in C^{\otimes}$. We say that $\langle A, \sigma \rangle$ weakly verifies c, or that $\langle A, \sigma \rangle \Downarrow_c$ holds, if there exists $\gamma' = \langle B, \rho \rangle$ such that $\gamma \longrightarrow^* \gamma'$ and $c \leq \exists_X \rho$ for $X = fv(\gamma') \setminus fv(\gamma)$.*

Definition 25 (Weak saturated bisimilarity). *A weak saturated bisimulation is a symmetric relation R on configurations such that whenever $(\gamma_1, \gamma_2) \in R$*

1. *if $\gamma_1 \downarrow_c$ then $\gamma_2 \Downarrow_c$ for all $c \in C^{\otimes}$;*
2. *if $\gamma_1 \longrightarrow \gamma_1'$ then there exists γ_2' such that $\gamma_2 \longrightarrow^* \gamma_2'$ and $(\gamma_1', \gamma_2') \in R$;*
3. *$(\gamma_1 \otimes d, \gamma_1 \otimes d) \in R$ for all $d \in C^{\otimes}$.*

We say that γ_1 and γ_2 are weakly saturated bisimilar ($\gamma_1 \approx_s \gamma_2$) if there exists a weak saturated bisimulation R such that $(\gamma_1, \gamma_2) \in R$. We write $A \approx_s B$ if $\langle A, \bot \rangle \approx_s \langle B, \bot \rangle$.

The asymmetry is functional to later sections. However, it is clearly equivalent to the standard symmetric version.

Definition 26 (Weak saturated bisimilarity, 2). *Weak saturated bisimilarity coincides with the relation obtained from Definition 23 by replacing $\longrightarrow$ with $\longrightarrow^*$ and $\downarrow_c$ with $\Downarrow_c$.*

Since $\sim_s$ (and $\approx_s$) is a saturated bisimulation, it is clearly upward closed and it is also a congruence: indeed, a context can modify the behaviour of a configuration only by adding constraints to its store.

We now show that $\approx_s$, as given in Definition 25, coincides with the observational equivalence $\sim_o$ (see Definition 21). First we recall the notion of and a classic result on *cofinality*: two (possibly infinite) chains $c_0 \leq c_1 \leq \ldots$ and $d_0 \leq d_1 \leq \ldots$ are said to be *cofinal* if for all c_i there exists a d_j such that $c_i \leq d_j$ and, vice-versa, for all d_i there exists a c_j such that $d_i \leq c_j$.

Lemma 11. *Let $c_0 \leq c_1 \leq \ldots$ and $d_0 \leq d_1 \leq \ldots$ be two chains.* (1) *If they are cofinal, then they have the same limit, i.e., $\bigvee_i c_i = \bigvee_i d_i$.* (2) *If the elements of the chains are $\otimes$-compact and $\bigvee_i c_i = \bigvee_i d_i$, then the two chains are cofinal.*

Proof 8. *Let us tackle* (2)*, and consider the sequence $e_0 = c_0$ and $e_i = c_{i+1} \ominus c_i$. Each e_i is the difference between two consecutive elements of a chain. Since the CLIM is invertible we have $c_k = \bigotimes_{i \leq k} e_i$ and thus $\bigvee_i c_i = \bigotimes_i e_i$. Since each d_j is $\otimes$-compact and $d_j \leq \bigotimes_i e_i$, there is a k such that $d_j \leq \bigotimes_{i \leq k} e_i$. The same reasoning is applied to the chain $d_0 \leq d_1 \leq \ldots$, thus the result holds.*

To prove Theorem 2, besides cofinality from Lemma 11 we need to relate weak barbs and fair computations.

Lemma 12. *Let $\xi = \gamma_0 \longrightarrow \gamma_1 \longrightarrow \ldots$ be a (possibly infinite) fair computation. If $\gamma_0 \Downarrow_d$ then there exists a store σ_i in ξ such that $d \leq \exists_{X_i}\sigma_i$ for $X_i = fv(\gamma_i) \setminus fv(\gamma_0)$.*

The lemma is a direct consequence of confluence (see Theorem 1).

Theorem 2. *Let A, B be agents. Then $A \sim_o B$ if and only if $A \approx_s B$.*

Proof 9. *The proof proceeds as follows.*

From $\approx_s$ to $\sim_o$. *Assume $\langle A, \bot\rangle \approx_s \langle B, \bot\rangle$ and take a $\otimes$-compact $c \in C^{\otimes}$. Let*

$$\langle A, c\rangle \longrightarrow \langle A_0, \sigma_0\rangle \longrightarrow \langle A_1, \sigma_1\rangle \longrightarrow \ldots \longrightarrow \langle A_n, \sigma_n\rangle \ldots \longrightarrow \ldots \quad (8)$$

$$\langle B, c\rangle \longrightarrow \langle B_0, \rho_0\rangle \longrightarrow \langle B_1, \rho_1\rangle \longrightarrow \ldots \longrightarrow \langle B_n, \rho_n\rangle \ldots \longrightarrow \ldots \quad (9)$$

be two fair computations. Since $\approx_s$ is upward closed, $\langle A, c\rangle \approx_s \langle B, c\rangle$ and thus $\langle B, c\rangle \Downarrow_{\sigma_i}$ for all σ_i. By Lemma 12, it follows that there exists an ρ_j (in the above computation) such that $\exists_{\Gamma_i}\sigma_i \leq \sigma_i \leq \exists_{\Gamma'_j}\rho_j$, and analogously for all ρ_i. Then $\sigma_0 \leq \sigma_1 \leq \ldots$ and $\rho_0 \leq \rho_1 \leq \ldots$ are cofinal and by Lemma 11 it holds that $\bigvee_i \exists_{\Gamma_i}\sigma_i = \bigvee_i \exists_{\Gamma'_i}\rho_i$, which means $Result(\langle A, c\rangle) = Result(\langle B, c\rangle)$.

From $\sim_o$ to $\approx_s$. *Assume $A \sim_o B$. First, we show that $\langle A, c\rangle$ and $\langle B, c\rangle$ satisfy the same weak barbs for all $c \in C$. Let (8) and (9) be two fair computations. Since $A \sim_o B$, then $\bigvee_i \exists_{\Gamma_i}\sigma_i = \bigvee_i \exists_{\Gamma'_j}\rho_i$. Since all (the projections of) the intermediate stores of the computations are $\otimes$-compact, then by Lemma 11 for all σ_i there exists an ρ_j such that $\exists_{\Gamma_i}\sigma_i \leq \exists_{\Gamma'_j}\rho_j$. Now suppose that $\langle A, c\rangle \Downarrow_d$. By Lemma 12, there exists a σ_i such that $d \leq \exists_{\Gamma_i}\sigma_i$. Thus $\langle B, c\rangle \Downarrow_d$.*

It is now easy to prove that $R = \{(\gamma_1, \gamma_2) \mid \exists c.\langle A, c\rangle \longrightarrow^* \gamma_1 \& \langle B, c\rangle \longrightarrow^* \gamma_2\}$ is a weak saturated bisimulation (Definition 25). Take $(\gamma_1, \gamma_2) \in R$. If $\gamma_1 \Downarrow_d$ then $\langle A, c\rangle \Downarrow_d$ and, by the above observation, $\langle B, c\rangle \Downarrow_d$. Since SCCP is confluent, also $\gamma_2 \Downarrow_d$. The fact that R is closed under $\longrightarrow^*$ is evident from the definition of R. While for proving that R is upward-closed take $\gamma_1 = \langle A', \sigma'\rangle$ and $\gamma_2 = \langle B', \rho'\rangle$. By item 4 of Lemma 6 (i.e., language monotonicity) for all $a \in C$, $\langle A, c \otimes a\rangle \longrightarrow^* \langle A', \sigma' \otimes a\rangle$ and $\langle B, c \otimes a\rangle \longrightarrow^* \langle B', \rho' \otimes a\rangle$. Thus, by definition of R, $(\langle A', \sigma' \otimes a\rangle, \langle B', \rho' \otimes a\rangle) \in R$.

5. A Labelled Transition System for Soft CCP

Although $\approx_s$ is fully abstract, it is somewhat unsatisfactory because of the upward-closure, i.e., the quantification in condition *3* of Definition 25. Intuitively, this means that to prove that two agents are bisimilar *all* the possible stores should be tested in order to verify their equivalence, thus resulting in potentially inefficient proof techniques.

Definition 27 (Labelled reductions). *Let* $\Gamma = \mathcal{A} \times C^{\otimes}$ *be the set of* configurations *and* V *the set of variables. The* labelled direct reduction semantics *for SCCP is the pair* $\langle \Gamma, \mapsto \rangle$ *such that* $\mapsto \subseteq \Gamma \times \Gamma$ *is indexed over the couple* $\langle C^{\otimes}, 2^V \rangle$*, i.e.,* $\mapsto = \bigcup_{\alpha \in C^{\otimes}, \Delta \subseteq V} \overset{\alpha}{\mapsto}_{\Delta}$ *and* $\overset{\alpha}{\mapsto}_{\Delta} \subseteq \Gamma \times \Gamma$*, obtained by the rules in Table 3.*

The labelled reduction semantics *for SCCP is the pair* $\langle \Gamma, \rightarrow \rangle$ *such that* $\rightarrow \subseteq \Gamma \times \Gamma$ *is the family of binary relations indexed over the couple* $\langle C^{\otimes}, 2^V \rangle$*;* $\overset{\alpha}{\rightarrow}_{\Delta} \subseteq \Gamma \times \Gamma$ *can obtained by the rules in Table 3 and Table 4.*

In Table 3 and Table 4 we refine the notion of transition (respectively given in Table 1 and Table 2) by adding a label that carries additional information about the constraints that cause the reduction. Hence, we define a new labelled transition system (LTS) obtained by the family of relations $\overset{\alpha}{\mapsto}_{\Delta} \subseteq \Gamma \times \Gamma$ indexed over $\langle C^{\otimes}, 2^V \rangle$; as a reminder, Γ is the set of configurations, $C^{\otimes}$ the set of $\otimes$-compact constraints, and, as for the unlabelled semantics in Section 3, transitions are indexed by sets of variables. Rules in Table 3 and Table 4 are identical to those in Table 1 and Table 2, except for a constraint α that represents the minimal information that must be added to σ in order to fire an action from $\langle A, \sigma \rangle$ to $\langle A', \sigma' \rangle$, i.e., $\langle A, \sigma \otimes \alpha \rangle \longrightarrow_{\Delta} \langle A', \sigma' \rangle$.

Rule **LA2** says that $\langle \mathbf{ask}(c) \rightarrow A, \sigma \rangle$ can evolve to $\langle A, \sigma \otimes \alpha \rangle$ if the environment provides a minimal constraint α that added to the store σ entails c, i.e., $\alpha = c \oplus \sigma$. Notice that, differently from [2], here the definition of this minimal label comes directly from a derived operator of the underlying CLIM (i.e., from $\oplus$), which by Lemma 1 preserves $\otimes$-compactness.

Example 7 (Labelled ask). *We provide an example of agent computation to clarify the semantics in Table 3. Consider the CLIM* $\langle \mathbb{R}^+, \leq, + \rangle$*, and the associated CLIM of soft constraints, as in the examples of Section 2.4. Then, consider the constraints* $\sigma = x \hat{+} y$ *and* $c = 2x \hat{+} 3y$*, both with support* $\{x, y\}$*.*

We use the querying agent $\mathbf{ask}(c) \rightarrow A$*, executed in a store* σ*: then, according to rule* **LA2**, $\langle \mathbf{ask}(c) \rightarrow A, \sigma \rangle \overset{x \hat{+} 2y}{\longmapsto}_{\{x,y\}} \langle A, \sigma \otimes (x \hat{+} 2y) \rangle$*. The soft constraint*

LA1	$\dfrac{sv(\sigma) \cup sv(c) \subseteq \Delta}{\langle \mathbf{tell}(c), \sigma \rangle \overset{\perp}{\longmapsto}_{\Delta} \langle \mathbf{stop}, \sigma \otimes c \rangle}$	**Tell**
LA2	$\dfrac{sv(\sigma) \cup sv(c) \cup fv(A) \subseteq \Delta}{\langle \mathbf{ask}(c) \rightarrow A, \sigma \rangle \overset{c \oplus \sigma}{\longmapsto}_{\Delta} \langle A, \sigma \otimes (c \oplus \sigma) \rangle}$	**Ask**
LA3	$\dfrac{sv(\sigma) \cup \{y\} \subseteq \Delta \wedge p(x) = A \in \mathcal{P}}{\langle p(y), \sigma \rangle \overset{\perp}{\mapsto}_{\Delta} \langle A[^{y}/_{x}], \sigma \rangle}$	**Rec**
LA4	$\dfrac{sv(\sigma) \cup fv(\exists_x A) \subseteq \Delta \wedge w \notin \Delta}{\langle \exists_x A, \sigma \rangle \overset{\perp}{\mapsto}_{\Delta} \langle A[^{w}/_{x}], \sigma \rangle}$	**Hide**

Table 3: Axioms of the labelled semantics for SCCP.

LR1	$\dfrac{\langle A, \sigma \rangle \xrightarrow{\alpha}_{\Delta} \langle A', \sigma' \rangle \wedge fv(B) \subseteq \Delta}{\langle A \parallel B, \sigma \rangle \xrightarrow{\alpha}_{\Delta} \langle A' \parallel B, \sigma' \rangle}$	**Par1**
LR2	$\dfrac{\langle A, \sigma \rangle \xrightarrow{\alpha}_{\Delta} \langle A', \sigma' \rangle \wedge fv(B) \subseteq \Delta}{\langle B \parallel A, \sigma \rangle \xrightarrow{\alpha}_{\Delta} \langle B \parallel A', \sigma' \rangle}$	**Par2**

Table 4: Contextual rules of the labelled semantics for SCCP.

$c \ominus \sigma = x \hat{+} 2y$ *represents the minimal amount of information that must be added to* σ *in order to fire the ask action. And since the chosen CLIM is invertible, we also have that* $\sigma \otimes (x \hat{+} 2y) = c$.

If instead we define $\sigma = 2x \hat{+} 3y$ *and* $c = x \hat{+} y$, *then nothing needs to be added to the store in order to fire the same action, and* $\langle \mathbf{ask}(c) \to A, \sigma \rangle \overset{\perp}{\mapsto}_{\{x,y\}} \langle A, \sigma \rangle$.

The LTS is sound and complete with respect to the unlabelled semantics.

Lemma 13 (Soundness). *Let* A *be an agent and* $\sigma \in C^{\otimes}$. *If* $\langle A, \sigma \rangle \overset{\alpha}{\mapsto}_{\Delta} \langle A', \sigma' \rangle$ *then* $\langle A, \sigma \otimes \alpha \rangle \mapsto_{\Delta} \langle A', \sigma' \rangle$.

Proof 10. *We just consider* **LA2***: The other cases are straightforward.*

By ***LA2*** *we have* $A = \mathbf{ask}(c) \to A'$, $\alpha = c \ominus \sigma$, *and* $\sigma' = (\sigma \otimes (c \ominus \sigma))$. *Since it always holds that* $c \leq (\sigma \otimes (c \ominus \sigma))$ *then by* ***A2*** *we have* $\langle A, \sigma \otimes \alpha \rangle \mapsto_{\Delta} \langle A', \sigma' \rangle$.

We are going to need a stronger notion of completeness. Intuitively, whenever there exists a direct reduction $\langle A, \sigma \rangle \mapsto$, then a corresponding labelled direct reduction exists originating from any $\langle A, \rho \rangle$ such that $\rho \leq \sigma$.

Lemma 14 (Completeness). *Let* A *be an agent and* $\sigma, \rho \in C^{\otimes}$ *such that* $\rho \leq \sigma$. *Moreover, let us assume* C *to be invertible. If* $\langle A, \sigma \rangle \mapsto_{\Delta} \langle A', \sigma' \rangle$ *then there exist* $\alpha, a \in C^{\otimes}$ *such that* $\langle A, \rho \rangle \overset{\alpha}{\mapsto}_{\Delta} \langle A', \rho' \rangle$ *with* $\rho \otimes \alpha \otimes a = \sigma$ *and* $\rho' \otimes a = \sigma'$.

Proof 11. *We just consider* **LA2***: The other cases are straightforward and they are verified by always choosing* $\alpha = a = \perp$.

By ***LA2*** *we have* $A = \mathbf{ask}(c) \to A'$, $\sigma' = \sigma$, *and* $c \leq \sigma$. *Now consider* $\langle A, \rho \rangle \overset{\alpha}{\mapsto}_{\Delta} \langle A', \rho' \rangle$, *where* $\alpha = c \ominus \rho$ *and* $\rho' = \rho \otimes \alpha$. *Also, let us note that* $c \leq \sigma$ *and* $\rho \leq \sigma$ *imply* $\rho' \leq \sigma$ *by invertibility. Thus, we can take* $a = \sigma \ominus \rho'$, *and the conditions are easily verified again by the invertibility of* C.

The complex statement boils down to the checking for the satisfaction of a constraint c in a store ρ that does not necessarily verify it. This is the equivalent of [2, Lemma 5] for the crisp language, and it is going to be needed in Lemma 15 of the next section to prove that weak and strong bisimilarities are upward closed with respect to the store.

The lemmata above can in any case be simplified to state that the labelled reduction semantics is in fact an extension of the unlabelled one.

Proposition 3. *Let* A *be an agent and* $\sigma \in C^{\otimes}$. *Then* $\langle A, \sigma \rangle \xrightarrow{\perp}_{\Delta} \langle A', \sigma' \rangle$ *if and only if* $\langle A, \sigma \rangle \longrightarrow_{\Delta} \langle A', \sigma' \rangle$.

5.1. Strong and Weak Bisimilarity on the LTS.

We now define an equivalence that characterises $\sim_s$ without the upward closure condition. As it occurs with the crisp language, and differently from calculi such as Milner's CCS, barbs cannot be removed from the definition of bisimilarity because they cannot be inferred by the transitions.

Definition 28 (Strong bisimilarity). *A strong bisimulation is a symmetric relation R on configurations such that whenever $(\gamma_1, \gamma_2) \in R$*

1. *if $\gamma_1 \downarrow_c$ then $\gamma_2 \downarrow_c$ for all $c \in C^{\otimes}$;*
2. *if $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ then $\exists \gamma_2'$ such that $\gamma_1 \otimes \alpha \longrightarrow \gamma_2'$ and $(\gamma_1', \gamma_2') \in R$.*

We say that γ_1 and γ_2 are strongly bisimilar ($\gamma_1 \sim \gamma_2$) if there exists a strong bisimulation R such that $(\gamma_1, \gamma_2) \in R$.

Whenever σ and ρ are $\otimes$-compact elements, the first condition is equivalent to require $\sigma \leq \rho$. Thus $(\gamma_1, \gamma_2) \in R$ would imply that γ_1 and γ_2 have the same store. As for the second condition, we adopted a *semi-saturated* equivalence, introduced for CCP in [2]. In the bisimulation game a label can be simulated by a reduction including in the store the label itself.

Definition 29 (Weak bisimilarity). *A weak bisimulation is a symmetric relation R on configurations such that whenever $(\gamma_1, \gamma_2) \in R$*

1. *if $\gamma_1 \downarrow_c$ then $\gamma_2 \Downarrow_c$ for all $c \in C^{\otimes}$;*
2. *if $\gamma_1 \xrightarrow{\alpha} \gamma_1'$ then $\exists \gamma_2'$ such that $\gamma_1 \otimes \alpha \longrightarrow^* \gamma_2'$ and $(\gamma_1', \gamma_2') \in R$.*

We say that γ_1 and γ_2 are weakly bisimilar ($\gamma_1 \approx \gamma_2$) if there exists a weak bisimulation R such that $(\gamma_1, \gamma_2) \in R$.

With respect to the weak equivalence for crisp constraints, some of its characteristic equalities do not hold, e.g., $\mathbf{ask}(c) \rightarrow \mathbf{tell}(c) \not\approx \mathbf{stop}$. As usual, this is due to the fact that the underlying CLIM may not be idempotent.

We can now conclude by proving the equivalence between $\sim_s$ and $\sim$ and between $\approx_s$ and $\approx$ (hence, $\approx$ is further equivalent to $\sim_o$, using Proposition 2). We start by showing that $\sim$ is preserved under closure.

Lemma 15. *Let $\langle A, \sigma\rangle$, $\langle B, \rho\rangle$ be configurations and $c \in C^{\otimes}$. Moreover, let C be invertible. If $\langle A, \sigma\rangle \sim \langle B, \rho\rangle$, then $\langle A, \sigma \otimes c\rangle \sim \langle B, \rho \otimes c\rangle$.*

Proof 12. *We need to show that $R = \{(\langle A, \sigma \otimes a\rangle \sim \langle B, \rho \otimes a\rangle) \mid \langle A, \sigma\rangle \sim \langle B, \rho\rangle\}$ satisfies the two properties in Definition 28.*

i) From the hypothesis $\langle A, \sigma\rangle \sim \langle B, \rho\rangle$, we have that $\rho = \sigma$, thus $\langle A, \sigma \otimes a\rangle$ and $\langle B, \rho \otimes a\rangle$ satisfy the same barbs.

ii) Let us assume $\langle A, \sigma \otimes c\rangle \xrightarrow{\alpha} \langle A', \sigma'\rangle$, we need to prove the existence of B' and ρ' such that $\langle B, \rho \otimes c \otimes \alpha\rangle \to \langle B', \rho'\rangle$ and $(\langle A', \sigma'\rangle, \langle B', \rho'\rangle) \in R$. By Lemma 13 and Lemma 14 we obtain $\langle A, \sigma\rangle \xrightarrow{\alpha'} \langle A', \sigma''\rangle$ and the existence of a' such that $\sigma \otimes \alpha' \otimes a' = \sigma \otimes c \otimes \alpha$ (1) and $\sigma'' \otimes a' = \sigma'$ (2). By invertibility, we also get that (1') $\alpha' \otimes a' = c \otimes \alpha$. From the labelled transition of $\langle A, \sigma\rangle$ and the hypothesis $\langle A, \sigma\rangle \sim \langle B, \rho\rangle$, we have that $\langle B, \rho \otimes \alpha'\rangle \to \langle B', \rho''\rangle$, with $\langle A, \sigma''\rangle \sim \langle B, \rho''\rangle$ (3). By (1') we have $\langle B, \rho \otimes c \otimes \alpha\rangle = \langle B, \rho \otimes \alpha' \otimes a'\rangle$ and $\langle B, \rho \otimes \alpha' \otimes a'\rangle \to \langle B, \rho'' \otimes a'\rangle$ (due to operational monotonicity). Finally, by the definition of R and (3), we conclude that $(\langle A', \sigma'' \otimes a'\rangle, \langle B', \rho'' \otimes a'\rangle) \in R$, and, by (2), $\langle A', \sigma'' \otimes a'\rangle = \langle A', \sigma'\rangle$.

Theorem 3. *$\sim_s \subseteq \sim$. Moreover, let C be invertible. Then $\sim_s = \sim$*

Proof 13. *The inclusion $\sim \subseteq \sim_s$ can be proved by using Lemma 15.*

From $\sim$ to $\sim_s$. *We show that $R = \{(\langle A, \sigma\rangle, \langle B, \rho\rangle) \mid \langle A, \sigma\rangle \sim \langle B, \rho\rangle\}$ is a saturated bisimulation, i.e., if $(\langle A, \sigma\rangle, \langle B, \rho\rangle) \in R$ the conditions in Definition 23 hold.*

i) If $\langle A, \sigma\rangle \downarrow_c$, then we have $\langle B, \rho\rangle \downarrow_c$ by the hypothesis $\langle A, \sigma\rangle \sim \langle B, \rho\rangle$.

ii) Suppose that $\langle A, \sigma\rangle \to \langle A', \sigma'\rangle$. By Proposition 3 we have $\langle A, \sigma\rangle \xrightarrow{\perp} \langle A', \sigma'\rangle$. Since $\langle A, \sigma\rangle \sim \langle B, \rho\rangle$, then $\langle B, \rho \otimes \perp\rangle \to \langle B', \rho'\rangle$ with $\langle A', \sigma'\rangle \sim \langle B', \rho'\rangle$. Since $\rho = \rho \otimes \perp$, we have $\langle B, \rho\rangle \to \langle B', \rho'\rangle$.

iii) By Lemma 15, $(\langle A, \sigma \otimes c'\rangle, \langle B, \rho \otimes c'\rangle) \in R$ for all $c' \in C^{\otimes}$.

From $\sim_s$ to $\sim$. *We show that $R = \{(\langle A, \sigma\rangle, \langle B, \rho\rangle) \mid \langle A, \sigma\rangle \sim_s \langle B, \rho\rangle\}$ is a strong bisimulation, i.e., if $(\langle A, \sigma\rangle, \langle B, \rho\rangle) \in R$ the conditions in Definition 28 hold.*

i) If $\langle A, \sigma\rangle \downarrow_c$, then we have $\langle B, \rho\rangle \downarrow_c$ by the hypothesis $\langle A, \sigma\rangle \sim_s \langle B, \rho\rangle$.

ii) Suppose that $\langle A, \sigma\rangle \xrightarrow{\alpha} \langle A', \sigma'\rangle$. Then by Lemma 13 we have $\langle A, \sigma \otimes \alpha\rangle \to \langle A', \sigma'\rangle$. Since $\langle A, \sigma\rangle \sim_s \langle B, \rho\rangle$, then $\langle A, \sigma \otimes \alpha\rangle \sim_s \langle B, \rho \otimes \alpha\rangle$ and thus $\langle B, \rho \otimes \alpha\rangle \to \langle B', \rho'\rangle$ with $\langle A', \sigma'\rangle \sim_s \langle B', \rho'\rangle$.

To prove the correspondence between weak bisimulations, we need a result similar to Lemma 15. The key is the preservation of weak barbs by the addition of constraints to the store, which is trivial in the strong case.

Lemma 16. *Let $\langle A, \sigma\rangle$, $\langle B, \rho\rangle$ be configurations and $c, d \in C^{\otimes}$. If $\langle A, \sigma\rangle \approx \langle B, \rho\rangle$ and $\langle A, \sigma \otimes d\rangle \downarrow_c$, then $\langle B, \rho \otimes d\rangle \Downarrow_c$.*

Proof 14. *If $\langle A, \sigma \otimes a\rangle \downarrow_c$, then $c \leq \sigma \otimes d$. Since $\langle A, \sigma\rangle \approx \langle B, \rho\rangle$ and $\langle A, \sigma\rangle \downarrow_\sigma$, then there exists $\langle B', \rho'\rangle$ such that $\langle B, \rho\rangle \rightarrow^* \langle B', \rho'\rangle$ and $\sigma \leq \exists_\Gamma \rho'$ for $\Gamma = fv(\langle B', \rho'\rangle) \setminus fv(\langle B, \rho\rangle)$. Let us assume, without loss of generality, that $\Gamma \cap sv(d) = \emptyset$; since reductions are monotone (item 4 of Lemma 6), we have $\langle B, \rho \otimes d\rangle \rightarrow^* \langle B', \rho' \otimes d\rangle$. Finally, $c \leq \sigma \otimes d = \sigma \otimes \exists_\Gamma d \leq \exists_\Gamma \rho' \otimes \exists_\Gamma d \leq \exists_\Gamma(\rho' \otimes d)$, hence $\langle B, \rho \otimes d\rangle \Downarrow_c$.*

The result below uses Lemma 16 and rephrases the proof of Lemma 15.

Lemma 17. *Let $\langle A, \sigma\rangle$, $\langle B, \rho\rangle$ be configurations and $c \in C^{\otimes}$ Moreover, let us assume C to be invertible. If $\langle A, \sigma\rangle \approx \langle B, \rho\rangle$, then $\langle A, \sigma \otimes c\rangle \approx \langle B, \rho \otimes c\rangle$.*

Now the theorem below is proved by precisely mimicking the proof for Theorem 3 by using Lemma 17 instead of Lemma 15.

Theorem 4. *$\approx_s \subseteq \approx$. Moreover, let C be invertible. Then $\approx_s = \approx$.*

5.2. Labelled versus saturated semantics.

The main appeal of saturated semantics resides in being a congruence and, in fact, the minimal congruence contained in standard bisimulation [20]. The main drawback of this approach is that it is in principle necessary to check the behaviour of a process under every context. The problem is somewhat mitigated for SCCP, since it suffices to close the store with respect to any possible compact element (item 3 of Definition 23). At the same time, checking the feasibility of a reduction may require some computational effort, either for solving the combinatorial problem associated with calculating $\sigma \otimes d$, or for verifying if $c \leq \sigma$, as with agent $\mathbf{ask}(c) \rightarrow A$.

This is the reason for searching labelled semantics and suitable notions of bisimilarity that may alleviate such a burden. The intuition is to consider labels which somehow represent the "minimal context allowing a process to reduce", so that a bisimilarity-checking algorithm in principle needs to verify this minimal context only, instead of every one. The idea has been exploited in the framework of crisp CCP [2], and it is based on [21, 22].

Example 8. *Let us consider the agents* $\mathbf{ask}(c) \rightarrow \mathbf{stop}$ *and* $\mathbf{stop}$. *Intuitively, they are weak bisimilar because neither of them can make any move in the empty store. More formally, it has to be shown that* $\gamma \approx \gamma'$ *for configurations* $\gamma = \langle \mathbf{ask}(c) \rightarrow \mathbf{stop}, \bot \rangle$ *and* $\gamma' = \langle \mathbf{stop}, \bot \rangle$. *Consider the following relation*

$$\mathcal{R} = \{(\langle \mathbf{ask}(c) \rightarrow \mathbf{stop}, \bot \rangle, \langle \mathbf{stop}, \bot \rangle), (\langle \mathbf{stop}, c \rangle, \langle \mathbf{stop}, c \rangle)\}$$

It is quite easy to prove that it is a bisimulation, and in fact the smallest one identifying the two configurations. It suffices to note that by definition $c \ominus \bot = c$.

In order to prove that $\gamma \approx_s \gamma'$, *instead, we surely need to consider an infinite relation. Indeed, the smallest saturated bisimulation equating the two configurations is given by the relation below*

$$\mathcal{S} = \{(\langle \mathbf{ask}(c) \rightarrow \mathbf{stop}, d \rangle, \langle \mathbf{stop}, d \rangle), (\langle \mathbf{stop}, e \rangle, \langle \mathbf{stop}, e \rangle) \mid d, e \in C^{\otimes} \,\&\, c \leq e\}$$

The relation above is a saturated bisimulation, but any naive automatic check for that property might involve rather complex calculations.

Another reason for the complexity of checking saturated bisimilarity is the need of considering the closure $\longrightarrow^*$ of the reduction relation, which may cause a combinatorial explosion. Think of the two agents $\prod_{i \in I} \mathbf{ask}(c_i) \rightarrow \mathbf{stop}$ and $\mathbf{stop}$. With some ingenuity, they might be proved equivalent by exploiting the fact that saturated bisimilarity is a congruence, and by verifying that $\mathbf{stop} \parallel A \approx_s A$ for all the agents A. A direct proof would instead require a check for each store of the reductions arising from all the possible interleaving of the c_i elements.

6. An Axiomatisation of Simple Agents for Weak Bisimilarity

Once the behaviour of an agent is captured by an observational equivalence, it is natural to look for laws characterising it. Given its correspondence with the standard equivalence via fair computations, weak bisimilarity is the preferred behavioural semantics for soft CCP. A sound and complete axiomatisation was proposed for CCP in [1]. However, the lack of idempotence in the soft formalism makes unsound some of the axioms there, since posting a constraint twice is different from adding it just once. As an example, law *L1* of [1] states that $\mathbf{ask}(c) \rightarrow \mathbf{tell}(d) = \mathbf{ask}(c) \rightarrow (\mathbf{tell}(c) \parallel \mathbf{tell}(d))$, which is clearly false in the soft formalism.

$$\mathbf{tell}(\bot) = \mathbf{stop} \tag{1}$$
$$\mathbf{ask}(c) \to \mathbf{stop} = \mathbf{stop} \tag{2}$$
$$\mathbf{ask}(c) \to \mathbf{ask}(d) \to A = \mathbf{ask}(c \vee d) \to A \tag{3}$$
$$\mathbf{ask}(\bot) \to A = A \tag{4}$$
$$A \parallel B = B \parallel A \tag{5}$$
$$A \parallel (B \parallel C) = (A \parallel B) \parallel C \tag{6}$$
$$A \parallel \mathbf{stop} = A \tag{7}$$
$$\mathbf{tell}(c) \parallel \mathbf{tell}(d) = \mathbf{tell}(c \otimes d) \tag{8}$$
$$\mathbf{ask}(c) \to (A \parallel B) = (\mathbf{ask}(c) \to A) \parallel (\mathbf{ask}(c) \to B) \tag{9}$$
$$\mathbf{ask}(a) \to \mathbf{tell}(b) \parallel \mathbf{ask}(a \otimes b) \to \mathbf{tell}(d) = \mathbf{ask}(a) \to \mathbf{tell}(b \otimes d) \tag{10}$$
$$\mathbf{ask}(a) \to \mathbf{tell}(b) \parallel \mathbf{ask}(c) \to \mathbf{tell}(d) = \mathbf{ask}(a) \to \mathbf{tell}(b) \parallel \mathbf{ask}(c \vee (a \otimes b)) \to \mathbf{tell}(d) \text{ if } a \leq c \tag{11}$$

Figure 3: Axioms for simple agents.

Nevertheless, most of the axioms in [1] can be recovered, as long as we mix together the various operators available in a CLIM.[7] So, let an agent be simple if it is finite and contains no occurrence of an existential quantifier. Our set of axioms for simple agents of SCCP in presented in Figure 3.

As for sequential processes, in Equations. 1-2 we present the axioms involving the **stop** and in Equations 3-4 those related to *ask*. Those concerning the parallel composition operator are instead represented in Equations 5-7. Distributivity of *tell* and *ask* are presented in Equations 8-9, while Equations 10-11 show how further simplify the combination of *tell* and *ask* through parallel composition.

Proposition 4. *The axioms in Figure 3 are sound and complete for simple agents with respect to weak bisimilarity.*

Soundness is easily checked, while completeness is obtained by mimicking the proof schema adopted for the crisp case, exploiting a normal form that is in fact reminiscent of the one in [1, Definition 3.2].[8]

Lemma 18. *Let A be an agent. Then either $A = \mathbf{stop}$ or it can be decomposed as $||_i \mathbf{ask}(c_i) \to \mathbf{tell}(d_i)$ such that*

[7] Our axioms follow closely those of [1, p.341], replacing the law *L1* mentioned above with $\mathbf{tell}(\bot) = \mathbf{stop}$ and dropping altogether the laws *L11* and *L12*.

[8] With respect to the properties stated in [1, p.342], we weakened property 1 and dropped property 4, the latter being linked to axiom L12.

- $d_i \neq \bot$
- $c_i \neq c_j$ *for all* $i \neq j$
- $c_i < c_j$ *implies* $c_i \otimes d_i < c_j$

Axioms 1-8 guarantee that an agent can be decomposed as $||_i$ **ask**$(c_i) \rightarrow$ **tell**(d_i) such that $d_i \neq \bot$. Axiom 9 then ensures the second condition. Finally, by applying as long as possible first axiom 11 and then axiom 10, the final condition is also enforced.

7. Related Work

The aim of our work was to contribute to the topic of observational semantics for the family of CCP languages, which follows a more general approach for process calculi advanced in [23, 24]. In the following, we summarise the main features of related papers, commencing from non-soft languages, and then moving to similar works in the soft area.

There have been few attempts to define a notion of bisimilarity for CCP. The ones we are aware of are those in [1], [2] and [25]. The equivalences defined in [1] and [25] are not completely satisfactory in some cases: [1] may tell apart processes with identical behaviour (i.e., it is too fine grained), while [25] quantifies over all possible inputs from the environment, and hence it is not clear whether it can lead to a feasible proof technique. The more recent solution in [2] overcomes such limitations by defining a notion of bisimilarity for CCP that allows for benefiting of the feasible proof and verification techniques typically associated with bisimilarity. Indeed, the latter has been the starting point for our work.

The panorama of observational semantics is even more limited for the generalisation of CCP languages to soft paradigms [3], which is instead our main focus. There is only a couple of works in this direction we are aware of. In [5] the authors present an access-control oriented constraint-language based on soft constraints, and then only advance some results on *syntactic bisimilarity*. In [26] the authors define a labelled transition system for the *cc-pi* calculus, and a notion of open bisimilarity *à la* pi-calculus that is proved to be a congruence. The cc-pi calculus combines the synchronous communication paradigm of process calculi with the constraint handling mechanism of concurrent constraint programming. Note that the constraints in [26] are the soft constraints defined in [3].

Close observational semantics are related to *tuple*-based calculi. A tuple space is a multiset of tuples that are sequences of information items, which can be associatively selected from tuple spaces by means of a pattern-matching mechanism. Tuples can contain both values and code that can be subsequently accessed and evaluated. In these languages, tuples "substitute" constraints as information tokens: agents interact through a tuple store instead of a constraint store.

One of the earliest tuple-based language is *Linda* [27], which is a coordination language whose primitives operate on tuples. The basic primitives are: **in**, which atomically reads and removes (i.e., consumes) a tuple from tuple-space, **rd** non-destructively reads a tuple-space, **out** produces a tuple, writing it into tuple-space, and **eval** creates new processes to evaluate tuples, writing the result into tuple-space. A follow-up language is *Klaim*, which rests on an extension of Linda. A survey of obervational semantics for such languages can be e.g found in [28] and [29], respectively. While we believe that a proper comparison with these semantics is out of the scope of our work, which focuses on the characterisation of the standard semantics of SCCP based on observables of infinite computations with a newly introduced labelled one, we remark that the approach we investigated in this paper could be extended to labelled operational-semantics of Linda, along the lines of [30], where for instance a derivation $A \xrightarrow{\neg a} A'$ indicates that A becomes A' if tuple $\langle a \rangle$ is not available in the store.

8. Conclusions and Further Work

Inspired by the characterisation of the labelled semantics for the crisp variant of the language [2], in this paper we investigated observational and behavioural equivalences of the deterministic fragment of soft CCP [3]: we introduced the notion of $\otimes$-compactness, we proposed an observational semantics for the language and presented a novel behavioural characterisation in terms of weak bisimilarity (enhancing the syntactic bisimulation advanced in [5]). We then rounded the work by introducing an axiomatisation of the finite fragment of the language, in the spirit of [1].

The use of residuation theory (advanced in [13] for solving soft constraints problems) provides an elegant way to define the minimal information that enables the firing of actions in the LTS shown in Sec. 5. This choice allowed for the study of the behavioural equivalence of agents in terms of weak and strong bisimilarity on such LTS, and it allowed for relating

them to the corresponding barbed bisimilarities of (unlabelled) reductions and with the standard semantics via fair computations. The two kinds of equivalences, as well as the axiomatisation for weak bisimilarity, are presented in this paper for the first time.

For future work, we plan to provide a denotational semantics for soft CCP by building on the work for the crisp case in [1]. No denotational semantics has been yet proposed for [3]. Concerning the axioms, we will try and investigate the relationship between soft CCP and a logical system whose fundamental properties are closely related to the ones we have investigated in this paper; namely *affine linear logic* [31]. This logical system *rejects contraction* but *admits weakening*, which intuitively correspond to dropping idempotence and preserving monotonicity in the soft formalism. The denotational model of CCP is based on *closure operators*: Each agent is compositionally interpreted as a monotonic, extensive and idempotent operator/function on constraints. We shall then investigate a denotational model for soft CCP processes based on *pre-closure operators* [32].

We plan to consider two extensions of the language, checking how far the results given in this paper can be adapted. As evidenced by [33], a non-deterministic extension is an interesting challenge since the closure under any context for the saturated bisimilarity gets more elaborated than just closing with respect to the addition of constraints (Definition 23 and Definition 25, condition 3), and similarly one also needs to find the right formulation of bisimilarity for the labelled transitions systems. Also, the presence of residuation makes intuitive the definition of a retract operator for the calculus. Even if the operational semantics would be less affected, retraction requires a complete reformulation of the semantics via fair computations, since monotonicity (as stated in Lemma 6) would not hold [5].

Finally, we would like to find the same observational equivalences also for a semantics that considers local stores of agents. In this case, the hiding operator needs to carry some information on the variables it abstracts. According to [34], we should consider an extended operator $\exists^{\sigma}_{x}$, for σ the local store. The intuition is that variable x may be local to a component $\exists_x\sigma'$ of the store σ, yet visible at a global level: we must then evaluate A in the store when the local x is hidden, yet the possible duplications are removed (e.g., $\exists_x\sigma'$ may already occur in the global store σ). The final store contains in σ_1 the original σ' increased by what has been added by the step. To this end, we consider also to investigate the more general framework for distilling labelled semantics from unlabelled ones proposed in [24].

References

[1] V. A. Saraswat, M. C. Rinard, P. Panangaden, Semantic foundations of concurrent constraint programming, in: D. S. Wise (Ed.), POPL 1991, ACM Press, 1991, pp. 333–352.

[2] A. Aristizábal, F. Bonchi, C. Palamidessi, L. F. Pino, F. D. Valencia, Deriving labels and bisimilarity for concurrent constraint programming, in: M. Hofmann (Ed.), FOSSACS 2011, Vol. 6604 of LNCS, Springer, 2011, pp. 138–152.

[3] S. Bistarelli, U. Montanari, F. Rossi, Soft concurrent constraint programming, ACM Transactions on Computational Logic 7 (3) (2006) 563–589.

[4] M. G. Buscemi, U. Montanari, CC-Pi: A constraint-based language for specifying service level agreements, in: R. De Nicola (Ed.), ESOP 2007, Vol. 4421 of LNCS, Springer, 2007, pp. 18–32.

[5] S. Bistarelli, F. Santini, A secure non-monotonic soft concurrent constraint language, Fundamenta Informaticae 134 (3-4) (2014) 261–285.

[6] S. Bistarelli, M. Gabbrielli, M. C. Meo, F. Santini, Timed soft concurrent constraint programs, in: D. Lea, G. Zavattaro (Eds.), COORDINATION, Vol. 5052 of LNCS, 2008, pp. 50–66.

[7] S. Bistarelli, M. Gabbrielli, M. C. Meo, F. Santini, Timed soft concurrent constraint programs: An interleaved and a parallel approach, Theory and Practice of Logic Programming 15 (6) (2015) 743–782.

[8] F. Gadducci, F. Santini, L. F. Pino, F. D. Valencia, A labelled semantics for soft concurrent constraint programming, in: T. Holvoet, M. Viroli (Eds.), COORDINATION 2015, LNCS, Springer, 2015, pp. 133–149.

[9] G. Karner, Semiring-based constraint satisfaction and optimization, Semigroup Forum 45 (XX) (1992) 148–165.

[10] H. Fargier, J. Lang, Uncertainty in constraint satisfaction problems: a probabilistic approach, in: M. Clarke, R. Kruse, S. Moral (Eds.), ECSQUARU 1993, Vol. 747 of LNCS, Springer, 1993, pp. 97–104.

[11] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint satisfaction and optimization, Journal of ACM 44 (2) (1997) 201–236.

[12] J. Golan, Semirings and Affine Equations over Them: Theory and Applications, Kluwer, 2003.

[13] S. Bistarelli, F. Gadducci, Enhancing constraints manipulation in semiring-based formalisms, in: G. Brewka, S. Coradeschi, A. Perini, P. Traverso (Eds.), ECAI 2006, Vol. 141 of FAIA, IOS Press, 2006, pp. 63–67.

[14] N. Galatos, P. Jipsen, T. Kowalski, H. Ono, Residuated Lattices: An Algebraic Glimpse at Substructural Logics, Elsevier, 2007.

[15] F. Baccelli, G. Cohen, G. Olsder, J.-P. Quadrat, Synchronization and Linearity: An Algebra for Discrete Event Systems, Wiley, 1992.

[16] J. D. Monk, An introduction to cylindric set algebras, Logic Journal of IGPL 8 (4) (2000) 451–496.

[17] M. Z. Kwiatkowska, Infinite behaviour and fairness in concurrent constraint programming, in: J. W. de Bakker, W. P. de Roever, G. Rozenberg (Eds.), Sematics: Foundations and Applications, Vol. 666 of LNCS, Springer, 1992, pp. 348–383.

[18] S. Nyström, B. Jonsson, A fully abstract semantics for concurrent constraint programming, Information and Computation 146 (2) (1998) 138–180.

[19] R. Milner, D. Sangiorgi, Barbed bisimulation, in: W. Kuich (Ed.), ICALP 1992, Vol. 623 of LNCS, Springer, 1992, pp. 685–695.

[20] U. Montanari, V. Sassone, Dynamic congruence vs. progressing bisimulation for CCS, Fundamenta informaticae 16 (2) (1992) 171–199.

[21] J. J. Leifer, R. Milner, Deriving bisimulation congruences for reactive systems, in: C. Palamidessi (Ed.), CONCUR 2000, LNCS, 2000, pp. 243–258.

[22] F. Bonchi, F. Gadducci, G. V. Monreale, Reactive systems, barbed semantics, and the mobile ambients, in: L. de Alfaro (Ed.), FOSSACS 2009, LNCS, 2009, pp. 272–287.

[23] F. Bonchi, B. König, U. Montanari, Saturated semantics for reactive systems, in: 21th IEEE Symposium on Logic in Computer Science (LICS), IEEE Computer Society, 2006, pp. 69–80.

[24] F. Bonchi, F. Gadducci, G. V. Monreale, A general theory of barbs, contexts, and labels, ACM Transactions on Computational Logic 15 (4) (2014) 35:1–35:27.

[25] N. P. Mendler, P. Panangaden, P. J. Scott, R. A. G. Seely, A logical view of concurrent constraint programming, Nordic Journal of Computing 2 (2) (1995) 181–220.

[26] M. G. Buscemi, U. Montanari, Open bisimulation for the concurrent constraint pi-calculus, in: S. Drossopoulou (Ed.), ESOP 2008, Vol. 4960 of Lecture Notes in Computer Science, Springer, 2008, pp. 254–268.

[27] N. Carriero, D. Gelernter, Linda in context, Communications of ACM 32 (4) (1989) 444–458.

[28] R. De Nicola, R. Pugliese, An observational semantics for Linda, in: J. Desel (Ed.), STRICT 1995, Springer, 1995, pp. 129–143.

[29] R. De Nicola, D. Gorla, R. Pugliese, On the expressive power of KLAIM-based calculi, Theoretical Computer Science 356 (3) (2006) 387–421.

[30] N. Busi, R. Gorrieri, G. Zavattaro, On the expressiveness of Linda coordination primitives, Information and Computation 156 (1-2) (2000) 90–121.

[31] U. Dal Lago, S. Martini, Phase semantics and decidability of elementary affine logic, Theoretical Computer Science 318 (3) (2004) 409–433.

[32] A. Arkhangelskii, L. Pontryagin, General Topology I, Springer, 1990.

[33] L. F. Pino, F. Bonchi, F. D. Valencia, A behavioral congruence for concurrent constraint programming with non-deterministic choice, in: G. Ciobanu, D. Méry (Eds.), ICTAC 2014, Vol. 8687 of LNCS, Springer, 2014, pp. 351–368.

[34] F. S. de Boer, M. Gabbrielli, E. Marchiori, C. Palamidessi, Proving concurrent constraint programs correct, Transactions on Programming Languages and Systems 19 (5) (1997) 685–725.

Highlights of:
Observational and Behavioural Equivalences for Soft Concurrent Constraint Programming

Inspired by the characterisation of the labelled semantics for the crisp variant of the language [1], in this paper

- we propose an observational semantics for the deterministic fragment of Soft CCP [2] based on fair computations
- we enhance the current algebraic treatment of the domains used for representing the preferences in (soft) constraints
- we introduce strong and weak bisimulation equivalences for SCCP (enhancing the syntactic one advanced in [3])
- we present a characterisation of the observational semantics in terms of the weak (labelled) bisimilarity
- we round the work by introducing an axiomatisation of the finite fragment of the language, in the spirit of [4]

A key aspect of CCP is the idempotency of the operator for composing constraints: adding the same information twice does not change the store. Dropping idempotency requires a full reworking of the theory.

[1] A. Aristizabal, F. Bonchi, C. Palamidessi, L.F. Pino, F.D. Valencia, Deriving labels and bisimilarity for concurrent constraint programming, in: M. Hofmann (Ed.), FOSSACS 2011, Vol. 6604 of LNCS, Springer, 2011, pp. 138–152.

[2] S. Bistarelli, U. Montanari, F. Rossi, Soft concurrent constraint programming, ACM Transactions on Computational Logic 7 (3) (2006) 563–589.

[3] S. Bistarelli, F. Santini, A secure non-monotonic soft concurrent constraint language, Fundamenta Informaticae 134 (3-4) (2014) 261–285.

[4]V.A. Saraswat, M.C. Rinard, P. Panangaden, Semantic foundations of concurrent constraint programming, in: D. S. Wise (Ed.), POPL 1991, ACM Press, 1991, pp. 333–352.