Unique solutions of contractions, CCS, and their HOL formalisation

Chun Tian
Fondazione Bruno Kessler*
Trento, Italy
ctian@fbk.eu

Davide Sangiorgi
Università di Bologna and INRIA
Bologna, Italy
davide.sangiorgi@unibo.it

The unique solution of contractions is a proof technique for bisimilarity that overcomes certain syntactic constraints of Milner's "unique solution of equations" technique. The paper presents an overview of a rather comprehensive formalisation of the core of the theory of CCS in the HOL theorem prover (HOL4), with a focus towards the theory of unique solutions of contractions. (The formalisation consists of about 20,000 lines of proof scripts in Standard ML.) Some refinements of the theory itself are obtained. In particular we remove the constraints on summation, which must be weakly-guarded, by moving to *rooted contraction*, that is, the coarsest precongruence contained in the contraction preorder.

1 Introduction

A prominent proof method for bisimulation, put forward by Milner and widely used in his landmark CCS book [18] is the *unique solution of equations*, whereby two tuples of processes are componentwise bisimilar if they are solutions of the same system of equations. This method is important in verification techniques and tools based on algebraic reasoning [25, 4, 24].

In the versions of Milner's unique solution theorems for proving that all solutions are weakly (or rooted) bisimilar (in practice these are the most relevant cases), however, Milner's proof method has severe syntactical limitations, such that the equations must be "guarded and sequential," that is, the variables of the equations may only be used underneath a visible prefix and preceed, in the syntax tree, only by the sum and prefix operators. One way of overcoming such limitations is to replace equations with special inequations called *contractions* [27, 28]. Contraction is a preorder that, roughly, places some efficiency constraints on processes. The uniqueness of solutions of a system of contractions is defined as with systems of equations: any two solutions must be bisimilar. The difference with equations is in the meaning of a solution: in the case of contractions the solution is evaluated with respect to the contraction preorder, rather than bisimilarity. With contractions, most syntactic limitations of the unique-solution theorem can be removed. One constraint that still remains in [28] (in which the issue is bypassed using a more restrictive CCS syntax) is the occurrences of direct sums, due to the failure of the substitutivity of contraction under direct sums.

The main goal of the work described in this paper is a rather comprehensive formalisation of the core of the theory of CCS in the HOL theorem prover (HOL4), with a focus on the theory of unique solutions of contractions. The formalisation, however, is not confined to the theory of unique solutions of equations, but embraces a significant portion the theory of CCS [18] (mostly because the theory of unique solutions relies on a large number of more fundamental results). Indeed the formalisation encompasses the basic properties of strong and weak bisimilarity (e.g. the fixed-point and substitutivity

*Part of this work was carried out when the first author was studying at Università di Bologna.

Submitted to:
EXPRESS/SOS 2018

properties), the basic properties of rooted bisimilarity (the congruence induced by weak bisimilarity, also called observation congruence), and their algebraic laws. Further extensions (beyond Nesi [19]) include four versions of "bisimulation up to" techniques (e.g., bisimulation up-to bisimilarity), and of the expansion and contraction preorder (two efficiency-like refinements of weak bisimilarity). Concerning rooted bisimilarity, the formalisation includes Hennessy Lemma and Deng Lemma (Lemma 4.1 and 4.2 of [13]), and two long proofs saying the rooted bisimilarity is the largest (coarsest) congruence contained in (weak) bisimilarity: one following Milner's book [18], with the hypothesis that no processes can use up all labels; the other without such hypothesis, essentially formalising van Glabbeek's paper [11]. Similar theorems are proved for the rooted contraction preorder. In this respect, the work is an extensive experiment with the use of the HOL theorem prover and its most recent developments, including a package for expressing coinductive definitions.

From the view of CCS theory, this formalisation has offered us the possibility of further refining the theory of unique solutions of equations, as formally proving previous previously known results gives us a chance to see *what's really needed* for establishing that results. In particular, the existing theory has placed limitations on the body of the contractions due to the substitutivity problems of weak bisimilarity and other behavioural relations with respect to the sum operator. We have thus refined the contraction-based proof technique, by moving to *rooted contraction*, that is, the coarsest precongruence contained in the contraction preorder. The resulting unique-solution theorem is now valid for *rooted bisimilarity* (hence also for bisimilarity itself), and places no constraints on the occurrences of sums.

Another advantage of the formalisation is that we can take advantage of results about different equivalences or preorders that share a similar proof structure. Examples are: the results that rooted bisimilarity and rooted contraction are, respectively, the coarsest congruence contained in weak bisimilarity and the coarsest precongruence contained in the contraction preorder; the result about unique solution of equations for weak bisimilarity that uses the contraction preorder as an auxiliary relation, and other unique solution results (e.g., the one for rooted in which the auxiliary relation is rooted contraction); various forms of enhancements of the bisimulation proof method (the 'up-to' techniques) [18, 29]. In these cases, moving between proofs there are only a few places in which the HOL scripts have to be modified. Then the successful termination of the scripts gives us a guarantee that the proof is completed, removing the risk of overlooking or missing details as in hand-written proofs.

Structure of the paper Section 2 presents basic background materials on CCS, including its syntax, operational semantics, bisimilarity and rooted bisimilarity. Section 3 discussed equations and contractions. Section 4 presents rooted contraction and the related unique-solution result for rooted bisimilarity. Section 5 highlights our formalisation in HOL4. Finally, Section 6 and 7 discuss related work, conclusions, and a few directions for future work.

2 CCS

We assume a possibly infinite set of *names* $\mathscr{L} = \{a, b, \ldots\}$, which does not contain the special symbol τ, and a set of variables $A, B, \ldots$ for writing recursive behaviours. The class of the CCS processes is inductively built from $\mathbf{0}$ by the operators of prefixing, parallel composition, sum, restriction, recursion and relabeling:

$$\begin{array}{lcl} \mu & := & \tau \mid a \mid \overline{a} \\ P & := & \mathbf{0} \mid \mu.P \mid P_1 \mid P_2 \mid P_1 + P_2 \mid (\boldsymbol{\nu} a)\, P \mid A \mid \mathtt{rec}\, A.P \mid P\,[rf] \end{array}$$

$$\frac{}{\mu.P \xrightarrow{\mu} P} \qquad \frac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'} \qquad \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\frac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu} a)\,P \xrightarrow{\mu} (\boldsymbol{\nu} a)\,P'}\ \mu \neq a, \overline{a} \qquad \frac{P\{\mathtt{rec}\,A.P/A\} \xrightarrow{\mu} P'}{\mathtt{rec}\,A.P \xrightarrow{\mu} P'} \qquad \frac{P \xrightarrow{\mu} P'}{P\,[rf] \xrightarrow{rf(\mu)} P'\,[rf]}\ \forall a.\, rf(\overline{a}) = \overline{rf(a)}$$

Figure 1: Structural Operational Semantics of CCS

The operational semantics of CCS is given by means of a Labeled Transition System (LTS), shown in Fig. 1 (the symmetric version of the two rules for parallel composition and the rule for sum are omitted). A CCS expression uses only *weakly-guarded sums* if all occurrences of the sum operator are of the form $\mu_1.P_1 + \mu_2.P_2 + \ldots + \mu_n.P_n$, for some $n \geq 2$. The *immediate derivatives* of a process P are the elements of the set $\{P' \mid P \xrightarrow{\mu} P' \text{ for some } \mu\}$. Some standard notations for transitions: $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\stackrel{\mu}{\Longrightarrow}$ is $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$ (the composition of the three relations). Moreover, $P \stackrel{\widehat{\mu}}{\longrightarrow} P'$ holds if $P \xrightarrow{\mu} P'$ or ($\mu = \tau$ and $P = P'$); similarly $P \stackrel{\widehat{\mu}}{\Longrightarrow} P'$ holds if $P \stackrel{\mu}{\Longrightarrow} P'$ or ($\mu = \tau$ and $P = P'$). We write $P\,(\xrightarrow{\mu})^n P'$ if P can become P' after performing n μ-transitions. Finally, $P \xrightarrow{\mu}$ holds if there is P' with $P \xrightarrow{\mu} P'$, and similarly for other forms of transitions.

Further notations Letters $\mathcal{R}$, $\mathcal{S}$ range over relations. We use infix notation for relations, e.g., $P\,\mathcal{R}\,Q$ means that $(P,Q) \in \mathcal{R}$. We use a tilde to denote a tuple, with countably many elements; thus the tuple may also be infinite. All notations are extended to tuples componentwise; e.g., $\widetilde{P}\,\mathcal{R}\,\widetilde{Q}$ means that $P_i\,\mathcal{R}\,Q_i$, for each component i of the tuples $\widetilde{P}$ and $\widetilde{Q}$. And $C[\widetilde{P}]$ is the process obtained by replacing each hole $[\cdot]_i$ of the context C with P_i. We write $\mathcal{R}^{\mathrm{c}}$ for the closure of a relation under contexts. Thus $P\,\mathcal{R}^{\mathrm{c}}\,Q$ means that there are a context C and tuples $\widetilde{P}, \widetilde{Q}$ with $P = C[\widetilde{P}], Q = C[\widetilde{Q}]$ and $\widetilde{P}\,\mathcal{R}\,\widetilde{Q}$. We use symbol $\stackrel{\text{def}}{=}$ for abbreviations. For instance, $P \stackrel{\text{def}}{=} G$, where G is some expression, means that P stands for the expression G. If $\leq$ is a preorder, then $\geq$ is its inverse (and conversely).

2.1 Bisimilarity and rooted bisimilarity

The equivalences we consider are mainly the *weak* ones, in that they abstract from the number of internal steps performed:

Definition 2.1. *A process relation $\mathcal{R}$ is a* bisimulation *if, whenever $P\,\mathcal{R}\,Q$, we have:*

1. *$P \xrightarrow{\mu} P'$ implies that there is Q' such that $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $P'\,\mathcal{R}\,Q'$;*
2. *$Q \xrightarrow{\mu} Q'$,implies that there is P' such that $P \stackrel{\widehat{\mu}}{\Longrightarrow} P'$ and $P'\,\mathcal{R}\,Q'$.*

*P and Q are **bisimilar**, written as $P \approx Q$, if $P\,\mathcal{R}\,Q$ for any bisimulation $\mathcal{R}$.*

We sometimes call bisimilarity the *weak* one, to distinguish it from *strong bisimilarity* ($\sim$) obtained by replacing in the above definition the weak answer $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ with the strong $Q \xrightarrow{\mu} Q'$. Bisimilarity (the weak one) is not preserved by the sum operator (except for guarded sums). For this, Milner introduced *observational congruence*, also called *rooted bisimilarity* [13, 26]:

Definition 2.2. *Two processes P and Q are* rooted bisimilar*, written $P \approx^{\mathrm{c}} Q$, if we have:*

1. *$P \xrightarrow{\mu} P'$ implies that there is Q' such that $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \approx Q'$;*

2. *$Q \xrightarrow{\mu} Q'$ implies that there is P' such that $P \stackrel{\mu}{\Rightarrow} P'$ and $P' \approx Q'$.*

Theorem 2.3. *$\approx^{\mathrm{c}}$ is a congruence in CCS, and it is the coarsest congruence contained in $\approx$.*

3 Equations and contractions

3.1 Systems of equations

Uniqueness of solutions of equations [18] intuitively says that if a context C obeys certain conditions, then all processes P that satisfy the equation $P \approx C[P]$ are bisimilar with each other. We need variables to write equations. We use capital letters X, Y, Z for these variables and call them *equation variables*. The body of an equation is a CCS expression possibly containing equation variables. Thus such expressions, ranged over by E, live in the CCS grammar extended with equation variables.

Definition 3.1. *Assume that, for each i of a countable indexing set I, we have variables X_i, and expressions E_i possibly containing such variables. Then $\{X_i = E_i\}_{i\in I}$ is a* system of equations. *(There is one equation for each variable X_i.)*

We write $E[\widetilde{P}]$ for the expression resulting from E by replacing each variable X_i with the process P_i, assuming $\widetilde{P}$ and $\widetilde{X}$ have the same length. (This is syntactic replacement.)

Definition 3.2. *Suppose $\{X_i = E_i\}_{i\in I}$ is a system of equations:*

- *$\widetilde{P}$ is a* solution of the system of equations for $\approx$ *if for each i it holds that $P_i \approx E_i[\widetilde{P}]$;*
- *it has* a unique solution for $\approx$ *if whenever $\widetilde{P}$ and $\widetilde{Q}$ are both solutions for $\approx$, then $\widetilde{P} \approx \widetilde{Q}$.*

For instance, the solution of the equation $X = a.X$ is the process $R \stackrel{\text{def}}{=} \mathtt{rec}\, A.\,(a.A)$, and for any other solution P we have $P \approx R$. In contrast, the equation $X = a \mid X$ has solutions that may be quite different, for instance, K and $K \mid b$, for $K \stackrel{\text{def}}{=} \mathtt{rec}\, K.\,(a.K)$. (Actually any process capable of continuously performing a actions (while behaves arbitrarily on other actions) is a solution for $X = a \mid X$.)

Definition 3.3 (guardedness of equations)**.** *A system of equations $\{X_i = E_i\}_{i\in I}$ is*

- weakly guarded *if, in each E_i, each occurrence of an equation variable is underneath a prefix;*
- (strongly) guarded *if, in each E_i, each occurrence of an equation variable is underneath a* ***visible*** *prefix;*
- sequential *if, in each E_i, each of its subexpressions with occurrence of an equation variable, apart from the variable itself, is in forms of prefixes or sums.*

Theorem 3.4 (unique solution of equations, [18])**.** *A system of guarded and sequential equations (without direct sums) $\{X_i = E_i\}_{i\in I}$ has a unique solution for $\approx$.*

To see the need of the sequentiality condition, consider the equation (from [18]) $X = \boldsymbol{\nu} a\,(a.X \mid \overline{a})$ where X is guarded but not sequential. Any process that does not use a is a solution.

3.2 Contractions

The constraints on the unique-solution Theorem 3.4 can be weakened if we move from equations to inequations called *contractions*.

Intuitively, the bisimilarity contraction $\succeq_{\mathrm{bis}}$ is a preorder in which $P \succeq_{\mathrm{bis}} Q$ holds if $P \approx Q$ and, in addition, Q has the *possibility* of being at least as efficient as P (as far as τ-actions performed). Process Q, however, may be nondeterministic and may have other ways of doing the same work, and these could be slow (i.e., involving more τ-steps than those performed by P).

Definition 3.5. *A process relation $\mathcal{R}$ is a* bisimulation contraction *if, whenever $P\,\mathcal{R}\,Q$,*

1. *$P \xrightarrow{\mu} P'$ implies there is Q' such that $Q \stackrel{\widehat{\mu}}{\longrightarrow} Q'$ and $P'\,\mathcal{R}\,Q'$;*
2. *$Q \xrightarrow{\mu} Q'$ implies there is P' such that $P \stackrel{\widehat{\mu}}{\Longrightarrow} P'$ and $P' \approx Q'$.*

Bisimilarity contraction*, written $\succeq_{\rm bis}$, is the union of all bisimulation contractions.*

In the first clause Q is required to match P's challenge transition with at most one transition. This makes sure that Q is capable of mimicking P's work at least as efficiently as P. In contrast, the second clause of Definition 3.5, on the challenges from Q, entirely ignores efficiency: it is the same clause of weak bisimulation — the final derivatives are even required to be related by $\approx$, rather than by $\mathcal{R}$.

Bisimilarity contraction is coarser than the *expansion relation* $\succeq_{\rm e}$ [27, 3]. This is a preorder widely used in proof techniques for bisimilarity and that intuitively refines bisimilarity by formalising the idea of 'efficiency' between processes. Clause (1) is the same in the two preorders. But in clause (2) expansion uses $P \stackrel{\mu}{\Longrightarrow} P'$, rather than $P \stackrel{\widehat{\mu}}{\Longrightarrow} P'$; moreover with contraction the final derivatives are simply required to be bisimilar. An expansion $P \succeq_{\rm e} Q$ tells us that Q is always at least as efficient as P, whereas the contraction $P \succeq_{\rm bis} Q$ just says that Q has the possibility of being at least as efficient as P.

Example 3.6. *We have $a \not\succeq_{\rm bis} \tau.a$. However, $a+\tau.a \succeq_{\rm bis} a$, as well as its converse, $a \succeq_{\rm bis} a+\tau.a$. Indeed, if $P \approx Q$ then $P \succeq_{\rm bis} P+Q$. The last two relations do not hold with $\succeq_{\rm e}$, which explains the strictness of the inclusion $\succeq_{\rm e} \subset \succeq_{\rm bis}$.*

Like (weak) bisimilarity and expansion, contraction is preserved by all operators but sum.

3.3 Systems of contractions

A *system of contractions* is defined as a system of equations, except that the contraction symbol $\succeq$ is used in the place of the equality symbol $=$. Thus a system of contractions is a set $\{X_i \succeq E_i\}_{i\in I}$ where I is an indexing set and expressions E_i may contain the contraction variables $\{X_i\}_{i\in I}$.

Definition 3.7. *Given a system of contractions $\{X_i \succeq E_i\}_{i\in I}$, we say that:*

- *$\widetilde{P}$ is a* solution (for $\succeq_{\rm bis}$) of the system of contractions *if $\widetilde{P} \succeq_{\rm bis} \widetilde{E}[\widetilde{P}]$;*
- *the system has* a unique solution (for $\approx$) *if $\widetilde{P} \approx \widetilde{Q}$ whenever $\widetilde{P}$ and $\widetilde{Q}$ are both solutions.*

The guardedness of contractions follows Def. 3.3 (for equations).

Lemma 3.8. *Suppose $\widetilde{P}$ and $\widetilde{Q}$ are solutions for $\succeq_{\rm bis}$ of a system of weakly-guarded contractions that uses weakly-guarded sums. For any context C that uses weakly-guarded sums, if $C[\widetilde{P}] \stackrel{\mu}{\Longrightarrow} R$, then there is a context C' that uses weakly-guarded sums such that $R \succeq_{\rm bis} C'[\widetilde{P}]$ and $C[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Longrightarrow}\approx C'[\widetilde{Q}]$.* [1]

Proof. (sketch from [28]) Let n be the length of the transition $C[\widetilde{P}] \stackrel{\mu}{\Longrightarrow} R$ (the number of 'strong steps' of which it is composed), and let $C''[\widetilde{P}]$ and $C''[\widetilde{Q}]$ be the processes obtained from $C[\widetilde{P}]$ and $C[\widetilde{Q}]$ by unfolding the definitions of the contractions n times. Thus in C'' each hole is underneath at least n prefixes, and cannot contribute to an action in the first n transitions; moreover all the contexts have only weakly-guarded sums.

We have $C[\widetilde{P}] \succeq_{\rm bis} C''[\widetilde{P}]$, and $C[\widetilde{Q}] \succeq_{\rm bis} C''[\widetilde{Q}]$, by the substitutivity properties of $\succeq_{\rm bis}$ (we exploit here the syntactic constraints on sums). Moreover, since each hole of the context C'' is underneath at

[1] There's no typo here: $C[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Longrightarrow}\approx C'[\widetilde{Q}]$ means $\exists \widetilde{R}.\ C[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Longrightarrow} \widetilde{R} \approx C'[\widetilde{Q}]$. Same as in Lemma 4.3.

least n prefixes, applying the definition of $\succeq_{\rm bis}$ on the transition $C[\widetilde{P}] \stackrel{\mu}{\Rightarrow} R$, we infer the existence of C' such that $C''[\widetilde{P}] \stackrel{\widehat{\mu}}{\Rightarrow} C'[\widetilde{P}] \preceq_{\rm bis} R$ and $C''[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Rightarrow} C'[\widetilde{Q}]$. Finally, again applying the definition of $\succeq_{\rm bis}$ on $C[\widetilde{Q}] \succeq_{\rm bis} C''[\widetilde{Q}]$, we derive $C[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Rightarrow}\approx C'[\widetilde{Q}]$. □

Theorem 3.9 (unique solution of contractions for $\approx$)**.** *A system of weakly-guarded contractions having only weakly-guarded sums, has a unique solution for $\approx$.*

Proof. (sketch from [28]) Suppose $\widetilde{P}$ and $\widetilde{Q}$ are two such solutions (for $\approx$) and consider the relation

$$\mathcal{R} \stackrel{\rm def}{=} \{(R,S) \mid R \approx C[\widetilde{P}], S \approx C[\widetilde{Q}] \text{ for some context } C \text{ (having only weakly-guarded sums)}\} \ . \quad (1)$$

We show that $\mathcal{R}$ is a bisimulation. Suppose $R\,\mathcal{R}\,S$ vis the context C, and $R \stackrel{\mu}{\rightarrow} R'$. We have to find S' with $S \stackrel{\widehat{\mu}}{\Rightarrow} S'$ and $R'\,\mathcal{R}\,S'$. From $R \approx C[\widetilde{P}]$, we derive $C[\widetilde{P}] \stackrel{\widehat{\mu}}{\Rightarrow} R'' \approx R'$ for some R''. By Lemma 3.8, there is C' with $R'' \succeq_{\rm bis} C'[\widetilde{P}]$ and $C[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Rightarrow}\approx C'[\widetilde{Q}]$. Hence, by definition of $\approx$, there is also S' with $S \stackrel{\widehat{\mu}}{\Rightarrow} S' \approx C'[\widetilde{Q}]$. This closes the proof, as we have $R' \approx C'[\widetilde{P}]$ and $S' \approx C'[\widetilde{Q}]$. □

4 Rooted contraction

The unique solution theorem of Section 3.3 requires a constrained syntax for sums, due to the congruence and precongruence problems of bisimilarity and contraction with such operator. We show here that the constraints can be removed by moving to the induced congruence and precongruence, the latter called *rooted contraction*:

Definition 4.1. *Two processes P and Q are in* rooted contraction*, written as $P \succeq^{\rm c}_{\rm bis} Q$, if*

1. *$P \stackrel{\mu}{\rightarrow} P'$ implies that there is Q' with $Q \stackrel{\mu}{\rightarrow} Q'$ and $P' \succeq_{\rm bis} Q'$;*
2. *$Q \stackrel{\mu}{\rightarrow} Q'$ implies that there is P' with $P \stackrel{\mu}{\Rightarrow} P'$ and $P' \approx Q'$.*

The precise formulation of this definition was guided by the HOL theorem prover and the following two principles: (1) the definition should not be recursive, along the lines of rooted bisimilarity $\approx^c$ in Def. 2.2; (2) the definition should be built on top of existing *contraction* relation $\succeq_{\rm bis}$ (because of its completeness). A few other candicates were quickly tested and rejected, e.g., because of precongruence issue. The proof of the precongruence result below is along the lines of the analogous result for rooted bisimilarity with respect to bisimilarity.

Theorem 4.2. *$\succeq^{\rm c}_{\rm bis}$ is a precongruence in CCS, and it is the coarsest precongruence contained in $\succeq_{\rm bis}$.*

For a system of rooted contractions, the meaning of "solution for $\succeq^{\rm c}_{\rm bis}$" and of *a unique solution for* $\approx^{\rm c}$ is the expected one — just replace in Definition 3.7 the preorder $\succeq_{\rm bis}$ with $\succeq^{\rm c}_{\rm bis}$, and the equivalence $\approx$ with $\approx^{\rm c}$. For this new relation, the analogous of Lemma 3.8 and of Theorem 3.9 can now be stated without constraints on the sum operator. The schema of the proofs is almost identical, because all properties of $\succeq^{\rm c}_{\rm bis}$ needed in this proof is its precongruence, which is indeed true on unrestricted contexts including direct sums:

Lemma 4.3. *Suppose $\widetilde{P}$ and $\widetilde{Q}$ are solutions for $\succeq^{\rm c}_{\rm bis}$ of a system of weakly-guarded contractions. For any context C, if $C[\widetilde{P}] \stackrel{\mu}{\Rightarrow} R$, then there is a context C' such that $R \succeq_{\rm bis} C'[\widetilde{P}]$ and $C[\widetilde{Q}] \stackrel{\mu}{\Rightarrow}\approx C'[\widetilde{Q}]$.*

Theorem 4.4 (unique solution of contractions for $\approx^{\rm c}$)**.** *A system of weakly-guarded contractions has a unique solution for $\approx^{\rm c}$. (thus also for $\approx$)*

Proof. We first follow the same steps as in the proof of Theorem 3.9 to show the relation $\mathcal{R}$ (now with $\succeq^{c}_{\text{bis}}$ and unrestrict context C) in (1) is bisimulation, exploting Lemma 4.3. Then it remains to show that, for any two process P and Q with action μ, if $P \xrightarrow{\mu} P'$ then there is Q' such that $Q \stackrel{\mu}{\Longrightarrow} Q'$ (not $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$!) and $P' \,\mathcal{R}\, Q'$, and also for the converse direction, exploting Lemma 4.13 of [18] (surprisingly). By definition of *bisimulation* (not $\approx$!) and $\approx^{c}$, we actually proved $P \approx^{c} Q$ instead of $P \approx Q$. □

5 Formalisation

We highlight here a formalisation of CCS in the HOL theorem prover (HOL4) [31], including the new concepts and theorems proposed in the first half of this paper. The whole formalisation (apart from minor fixes and extensions in this paper) is described in [32], and the proof scripts are in HOL4 official examples[2]. The current work consists of about 20,000 lines of proof scripts in Standard ML.

Higher Order Logic (or *HOL Logic*) [2] is a variant of Church's simple theory of types (STT) [8] (or *simple-typed λ-calculus*), plus a higher order version of Hilbert's choice operator ε, Axiom of Infinity, and Rank-1 (prenex) polymorphism. HOL4 has implemented the original HOL Logic, while some other theorem provers in HOL family (e.g. Isabelle/HOL) have certain extensions. Indeed the HOL Logic has considerable simpler logical foundations than most other theorem provers. As a result, formal theories built in HOL4 is relatively easier convincible and can be also easily migrated into other theorem provers, sometimes even automatically [15].

HOL4 is written in Standard ML, a single programming language who plays three different roles:

1. It serves as the underlying implementation language for the core HOL engine;
2. it is used to implement tactics (and tacticals) for writing proofs;
3. it is used as the command language of the HOL interactive shell.

Moreover, using the same language HOL4 users can write complex automatic verification tools by calling HOL's interactive theorem proving facilities. (The formal proof of theorems in CCS theory is mostly done by an *interactive process* closely following their informal proofs.)

In this formalisation we consider only single-variable equations/contractions. This considerably simplifies the required proofs in HOL, enhances the readibility of proof scripts, without loss of generality (for paper proofs, the general case is just a routine adaptation).

5.1 CCS and its transitions by SOS rules

In our CCS formalisation, the type "β `Label`" (`'b` or β is the type variable for actions) accounts for visible actions, divided into input and output actions, defined by HOL's Datatype package:

```
val _ = Datatype `Label = name 'b | coname 'b`;
```

The type "β `Action`" is the union of all visible actions, plus invisible action τ (now based on HOL's `option` theory). The cardinality of "β `Action`" (and therefore of all CCS types built on top of it) depends on the choice (or *type-instantiation*) of β.

The type "(`'a`, β) `CCS`", accounting for the CCS syntax[3], is then defined inductively: (`'a` or α is

[2] `https://github.com/HOL-Theorem-Prover/HOL/tree/master/examples/CCS`

[3] The order of type variables α and β is irrelevant. Our choice is aligned with other CCS literals. $\text{CCS}(h,k)$ is the CCS subcalculus which can use at most h constants and k actions. [12] Thus, to formalize theorems on such a CCS subcalculus, the needed CCS type can be retrieved by instantiating the type variables α and β in "(`'a`, β) `CCS`" with types having the corresponding cardinalities h and k. Monica Nesi goes too far by adding another type variable γ for value-passing CCS [20].

the type variable for recursion variables, "β `Relabeling`" is the type of all relabeling functions, ‘ is for backquotes of HOL terms):

```
val _ = Datatype ‘CCS = nil
                      | var 'a
                      | prefix ('b Action) CCS
                      | sum CCS CCS
                      | par CCS CCS
                      | restr (('b Label) set) CCS
                      | relab CCS ('b Relabeling)
                      | rec 'a CCS‘;
```

We have added some grammar support, using HOL's powerful pretty printer, to represent CCS processes in more readable forms (c.f. the column **HOL (abbrev.)** in Table 1, which summarizes the main syntactic notations of CCS). For the restriction operator, we have chosen to allow a set of names as a parameter, rather than a single name as in the ordinary CCS syntax; this simplifies the manipulation of processes with different orders of nested restrictions.

Operator	CCS Notation	HOL term	HOL (abbrev.)
nil	$\mathbf{0}$	`nil`	`nil`
prefix	$u.P$	`prefix u P`	$u..P$
sum	$P+Q$	`sum P Q`	$P + Q$
parallel	$P \mid Q$	`par P Q`	$P \parallel Q$
restriction	$(\nu\, L)\, P$	`restr L P`	$\nu\ L\ P$
recursion	$\mathtt{rec}\, A.P$	`rec A P`	`rec` $A\ P$
relabeling	$P\,[rf]$	`relab P rf`	`relab` $P\ rf$
constant	A	`var A`	`var` A
invisible action	τ	`tau`	τ
input action	a	`label (name a)`	`In` a
output action	$\overline{a}$	`label (coname a)`	`Out` a

Table 1: Syntax of CCS operators, constant and actions

The transition semantics of CCS processes follows Structural Operational Semantics (SOS) in Fig. 1:

$\vdash u..P \xrightarrow{u} P$ [PREFIX]

$\vdash P \xrightarrow{u} P' \Rightarrow P + Q \xrightarrow{u} P'$ [SUM1]

$\vdash P \xrightarrow{u} P' \Rightarrow Q + P \xrightarrow{u} P'$ [SUM2]

$\vdash P \xrightarrow{u} P' \Rightarrow P \parallel Q \xrightarrow{u} P' \parallel Q$ [PAR1]

$\vdash P \xrightarrow{u} P' \Rightarrow Q \parallel P \xrightarrow{u} Q \parallel P'$ [PAR2]

$\vdash P \xrightarrow{\mathtt{label}\ l} P' \wedge Q \xrightarrow{\mathtt{label}\ (\mathtt{COMPL}\ l)} Q' \Rightarrow P \parallel Q \xrightarrow{\tau} P' \parallel Q'$ [PAR3]

$\vdash P \xrightarrow{u} Q \wedge ((u = \tau) \vee (u = \mathtt{label}\ l) \wedge l \notin L \wedge \mathtt{COMPL}\ l \notin L) \Rightarrow$
$\nu\ L\ P \xrightarrow{u} \nu\ L\ Q$ [RESTR]

$\vdash P \xrightarrow{u} Q \Rightarrow \mathtt{relab}\ P\ rf \xrightarrow{\mathtt{relabel}\ rf\ u} \mathtt{relab}\ Q\ rf$ [RELABELING]

$\vdash \mathtt{CCS\_Subst}\ P\ (\mathtt{rec}\ A\ P)\ A \xrightarrow{u} P' \Rightarrow \mathtt{rec}\ A\ P \xrightarrow{u} P'$ [REC]

The rule `REC` (Recursion) says that if we substitute all appearances of variable A in P to $(\mathbf{rec}\ A\ P)$ and the resulting process has a transition to P' with action u, then $(\mathbf{rec}\ A\ P)$ has the same transition. From HOL's viewpoint, these SOS rules are *inductive definitions* on the 3-ary relation `TRANS` of type "`(`α`, `β`) CCS` $\rightarrow\beta$ `Action` $\rightarrow$`(`α`, `β`) CCS` $\rightarrow$`bool`", generated by HOL's `Hol_reln` function.

A useful function that we have defined, exploiting the interplay between HOL4 and Standard ML (and following an idea by Nesi [19]) is a complex Standard ML function taking a CCS process and returning a theorem indicating all its direct transitions.[4] For instance, we know that the process $(a.0 \mid \bar{a}.0)$ has three possible transitions: $(a.0 \mid \bar{a}.0) \xrightarrow{a} (0 \mid \bar{a}.0)$, $(a.0 \mid \bar{a}.0) \xrightarrow{\bar{a}} (a.0 \mid 0)$ and $(a.0 \mid \bar{a}.0) \xrightarrow{\tau} (0 \mid 0)$. To completely describe all possible transitions of a process, if done manually, the following facts should be proved: (1) there exists transitions from $(a.0 \mid \bar{a}.0)$ (optional); (2) the correctness for each of the transitions; and (3) the non-existence of other transitions.

For large processes it may be surprisingly hard to manually prove the non-existence of transitions. Hence the usefulness of appealing to the new function `CCS_TRANS_CONV`. For instance this function is called on the process $(a.0 \mid \bar{a}.0)$ thus: (‘‘ is for double-backquotes of HOL terms, > is HOL's prompt)

```
> CCS_TRANS_CONV ‘‘par (prefix (label (name "a")) nil)
                       (prefix (label (coname "a")) nil)‘‘
```

This returns the following theorem, indeed describing all immediate transitions of the process:

$\vdash$ `In “a”..nil` $\parallel$ `Out “a”..nil` $-u\rightarrow$ E $\iff$
  ((`u = In “a”`) $\wedge$ (E `= nil` $\parallel$ `Out “a”..nil`) $\vee$
   (`u = Out “a”`) $\wedge$ (E `= In “a”..nil` $\parallel$ `nil`)) $\vee$
  (`u` $= \tau$) $\wedge$ (E `= nil` $\parallel$ `nil`) [Example.ex_A]

5.2 Bisimulation and Bisimilarity

To define (weak) bisimilarity, we first need to define weak transitions of CCS processes. Following the name adopted by Nesi [19], we define a (possibly empty) sequence of τ-transitions between two processes as a new relation called `EPS` ($\stackrel{\epsilon}{\Rightarrow}$), which is the RTC (reflexive transitive closure, *) of the single-step τ-transition:

`EPS` $= (\lambda E\ E'.\ E\ -\tau\rightarrow\ E')^*$ [EPS_def]

Then we can define a weak transition as an ordinary transition wrapped by two ϵ-transitions:

$E\ =u\Rightarrow\ E' \iff \exists E_1\ E_2.\ E \stackrel{\epsilon}{\Rightarrow} E_1 \wedge E_1\ -u\rightarrow\ E_2 \wedge E_2 \stackrel{\epsilon}{\Rightarrow} E'$ [WEAK_TRANS]

For the definition of bisimilarity and the associated coinduction principle [30], we have taken advantage of HOL's coinductive relation package (`Hol_coreln` [1]), a new tool since its Kananaskis-11 release (March 3, 2017).[5] This essentially amounts to defining bisimilarity as the greatest fixed-point of the appropriate functional on relations. Precisely we call the `Hol_coreln` command as follows: (here `WB` is meant to be `WEAK_EQUIV` ($\approx$) in the rest of this paper; `!` and `?` stand for universal and existential quantifiers.)

```
val (WB_rules, WB_coind, WB_cases) = Hol_coreln ‘
    (!(P :('a, 'b) CCS) (Q :('a, 'b) CCS).
       (!l.
         (!P'. TRANS P (label l) P' ==>
               (?Q'. WEAK_TRANS Q (label l) Q' /\ WB P' Q')) /\
         (!Q'. TRANS Q (label l) Q' ==>
               (?P'. WEAK_TRANS P (label l) P' /\ WB P' Q'))) /\
       (!P'. TRANS P tau P' ==> (?Q'. EPS Q Q' /\ WB P' Q')) /\
       (!Q'. TRANS Q tau Q' ==> (?P'. EPS P P' /\ WB P' Q'))
     ==> WB P Q)‘;
```

[4] If the input process could yield something infinite branching, due to the use of recursion or relabeling operators, the program will loop forever without outputing a theorem.

[5] https://hol-theorem-prover.org/kananaskis-11.release.html#new-tools

Hol_coreln returns 3 theorems, of the first being always the same as input term[6] (now proved automatically as a theorem). The second and third theorems, namely WB_coind and WB_cases, express the coinduction proof method for bisimilarity (i.e. any bisimulation is contained in bisimilarity) and the fixed-point property of bisimilarity (bisimilarity itself is a bisimulation, thus the largest bisimulation):

1. $\vdash \forall WB'.$
$(\forall a_0\ a_1.$
$WB'\ a_0\ a_1 \Rightarrow$
$(\forall l.$
$(\forall P'.$
$a_0\ -\texttt{label}\ l\rightarrow\ P' \Rightarrow$
$\exists Q'.\ a_1\ =\texttt{label}\ l\Rightarrow\ Q' \wedge WB'\ P'\ Q') \wedge$
$\forall Q'.$
$a_1\ -\texttt{label}\ l\rightarrow\ Q' \Rightarrow$
$\exists P'.\ a_0\ =\texttt{label}\ l\Rightarrow\ P' \wedge WB'\ P'\ Q') \wedge$
$(\forall P'.\ a_0\ -\tau\rightarrow\ P' \Rightarrow \exists Q'.\ a_1\ \stackrel{\epsilon}{\Rightarrow}\ Q' \wedge WB'\ P'\ Q') \wedge$
$\forall Q'.\ a_1\ -\tau\rightarrow\ Q' \Rightarrow \exists P'.\ a_0\ \stackrel{\epsilon}{\Rightarrow}\ P' \wedge WB'\ P'\ Q') \Rightarrow$
$\forall a_0\ a_1.\ WB'\ a_0\ a_1 \Rightarrow \texttt{WB}\ a_0\ a_1$ [WB_coind, WEAK_EQUIV_coind]

2. $\vdash \forall a_0\ a_1.$
$\texttt{WB}\ a_0\ a_1 \Longleftrightarrow$
$(\forall l.$
$(\forall P'.$
$a_0\ -\texttt{label}\ l\rightarrow\ P' \Rightarrow$
$\exists Q'.\ a_1\ =\texttt{label}\ l\Rightarrow\ Q' \wedge \texttt{WB}\ P'\ Q') \wedge$
$\forall Q'.$
$a_1\ -\texttt{label}\ l\rightarrow\ Q' \Rightarrow$
$\exists P'.\ a_0\ =\texttt{label}\ l\Rightarrow\ P' \wedge \texttt{WB}\ P'\ Q') \wedge$
$(\forall P'.\ a_0\ -\tau\rightarrow\ P' \Rightarrow \exists Q'.\ a_1\ \stackrel{\epsilon}{\Rightarrow}\ Q' \wedge \texttt{WB}\ P'\ Q') \wedge$
$\forall Q'.\ a_1\ -\tau\rightarrow\ Q' \Rightarrow \exists P'.\ a_0\ \stackrel{\epsilon}{\Rightarrow}\ P' \wedge \texttt{WB}\ P'\ Q'$ [WB_cases, WEAK_EQUIV_cases]

The coinduction principle WB_coind says that any bisimulation is contained in the resulting relation (i.e. it is largest), but it didn't constrain the resulting relation in the set of fixed points (e.g. even the universal relation — the set of all pairs — would fit with this theorem); the purpose of WB_cases is to further assert that the resulting relation is indeed a fixed point. Thus WB_coind and WB_cases together make sure that bisimilarity is the greatest fixed point, as the former contributes to "greatest" while the latter contributes to "fixed point". Without HOL's coinductive relation package, bisimilarity would have to be defined by following literally Def. 2.1; then other properties of bisimilarity, such as the fixed-point property in WB_cases, would have to be derived manually (which is quite hard; indeed it was one of the main results in Nesi's formalisation work in HOL88 [19]).

5.3 Context, guardedness and (pre)congruence

We have chosen to use λ-expressions (with the type "$(\alpha,\ \beta)\ \texttt{CCS} \rightarrow (\alpha,\ \beta)\ \texttt{CCS}$") to represent *multi-hole contexts*. This choice has a significant advantage over *one-hole contexts*, as each hole corresponds to one appearence of the *same* variable in single-variable expressions (or equations). Thus *contexts* can

[6]Our mixing of HOL notation and mathematical notation in this paper is not arbitrary. We have to paste here the original proof scripts, which is written in HOL's ASCII term notation (c.f. [1] for more details). HOL4 also suppports writing Unicode symbols directly in proof scripts but we did not make use of them. However, all formal definitions and theorems in the paper are automatically generated from HOL4 in which we have made an effort for generating Unicode and TeX outputs as natural as possible. What is really arbitrary is the presense/absense of outermost universal quantifiers in all generated theorems.

be directly used in formulating the unique solution of equations theorems in single-variable cases. The precise definition is given inductively:

$\texttt{CONTEXT}\ (\lambda t.\ t)$
$\texttt{CONTEXT}\ (\lambda t.\ p)$
$\texttt{CONTEXT}\ e \Rightarrow \texttt{CONTEXT}\ (\lambda t.\ a..e\ t)$
$\texttt{CONTEXT}\ e_1 \wedge \texttt{CONTEXT}\ e_2 \Rightarrow \texttt{CONTEXT}\ (\lambda t.\ e_1\ t + e_2\ t)$
$\texttt{CONTEXT}\ e_1 \wedge \texttt{CONTEXT}\ e_2 \Rightarrow \texttt{CONTEXT}\ (\lambda t.\ e_1\ t \parallel e_2\ t)$
$\texttt{CONTEXT}\ e \Rightarrow \texttt{CONTEXT}\ (\lambda t.\ \nu\ L\ (e\ t))$
$\texttt{CONTEXT}\ e \Rightarrow \texttt{CONTEXT}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf)$ [CONTEXT_rules]

A context is *weakly guarded* (WG) if each hole is underneath a prefix:

$\texttt{WG}\ (\lambda t.\ p)$
$\texttt{CONTEXT}\ e \Rightarrow \texttt{WG}\ (\lambda t.\ a..e\ t)$
$\texttt{WG}\ e_1 \wedge \texttt{WG}\ e_2 \Rightarrow \texttt{WG}\ (\lambda t.\ e_1\ t + e_2\ t)$
$\texttt{WG}\ e_1 \wedge \texttt{WG}\ e_2 \Rightarrow \texttt{WG}\ (\lambda t.\ e_1\ t \parallel e_2\ t)$
$\texttt{WG}\ e \Rightarrow \texttt{WG}\ (\lambda t.\ \nu\ L\ (e\ t))$
$\texttt{WG}\ e \Rightarrow \texttt{WG}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf)$ [WG_rules]

A context is *(strongly) guarded* (SG) if each hole is underneath a *visible* prefix:

$\texttt{SG}\ (\lambda t.\ p)$
$\texttt{CONTEXT}\ e \Rightarrow \texttt{SG}\ (\lambda t.\ \texttt{label}\ l..e\ t)$
$\texttt{SG}\ e \Rightarrow \texttt{SG}\ (\lambda t.\ a..e\ t)$
$\texttt{SG}\ e_1 \wedge \texttt{SG}\ e_2 \Rightarrow \texttt{SG}\ (\lambda t.\ e_1\ t + e_2\ t)$
$\texttt{SG}\ e_1 \wedge \texttt{SG}\ e_2 \Rightarrow \texttt{SG}\ (\lambda t.\ e_1\ t \parallel e_2\ t)$
$\texttt{SG}\ e \Rightarrow \texttt{SG}\ (\lambda t.\ \nu\ L\ (e\ t))$
$\texttt{SG}\ e \Rightarrow \texttt{SG}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf)$ [SG_rules]

A context is *sequential* (SEQ) if each of its *subcontexts* with a hole, apart from the hole itself, is in forms of prefixes or sums: (c.f. Def. 3.3 and p.101,157 of [18] for the informal definitions.)

$\texttt{SEQ}\ (\lambda t.\ t)$
$\texttt{SEQ}\ (\lambda t.\ p)$
$\texttt{SEQ}\ e \Rightarrow \texttt{SEQ}\ (\lambda t.\ a..e\ t)$
$\texttt{SEQ}\ e_1 \wedge \texttt{SEQ}\ e_2 \Rightarrow \texttt{SEQ}\ (\lambda t.\ e_1\ t + e_2\ t)$ [SEQ_rules]

In the same manner, we can also define variants of contexts (GCONTEXT) and weakly guarded contexts (WGS) in which only guarded sums are allowed (i.e. arbitrary sums are forbidden):

$\texttt{GCONTEXT}\ (\lambda t.\ t)$
$\texttt{GCONTEXT}\ (\lambda t.\ p)$
$\texttt{GCONTEXT}\ e \Rightarrow \texttt{GCONTEXT}\ (\lambda t.\ a..e\ t)$
$\texttt{GCONTEXT}\ e_1 \wedge \texttt{GCONTEXT}\ e_2 \Rightarrow \texttt{GCONTEXT}\ (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)$
$\texttt{GCONTEXT}\ e_1 \wedge \texttt{GCONTEXT}\ e_2 \Rightarrow \texttt{GCONTEXT}\ (\lambda t.\ e_1\ t \parallel e_2\ t)$
$\texttt{GCONTEXT}\ e \Rightarrow \texttt{GCONTEXT}\ (\lambda t.\ \nu\ L\ (e\ t))$
$\texttt{GCONTEXT}\ e \Rightarrow \texttt{GCONTEXT}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf)$ [GCONTEXT_rules]

$\texttt{WGS}\ (\lambda t.\ p)$
$\texttt{GCONTEXT}\ e \Rightarrow \texttt{WGS}\ (\lambda t.\ a..e\ t)$
$\texttt{GCONTEXT}\ e_1 \wedge \texttt{GCONTEXT}\ e_2 \Rightarrow \texttt{WGS}\ (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)$
$\texttt{WGS}\ e_1 \wedge \texttt{WGS}\ e_2 \Rightarrow \texttt{WGS}\ (\lambda t.\ e_1\ t \parallel e_2\ t)$
$\texttt{WGS}\ e \Rightarrow \texttt{WGS}\ (\lambda t.\ \nu\ L\ (e\ t))$
$\texttt{WGS}\ e \Rightarrow \texttt{WGS}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf)$ [WGS_rules]

A (pre)congruence is a relation on CCS processes defined on top of CONTEXT. The only difference between congruence and precongruence is that the former must be an equivalence (reflexive, symmetric, transitive), while the latter can be just a preorder (reflexive, transitive):

$$\texttt{congruence}\ R \iff \texttt{equivalence}\ R \wedge \forall x\ y\ ctx.\ \texttt{CONTEXT}\ ctx \Rightarrow R\ x\ y \Rightarrow R\ (ctx\ x)\ (ctx\ y) \quad \texttt{[congruence\_def]}$$

$$\texttt{precongruence}\ R \iff \texttt{PreOrder}\ R \wedge \forall x\ y\ ctx.\ \texttt{CONTEXT}\ ctx \Rightarrow R\ x\ y \Rightarrow R\ (ctx\ x)\ (ctx\ y) \quad \texttt{[precongruence\_def]}$$

For example, we can prove that, strong bisimilarity ($\sim$) and rooted bisimilarity ($\approx^c$) are both congruence by above definition: (the transitivity proof of rooted bisimilarity is actually not easy.)

$$\vdash \texttt{congruence STRONG\_EQUIV} \quad \texttt{[STRONG\_EQUIV\_congruence]}$$

$$\vdash \texttt{congruence OBS\_CONGR} \quad \texttt{[OBS\_CONGR\_congruence]}$$

Although weak bisimilarity ($\approx$) is *not* congruence with respect to CONTEXT, it is indeed "congruence" with respect to GCONTEXT (or if the CCS syntax were defined with only guarded sum operator [27]) as weak bisimilarity ($\approx$) is indeed preserved by weakly-guarded sums.

5.4 Coarsest (pre)congruence contained in $\approx$ ($\succeq_{\text{bis}}$)

As bisimilarity ($\approx$) is not congruence, for this reason rooted bisimilarity has been introduced (Def. 2.2). In this subsection we discuss two proofs of an important result stating that rooted bisimilarity is the coarsest congruence contained in bisimilarity [11, 13, 18] (thus it is the best one):

$$\forall p\ q.\ p \approx^{\text{c}} q \iff (\forall r.\ p + r \approx q + r)\ . \tag{2}$$

Actually the coarsest congruence contained in (weak) bisimilarity, namely the *bisimilarity congruence* [11], can be constructed as the *composition closure* (CC) of (weak) bisimilarity:

$$\texttt{WEAK\_CONGR = CC WEAK\_EQUIV} \quad \texttt{[WEAK\_CONGR]}$$

$$\texttt{CC}\ R = (\lambda g\ h.\ \forall c.\ \texttt{CONTEXT}\ c \Rightarrow R\ (c\ g)\ (c\ h)) \quad \texttt{[CC\_def]}$$

Indeed, for any relation R on CCS processes, the composition closure of R is always finer (i.e. smaller) than R, no matter if R is (pre)congruence or not[7]: (here $\subseteq_r$ stands for *relational subset*)

$$\vdash \forall R.\ \texttt{CC}\ R \subseteq_r R \quad \texttt{[CC\_is\_finer]}$$

Furthermore, we prove that any (pre)congruence contained in R (which itself may not be) is contained in the composition closure of R (hence the closure is the coarsest one):

$$\vdash \forall R\ R'.\ \texttt{congruence}\ R' \wedge R' \subseteq_r R \Rightarrow R' \subseteq_r \texttt{CC}\ R \quad \texttt{[CC\_is\_coarsest]}$$

$$\vdash \forall R\ R'.\ \texttt{precongruence}\ R' \wedge R' \subseteq_r R \Rightarrow R' \subseteq_r \texttt{CC}\ R \quad \texttt{[CC\_is\_coarsest']}$$

Given the central role of the sum operator, we also consider the closure of bisimilarity under such operator, called *equivalence compatible with sums* (SUM_EQUIV):

$$\texttt{SUM\_EQUIV} = (\lambda p\ q.\ \forall r.\ p + r \approx q + r) \quad \texttt{[SUM\_EQUIV]}$$

Rooted bisimilarity $\approx^{\text{c}}$ (a congruence contained in $\approx$), is now contained in WEAK_CONGR, which in turn is trivially contained in SUM_EQUIV, as shown in Fig. 2. Thus, to prove (2), the crux is to prove that SUM_EQUIV implies rooted bisimilarity ($\approx^{\text{c}}$), making all three relations ($\approx^{\text{c}}$, WEAK_CONGR and SUM_EQUIV) equivalent:

$$\forall p\ q.\ (\forall r.\ p + r \approx q + r) \Rightarrow p \approx^{\text{c}} q\ . \tag{3}$$

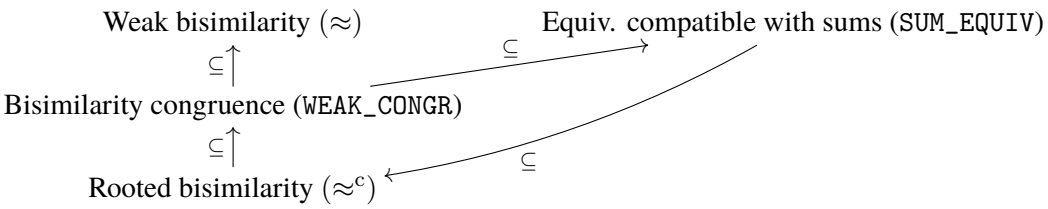

Figure 2: Relationship between the equivalences mentioned

The standard argument [18] requires that p and q do not use up all available labels (i.e. visible actions). Formalising such an argument requires however a detailed treatment on free and bound names of CCS processes (with the restriction operator being a binder), not done yet. However, the proof of (3) can be carried out just assuming that all immediate *weak* derivatives of p and q do not use up all available labels. We have formalised this property and called it the *free action* property:

$\texttt{free\_action}\ p \iff \exists a.\ \forall p'.\ \neg(p\ \texttt{=label}\ a\Rightarrow\ p')$ [free_action_def]

With this property, the actual formalisation of (3) says: [COARSEST_CONGR_RL]

$\vdash\ \texttt{free\_action}\ p \wedge \texttt{free\_action}\ q \Rightarrow (\forall r.\ p + r \approx q + r) \Rightarrow p \approx^c q$

With an almost identical proof, rooted contraction ($\succeq^{\mathrm{c}}_{\mathrm{bis}}$) is also the coarsest precongruence contained in bisimilarity contraction ($\succeq_{\mathrm{bis}}$) (the other direction of (2) is trivial): [COARSEST_PRECONGR_RL]

$\vdash\ \texttt{free\_action}\ p \wedge \texttt{free\_action}\ q \Rightarrow (\forall r.\ p + r \succeq_{bis} q + r) \Rightarrow p \succeq^c_{bis} q$

The formal proofs of above two results precisely follow Milner [18]. If only p (or q) has free actions while the other uses up all available labels, the classic assumption $\mathrm{fn}(p) \cup \mathrm{fn}(q) \neq \mathscr{L}$ (here fn stands for *free names*) does not hold, and the proof cannot be completed. Our assumption is a bit *weaker* in the sense that, p and q do not really need to have the *same* free action (also, a and $\overline{a}$ are *different* actions).

There exists a different, more complex proof of (2), given by van Glabbeek [11], which does not require any additional assumption. The core lemma says, for any two processes p and q, if there exists a *stable* (i.e. τ-free) process k which is not bisimilar with any derivative of p and q, then SUM_EQUIV indeed implies rooted bisimilarity ($\approx^{\mathrm{c}}$):

$$\begin{array}{l}\vdash\ (\exists k.\\ \qquad \texttt{STABLE}\ k \wedge (\forall p'\ u.\ p\ =u\Rightarrow\ p' \Rightarrow \neg(p' \approx k)) \wedge\\ \qquad \forall q'\ u.\ q\ =u\Rightarrow\ q' \Rightarrow \neg(q' \approx k)) \Rightarrow\\ \quad (\forall r.\ p + r \approx q + r) \Rightarrow\\ \quad p \approx^c q\end{array}$$ [PROP3_COMMON]

$\texttt{STABLE}\ p \iff \forall u\ p'.\ p \xrightarrow{u} p' \Rightarrow u \neq \tau$ [STABLE]

To actually get this process k, the proof relies on arbitrary infinite sum of processes and uses transfinite induction to obtain an arbitary large sequence of processes (firstly introduced by Jan Willem Klop [11]) that are all pairwise non-bisimilar. We have partially formalised this proof, because the typed logic implemented in various HOL systems (including Isabelle/HOL) is not strong enough to define a type for all possible ordinal values [21], thus we have replaced transfinite induction with plain induction. As a consequence, the final result is about a restricted class of processes (which we have taken to be the finite-state processes). This proof uses extensively HOL's pred_set theory [17] and has an interesting mix of CCS and pure mathematics in it. (c.f. [32] for more details.)

[7]But if R is equivalence (or preorder), the composition closure of R must be congruence (or precongruence). Also there is no need to put $R\ g\ h$ in the antecedent of CC_def, as this is anyhow obtained from the trivial context $(\lambda x.\ x)$.

5.5 Unique solution of contractions

A delicate point in the formalisation of the results about unique solution of contractions are the proof of Lemma 4.3 and lemmas alike; in particular, there is an induction on the length of weak transitions. For this, rather than introducing a refined form of weak transition relation enriched with its length, we found it more elegant to work with traces (a motivation for this is to set the ground for extensions of this formalisation work to trace equivalence in place of bisimilarity).

A trace is represented by the initial and final processes, plus a list of actions so performed. For this, we first define the concept of label-accumulated reflexive transitive closure (LRTC). Given a labeled transition relation R on CCS, LRTC R is a label-accumulated relation representing the trace of transitions:

```
LRTC R a l b ⟺
∀P.
    (∀x. P x [] x) ∧
    (∀x h y t z. R x h y ∧ P y t z ⇒ P x (h::t) z) ⇒
    P a l b                                                    [LRTC_DEF]
```

The trace relation for CCS can be then obtained by calling LRTC on the (strong, or single-step) labeled transition relation TRANS ($\xrightarrow{\mu}$) defined by SOS rules:

```
TRACE = LRTC TRANS                                             [TRACE_def]
```

If the list of actions is empty, that means that there is no transition and hence, if there is at most one visible action (i.e., a label) in the list of actions, then the trace is also a weak transition. Here we have to distinguish between two cases: no label and unique label (in the list of actions). The definition of "no label" in an action list is easy (here MEM tests if a given element is a member of a list):

```
NO_LABEL L ⟺ ¬∃l. MEM (label l) L                              [NO_LABEL_def]
```

The definition of "unique label" can be done in many ways, the following definition (a suggestion from Robert Beers) avoides any counting or filtering in the list. It says that a label is unique in a list of actions if and only if there is no label in the rest of list:

```
UNIQUE_LABEL u L ⟺
∃L1 L2. (L1 ++ [u] ++ L2 = L) ∧ NO_LABEL L1 ∧ NO_LABEL L2     [UNIQUE_LABEL_def]
```

The final relationship between traces and weak transitions is stated and proved in the following theorem (where the variable $acts$ stands for a list of actions); it says, a weak transition $P \stackrel{u}{\Rightarrow} P'$ is also a trace $P \xrightarrow{acts} P'$ with a non-empty action list $acts$, in which either there is no label (for $u = \tau$), or u is the unique label (for $u \neq \tau$):

```
⊢ P =u⇒ P' ⟺
  ∃acts.
      TRACE P acts P' ∧ ¬NULL acts ∧
      if u = τ then NO_LABEL acts else UNIQUE_LABEL u acts   [WEAK_TRANS_AND_TRACE]
```

Now the formalised version of Lemma 3.8: [UNIQUE_SOLUTION_OF_CONTRACTIONS_LEMMA]

```
⊢ (∃E. WGS E ∧ P ≽bis E P ∧ Q ≽bis E Q) ⇒
  ∀C.
      GCONTEXT C ⇒
      (∀l R.
          C P =label l⇒ R ⇒
          ∃C'.
```

```
            GCONTEXT C' ∧ R ≿bis C' P ∧
            (WEAK_EQUIV ∘r (λx y. x =label l⇒ y)) (C Q)
              (C' Q)) ∧
    ∀R.
        C P =τ⇒ R ⇒
        ∃C'.
            GCONTEXT C' ∧ R ≿bis C' P ∧
            (WEAK_EQUIV ∘r EPS) (C Q) (C' Q)
```

Traces are actually used in the proof of above lemma via the following "unfolding lemma":

```
⊢ GCONTEXT C ∧ WGS E ∧ TRACE ((C ∘ FUNPOW E n) P) xs P' ∧
  LENGTH xs ≤ n ⇒
  ∃C'.
      GCONTEXT C' ∧ (P' = C' P) ∧
      ∀Q. TRACE ((C ∘ FUNPOW E n) Q) xs (C' Q)                [unfolding_lemma4]
```

It roughly says, for any context C and weakly-guarded context E, if $C[E^n[P]] \stackrel{xs}{\Longrightarrow} P'$ and the length of actions $xs \leqslant n$, then P has the form of $C'[P]$ (meaning that P is not touched during the transitions). Traces are used for reasoning about the number of intermediate actions in weak transitions. For instance, from Def. 3.5, it is easy to see that, a weak transition either becomes shorter or remains the same when moving between $\succeq_{\mathrm{bis}}$-related processes. This property is essential in the proof of Lemma 3.8. We show only one such lemma, for the case of τ-transitions passing into $\succeq_{\mathrm{bis}}$ (from left to right):

```
⊢ P ≿bis Q ⇒
  ∀xs P'.
      TRACE P xs P' ∧ NO_LABEL xs ⇒
      ∃xs' Q'.
          TRACE Q xs' Q' ∧ P' ≿bis Q' ∧ LENGTH xs' ≤ LENGTH xs ∧
          NO_LABEL xs'                                  [contracts_AND_TRACE_tau]
```

With all above lemmas, we can thus finally prove Theorem 3.9:

```
⊢ WGS E ⇒ ∀P Q. P ≿bis E P ∧ Q ≿bis E Q ⇒ P ≈ Q
                                          [UNIQUE_SOLUTION_OF_CONTRACTIONS]
```

5.6 Unique solution of rooted contractions

The formal proof of "unique solution of rooted contractions theorem" (Theorem 4.4) has the same initial proof steps as Theorem 3.9; it then requires a few more steps to handle rooted bisimilarity in the conclusion. Overall the two proofs are very similar, mostly because the only property we need from (rooted) contraction is its precongruence. Below is the formally verified version of Theorem 4.4 (having proved the precongruence of rooted contraction, we can use weakly-guarded expressions, without constraints on sums; that is, WG in place of WGS):

```
⊢ WG E ⇒ ∀P Q. P ≿c_bis E P ∧ Q ≿c_bis E Q ⇒ P ≈c Q
                                  [UNIQUE_SOLUTION_OF_ROOTED_CONTRACTIONS]
```

Having removed the constraints on sums, the result is similar to Milner's original 'unique solution of equations' theorem for *strong* bisimilarity ($\sim$) — the same weakly guarded context (WG) is required:

```
⊢ WG E ⇒ ∀P Q. P ∼ E P ∧ Q ∼ E Q ⇒ P ∼ Q          [STRONG_UNIQUE_SOLUTION]
```

In contrast, Milner's "unique solution of equations" theorem for rooted bisimilarity ($\approx^{\mathrm{c}}$) has more severe constraints (both strongly guarded and sequential):

```
⊢ SG E ∧ SEQ E ⇒ ∀P Q. P ≈c E P ∧ Q ≈c E Q ⇒ P ≈c Q
                                                [OBS_UNIQUE_SOLUTION]
```

6 Related formalisation work

Monica Nesi made the first CCS formalisations for both pure and value-passing CCS [19, 20] using early versions of the HOL theorem prover.[8] Her main focus was on implementing decision procedures (as a ML program, e.g. [9]) for automatically proving bisimilarities of CCS processes. Her work is the working basis of ours. However, the differences are substantial, especially in the way of defining bisimilarities. We greatly benefited from features and standard libraries in recent versions of HOL4, and our formalisation has covered a much larger spectrum of the (pure) CCS theory.

Bengtson, Parrow and Weber have made a substantial formalisation work on CCS, π-calculi and ψ-calculi using Isabelle/HOL and its `nominal` datatype package, with main focus on the handling of name binders [5, 6, 22]. Other formalisations in this area include the work of Solange Coupet-Grimal[9] in Coq and Chaudhuri et al. [7] in Abella. The latter focuses on 'bisimulation up-to' techniques for strong bisimilarity for CCS and π-calculus. Damien Pous also formalised up-to techniques and some CCS examples in Coq. [23] Formalisations less related to ours include Kahsai and Miculan [16] for the spi calculus in Isabelle, and Hirschkoff [14] for the π-calculus in Coq.

7 Conclusions and future work

We have highlighted a formalisation of the theory of CCS in the HOL4 theorem prover (for lack of space we have not discussed the formalisation of some basic algebraic theory, of the basic properties of the expansion preorder, and of a few versions of 'bisimulation up to' techniques). The formalisation has focused on the theory of unique solution of equations and contractions. It has also allowed us to further develop the theory, notably the basic properties of rooted contraction, and the unique solution theorem for it with respect to rooted bisimilarity. The formalisation brings up and exploits similarities between results and proofs for different equivalences and preorders. We think that the statements in the formalisation are easy to read and understand, as they are very close to the orginal statements found in standard CCS textbooks [13, 18].

For the future work, it would be worth extending the current work to multi-variable equations/contractions. Preliminary work shows that, a key aspect could be using unguarded constants as free variables (`FV`) and define guardness directly on type CCS (instead of CCS $\rightarrow$ CCS) then connect to previous context:

$\vdash$ `weakly_guarded1` $E \iff$
$\forall X.\ X \in$ `FV` $E \Rightarrow \forall e.$ `CONTEXT` $e \wedge$ (`(var` X`) =` E) $\Rightarrow$ `WG` e

Formalising other equivalences and preorders could also be considered, notably the trace equivalences, as well as more refined process calculi such as value-passing CCS (e.g. exploiting the type variable of actions). On another research line, one could examine the formalisation of a different approach [10] to unique solutions, in which the use of contraction is replaced by semantic conditions on process transitions such as divergence.

Acknowledgements We have benefitted from suggestions and comments from several people from the HOL community, including (in alphabet order) Robert Beers, Jeremy Dawson, Ramana Kumar, Michael Norrish, Konrad Slind, and Thomas Tuerk. The second half of this paper was written in memory of Mike J. Gordon, the creator of HOL theorem prover.

[8] Part of her old proof scripts is now available at `https://github.com/binghe/HOL-CCS/tree/master/CCS-Nesi`.
[9] `https://github.com/coq-contribs/ccs`

References

[1] (2018): *The HOL System DESCRIPTION*. Available at `http://sourceforge.net/projects/hol/files/hol/kananaskis-12/kananaskis-12-description.pdf`.

[2] (2018): *The HOL System LOGIC*. Available at `http://sourceforge.net/projects/hol/files/hol/kananaskis-12/kananaskis-12-logic.pdf`.

[3] S. Arun-Kumar & Matthew Hennessy (1992): *An efficiency preorder for processes*. *Acta Informatica* 29(8), pp. 737–760, doi:10.1007/BF01191894.

[4] J. C. M. Baeten, T. Basten & M. A. Reniers (2010): *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press.

[5] Jesper Bengtson (2010): *Formalising process calculi*. Ph.D. thesis, Acta Universitatis Upsaliensis.

[6] Jesper Bengtson & Joachim Parrow (2007): *A completeness proof for bisimulation in the pi-calculus using isabelle*. *Electronic Notes in Theoretical Computer Science* 192(1), pp. 61–75, doi:10.1.1.117.5660.

[7] Kaustuv Chaudhuri, Matteo Cimini & Dale Miller (2015): *A lightweight formalization of the metatheory of bisimulation-up-to*. In: *Proceedings of the* 2015 *Conference on Certified Programs and Proofs*, ACM, pp. 157–166, doi:10.1145/2676724.2693170.

[8] Alonzo Church (1940): *A formulation of the simple theory of types*. *The journal of symbolic logic* 5(2), pp. 56–68, doi:10.2307/2266170.

[9] Rance Cleaveland, Joachim Parrow & Bernhard Steffen (1993): *The Concurrency Workbench: A semantics-based tool for the verification of concurrent systems*. *ACM Transactions on Programming Languages and Systems* (*TOPLAS*) 15(1), pp. 36–72, doi:10.1145/151646.151648.

[10] Adrien Durier, Daniel Hirschkoff & Davide Sangiorgi (2017): *Divergence and Unique Solution of Equations*. In Roland Meyer & Uwe Nestmann, editors: 28*th International Conference on Concurrency Theory* (*CONCUR* 2017), *Leibniz International Proceedings in Informatics* (*LIPIcs*) 85, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 11:1–11:16, doi:10.4230/LIPIcs.CONCUR.2017.11. Available at `http://drops.dagstuhl.de/opus/volltexte/2017/7784`.

[11] Rob J. van Glabbeek (2005): *A characterisation of weak bisimulation congruence*. In: *Processes, Terms and Cycles*: *Steps on the Road to Infinity*, Springer, pp. 26–39, doi:10.1007/11601548_4.

[12] Roberto Gorrieri (2017): *CCS (25, 12) is Turing-complete*. *Fundamenta Informaticae* 154(1-4), pp. 145–166, doi:10.3233/FI-2017-1557.

[13] Roberto Gorrieri & Cristian Versari (2015): *Introduction to Concurrency Theory*. Transition Systems and CCS, Springer, Cham.

[14] Daniel Hirschkoff (1997): *A full formalisation of π-calculus theory in the calculus of constructions*. In: *International Conference on Theorem Proving in Higher Order Logics*, Springer, pp. 153–169, doi:10.1007/BFb0028392.

[15] Joe Hurd (2011): *The OpenTheory standard theory library*. In: *NASA Formal Methods Symposium*, Springer, pp. 177–191, doi:10.1007/978-3-642-20398-5_14.

[16] Temesghen Kahsai & Marino Miculan (2008): *Implementing spi calculus using nominal techniques*. In: *Conference on Computability in Europe*, Springer, pp. 294–305, doi:10.1007/978-3-540-69407-6_33.

[17] Tom F. Melham (1992): *The HOL pred_sets Library*. *Univ. of Cambridge Computer Lab*, doi:10.1.1.219.5390.

[18] Robin Milner (1989): *Communication and concurrency*. Prentice-Hall.

[19] Monica Nesi (1992): *A formalization of the process algebra CCS in high order logic*. Technical Report UCAM-CL-TR-278, University of Cambridge, Computer Laboratory. Available at `http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-278.pdf`.

[20] Monica Nesi (1999): *Formalising a Value-Passing Calculus in HOL*. *Formal Aspects of Computing* 11(2), pp. 160–199, doi:10.1007/s001650050046.

[21] Michael Norrish & Brian Huffman (2013): *Ordinals in HOL: Transfinite arithmetic up to (and beyond) ω_1*. In: *International Conference on Interactive Theorem Proving*, Springer, pp. 133–146, doi:10.1007/978-3-642-39634-2_12.

[22] Joachim Parrow & Jesper Bengtson (2009): *Formalising the pi-calculus using nominal logic*. *Logical Methods in Computer Science* 5, doi:10.2168/LMCS-5(2:16)2009.

[23] Damien Pous (2007): *New up-to techniques for weak bisimulation*. *Theoretical Computer Science* 380, pp. 164–180, doi:10.1016/j.tcs.2007.02.060.

[24] A. W. Roscoe (1998): *The theory and practice of concurrency*. Prentice Hall. Available at `http://www.cs.ox.ac.uk/people/bill.roscoe/publications/68b.pdf`.

[25] A. W. Roscoe (2010): *Understanding Concurrent Systems*. Springer.

[26] Davide Sangiorgi (2011): *Introduction to Bisimulation and Coinduction*. Cambridge University Press.

[27] Davide Sangiorgi (2015): *Equations, contractions, and unique solutions*. In: *ACM SIGPLAN Notices*, 50, ACM, pp. 421–432, doi:10.1145/2676726.2676965. Available at `https://hal.inria.fr/hal-01089205`.

[28] Davide Sangiorgi (2017): *Equations, contractions, and unique solutions*. *ACM Transactions on Computational Logic* (*TOCL*) 18, p. 4, doi:10.1145/2971339. Available at `https://hal.inria.fr/hal-01647063`.

[29] Davide Sangiorgi & Robin Milner (1992): *The problem of "Weak Bisimulation up to"*. In: *International Conference on Concurrency Theory*, Springer, pp. 32–46, doi:10.1007/BFb0084781.

[30] Davide Sangiorgi & Jan Rutten (2011): *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press.

[31] Konrad Slind & Michael Norrish (2008): *A brief overview of HOL4*. In: *International Conference on Theorem Proving in Higher Order Logics*, Springer, pp. 28–32, doi:10.1007/978-3-540-71067-7_6.

[32] Chun Tian (2017): *A Formalization of Unique Solutions of Equations in Process Algebra*. Master's thesis, AlmaDigital, Bologna. Available at `http://amslaurea.unibo.it/14798/`.