Unique solutions of contractions, CCS, and their HOL formalisation

Unique Solutions of Contractions, CCS, and their HOL Formalisation[☆]

Chun Tian[a,1], Davide Sangiorgi[b]

[a] *University of Trento, Italy and Fondazione Bruno Kessler, Italy*
[b] *University of Bologna, Italy and INRIA, France*

Abstract

The unique solution of contractions is a proof technique for (weak) bisimilarity that overcomes certain syntactic limitations of Milner's "unique solution of equations" theorem. This paper presents an overview of a comprehensive formalisation of Milner's Calculus of Communicating Systems (CCS) in the HOL theorem prover (HOL4), with a focus towards the theory of unique solutions of equations and contractions. The formalisation consists of about 24,000 lines (1MB) of code in total. Some refinements of the "unique solution of contractions" theory itself are obtained. In particular we remove the constraints on summation, which must be guarded, by moving from contraction to *rooted contraction*. We prove the "unique solution of rooted contractions" theorem and show that rooted contraction is the coarsest precongruence contained in the contraction preorder.

Keywords: process calculi, theorem proving, coinduction, unique solution of equations, congruence

1. Introduction

A prominent proof method for bisimulation, put forward by Robin Milner and widely used in his landmark CCS book [2], 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 [3, 4, 5].

Milner's unique-solution theorem for weak bisimilarity, however, has severe syntactic limitations: the equations must be both *guarded* and *sequential*. That is, the variables of the equations can only occur underneath visible prefixes and summation. One way to overcome these limitations is to replace the equations with special inequations called *contractions* [6, 7]. The contraction relation is a preorder that, roughly, places some efficiency constraints on processes. The unique solution of contractions is defined as with equations: any two solutions must be componentwise bisimilar. The difference from 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 are eliminated. One constraint that still remains in [7] (where the issue is bypassed by using a more restrictive CCS syntax) is the occurrences of arbitrary sums (e.g. $P+Q$) due to the non-substitutivity of the contraction preorder in this case.

This paper presents a comprehensive formalisation of Milner's Calculus of Communicating Systems (CCS) in the HOL theorem prover (HOL4), with a focus towards the theory of unique solutions of equations and contractions. Many results in Milner's CCS book [2] are covered, since the unique-solution theorems rely on a large number of fundamental results. Indeed the formalisation encompasses all basic properties of strong, weak and rooted bisimilarities (e.g. the fixed-point and substitutivity properties), and their algebraic laws. Further extensions include several versions of "bisimulation up to" techniques, and properties of the

[☆] This is an extended and refined version of the paper with the same title published in EXPRESS/SOS 2018 [1].

Email addresses: chun.tian@unitn.it (Chun Tian), davide.sangiorgi@unibo.it (Davide Sangiorgi)

[1] Part of this work was carried out when the first author was studying at the University of Bologna, Italy.

expansion and contraction preorders. Concerning rooted bisimilarity, the formalisation includes Hennessy's Lemma, Deng's Lemma, and two theorems saying that rooted bisimilarity is the coarsest (largest) congruence contained in weak bisimilarity ($\approx$): one is classical with the hypothesis that no process uses up all labels; the other one is without such hypothesis, essentially formalising van Glabbeek's proof [8]. With this respect, the work is also an extensive experiment using the HOL theorem prover with its recent developments, including its coinduction package.

This formalisation has offered us the possibility of further refining the theory of unique solutions of contractions. In particular, existing results [7] have 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. In this paper, the contraction-based proof technique is further refined by moving from the contraction preorder to *rooted contraction*, which is shown to be 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 summation.

Another benefit of the formalisation is that we can take advantage of results about different equivalences and preorders that share similar proof structures. Such structural simililarities can be found, for instance, in the following cases: the proofs 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 proofs about unique solution(s) of equations for weak bisimilarity that use the contraction preorder as an auxiliary relation, and other unique-solution theorems (e.g. the one for rooted bisimilarity in which the auxiliary relation is rooted contraction); the proofs about various forms of enhancements of the bisimulation proof method (the "up-to" techniques). In these cases, when moving between proofs there are only a few places in the HOL proof scripts that have to be modified. Then the successful termination of a proof gives us the guarantee that the proof is correct, eliminating the risks of overlooking or missing details as in *paper-and-pencil* proofs.

Concerning the formalisation of unique-solution theorems, we first consider the case of a single equation (or contraction), the *univariable case*. Then we turn to the *multivariable case* where more equations (or contractions) with multiple equation variables are involved. The univariable versions of the unique-solution theorems can directly use λ-functions to represent CCS equations, while the multivariable versions require more careful and delicate treatments of CCS expressions, with multiple variables and substitutions acting on them. In contrast to literature such as [9], we have followed Milner's original approach [10] and adopted the same type for both CCS equations and processes: those undefined constants in CCS terms are treated as (free) equation variables, and a CCS process is a CCS expression without equation variables. This allows us a smoother move from the univariable case to the multivariable one. (See Section 5 for more details.)

Structure of the paper. In Section 2 we recall the core theory of CCS, including its syntax, operational semantics, bisimilarity and rooted bisimilarity. Then, Section 3 discusses equations and contractions, and in particular, Section 3.4 presents rooted contraction and the related unique-solution result for rooted bisimilarity. In Section 4 we highlight the CCS formalisation in HOL4, with the unique-solution theorems in the univariable case. Section 5 describes the extension to the multivariable case. Finally, in Section 6 and 7 we discuss the related work, conclusions, and a few directions for future work.

2. Calculus of Communicating Systems (CCS)

We assume a possibly infinite set of *names* $\mathscr{L} = \{a, b, \ldots\}$ yielding *input* and *output labels* (or *actions*), a special *invisible* action $\tau \notin \mathscr{L}$, and another possibly infinite set of *agent variables* $\mathscr{X} = \{X, Y, \ldots\}$. The class of CCS terms is then inductively defined from $\mathbf{0}$ (the terminated process) and agent variables by the operators of *prefixing* (.), *parallel composition* (|), *summation* (or *binary choice*, +), *restriction* (ν), *relabeling* ([·]) and *recursion* (`rec`):

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

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

$$\frac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu} L)\,P \xrightarrow{\mu} (\boldsymbol{\nu} L)\,P'}\ \mu, \overline{\mu} \notin L \qquad \frac{P\{\mathtt{rec}\,X.\,P/X\} \xrightarrow{\mu} P'}{\mathtt{rec}\,X.\,P \xrightarrow{\mu} P'} \qquad \frac{P \xrightarrow{\mu} P'}{P\,[rf] \xrightarrow{rf(\mu)} P'\,[rf]}$$

Figure 1: Structural Operational Semantics of CCS. (The symmetric rules for + and | are omitted.)

In our presentation of CCS, the restriction operator takes a set of labels $L \subseteq \mathscr{L}$ rather than a single one. The relabeling operator takes a relabeling function $rf : \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\} \to \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$, that can handle multiple actions including τ. A valid relabeling function rf must however satisfy $rf(\tau) = \tau$ and $\forall l \in \mathscr{L} \cup \overline{\mathscr{L}}.\ rf(\bar{l}) = \overline{rf(l)}$ (with $\bar{\bar{l}} = l$ for all $l \in \mathscr{L}$). We sometimes omit a trailing $\mathbf{0}$, e.g., writing $a \mid b$ for $a.\,\mathbf{0} \mid b.\,\mathbf{0}$. A CCS process P may evolve to another one (i.e. having a *transition*), say P', under an action μ, written by $P \xrightarrow{\mu} P'$. The transition semantics of CCS processes is given by means of a Labeled Transition System (LTS) expressed in Structural Operational Semantics (SOS) rules shown in Fig. 1. A CCS process has or uses *guarded sums* if all occurrences of summation are in the form $a.\,P + b.\,Q$. The *immediate derivatives* of a process P are the elements of $\{P' \mid P \xrightarrow{\mu} P' \text{ for some } \mu\}$. We use ℓ to range over visible actions (i.e. inputs or outputs, excluding τ) and μ to range over all actions.

Some standard notations for transitions: $\stackrel{\epsilon}{\Longrightarrow}$ is the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\stackrel{\mu}{\Longrightarrow}$ is $\stackrel{\epsilon}{\Longrightarrow}\xrightarrow{\mu}\stackrel{\epsilon}{\Longrightarrow}$ (the composition of the three relations). Moreover, $P \xrightarrow{\widehat{\mu}} P'$ holds if $P \xrightarrow{\mu} P'$ or $\mu = \tau \wedge P = P'$; similarly $P \stackrel{\widehat{\mu}}{\Longrightarrow} P'$ holds if ($P \stackrel{\mu}{\Longrightarrow} P'$ or $\mu = \tau \wedge P = P'$). 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. We let $\mathcal{R}$, $\mathcal{S}$ range over binary relations, sometimes using infix notation for them, e.g. $P\,\mathcal{R}\,Q$ means $(P, Q) \in \mathcal{R}$. We use a tilde, as in $\widetilde{P}$, to denote (finite) tuples of elements. All relation notations can be extended to tuples componentwise, e.g., $\widetilde{P}\,\mathcal{R}\,\widetilde{Q}$ means $P_i\,\mathcal{R}\,Q_i$ for each index i of the tuples $\widetilde{P}$ and $\widetilde{Q}$. We use $\stackrel{\text{def}}{=}$ 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 here are mainly *weak* ones, in that they abstract from the number of internal steps being performed:

Definition 2.1. *A process relation $\mathcal{R}$ is a (weak)* bisimulation *if, whenever $P\,\mathcal{R}\,Q$,*

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 (weakly) bisimilar*, written as $P \approx Q$, if $P\,\mathcal{R}\,Q$ for some bisimulation $\mathcal{R}$.*

Strong bisimulation and strong bisimilarity ($\sim$) can be obtained by replacing the weak transition $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ in clause (1) with the transition $Q \xrightarrow{\mu} Q'$ (and similarly for the other clause). Weak bisimilarity is not preserved by the sum operator (except for guarded sums), i.e. $P_1 \approx Q_1$ and $P_2 \approx Q_2$ do not imply $P_1 + P_2 \approx Q_1 + Q_2$. For this reason, Milner introduced the concept of *observational congruence*, also called *rooted bisimilarity* [9, 11]:

Definition 2.2. *Two processes P and Q are* rooted bisimilar*, written as $P \approx^{\text{c}} Q$, if*

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* $P \stackrel{\mu}{\Longrightarrow} P'$ *and* $P' \approx Q'$.

Relation $\approx^{\mathrm{c}}$ is indeed preserved by the sum operator. The above definition also brings up a proof technique for proving rooted bisimilarity from a given bisimulation. The next lemma plays an important role in proving some key results in this paper:

Lemma 2.3 (rooted bisimilarity by bisimulation)**.** *Given a (weak) bisimulation* $\mathcal{R}$*, suppose two processes* P *and* Q *satisfy the following properties:*

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

Then P *and* Q *are rooted bisimilar, i.e.* $P \approx^{\mathrm{c}} Q$*.*

One basic property of rooted bisimiarity is congruence, indeed it is the coarsest congruence contained in bisimilarity (see Section 4.8 and Section 4.7 for more details):

Theorem 2.4. $\approx^{\mathrm{c}}$ *is a congruence in CCS, and it is the coarsest congruence contained in* $\approx$*.*

3. Equations and contractions

In the CCS syntax, a recursion $\mathtt{rec}\, A.\, P$ acts as a binder for A in the body P. This gives rise, in the expected manner, to the notions of *free* and *bound* recursion variables in a CCS expression. For instance, X is free in $a.\, X + \mathtt{rec}\, Y.\, (b.\, Y)$ while Y is bound; whereas X is both free and bound in $a.\, X + \mathtt{rec}\, X.\, (b.\, X)$. A term without free variables is a *process*.

In this paper (and the formalisation work), we use the agent variables also as *equation variables*. This eliminates the need of an additional type for CCS equations, and we can reuse the existing variable substitution operation (cf. the SOS rule for the Recursion in Fig. 1) for substitutions of equation variables. For example, the result of substituting variable X with $\mathbf{0}$ in $a.\, X + \mathtt{rec}\, X.\, (b.\, X)$, written as $(a.\, X + \mathtt{rec}\, X.\, (b.\, X))\{\mathbf{0}/X\}$, is $a.\, \mathbf{0} + \mathtt{rec}\, X.\, (b.\, X)$ with $\mathtt{rec}\, X.\, (b.\, X)$ untouched. Multivariable substitutions are written in the same syntax, e.g. $E\{\widetilde{P}/\widetilde{X}\}$. Whenever $\widetilde{X}$ is clear from the context, we may also write $E[\widetilde{P}]$ instead of $E\{\widetilde{P}/\widetilde{X}\}$ (and $E[P]$ for $E\{P/X\}$ if there is a single equation variable X).

3.1. Systems of equations

When discussing equations it is standard to talk about "contexts". This is a CCS expression possibly containing free variables that, however, may not occur within the body of recursive definitions. Milner's "unique solution of equations" theorems [2] intuitively say 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.

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

The components of $\widetilde{P}$ need not be different from each other, as it must be for the variables $\widetilde{X}$.

Definition 3.2 (solutions and the unique solution)**.** *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* $P_i \approx E_i[\widetilde{P}]$*;*
- *The system has* a unique solution for $\approx$ *if whenever* $\widetilde{P}$ *and* $\widetilde{Q}$ *are both solutions then* $\widetilde{P} \approx \widetilde{Q}$*.*

Similarily, the *(unique) solution of a system of equations for* $\sim$ (or for $\approx^{\mathrm{c}}$) can be obtained by replacing all occurrences of $\approx$ in the above definition with $\sim$ and $\approx^{\mathrm{c}}$, respectively.

For instance, the solution of the equation $X \approx a.\, X$ is the process $R \stackrel{\mathrm{def}}{=} \mathtt{rec}\, A.\, (a.\, A)$, and for any other solution P we have $P \approx R$. In contrast, the equation $X \approx a \mid X$ has solutions that may be quite different, for instance, K and $K \mid b$, for $K \stackrel{\mathrm{def}}{=} \mathtt{rec}\, K.\, (a.\, K)$. (Actually any process capable of continuously performing a–actions is a solution of $X \approx a \mid X$.) Examples of systems that do not have unique solutions are: $X = X$, $X = \tau.\, X$ and $X = a \mid X$.

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 each* X_i *is underneath a prefix;*
- guarded *if, in each* E_i*, each occurrence of each* X_i *is underneath a* visible *prefix;*
- sequential *if, in each* E_i*, each occurrence of each* X_i *is only underneath prefixes and sums.*

In other words, if a system of equations is sequential, then for each E_i, any subexpression of E_i in which X_j appears, apart from X_j itself, is a sum of prefixed expressions. For instance,

- $X = \tau.X + \mu.\mathbf{0}$ is sequential but not guarded, because the guarding prefix for the variable is not visible;
- $X = \ell.X \mid P$ is guarded but not sequential;
- $X = \ell.X + \tau.\boldsymbol{\nu}a\,(a.\overline{b} \mid a.\mathbf{0})$, as well as $X = \tau.(a.X + \tau.b.X + \tau)$ are both guarded and sequential.

Milner has three versions of "unique solution of equations" theorems, for $\sim$, $\approx$ and $\approx^{\mathrm{c}}$, respectively, though only the following two versions are explicitly mentioned in [2, p. 103, 158]:

Theorem 3.4 (unique solution of equations for $\sim$)**.** *Let* E_i *be weakly guarded with free variables in* $\widetilde{X}$*, and let* $\widetilde{P} \sim \widetilde{E}\{\widetilde{P}/\widetilde{X}\}$*,* $\widetilde{Q} \sim \widetilde{E}\{\widetilde{Q}/\widetilde{X}\}$*. Then* $\widetilde{P} \sim \widetilde{Q}$*.*

Theorem 3.5 (unique solution of equations for $\approx^{\mathrm{c}}$)**.** *Let* E_i *be guarded and sequential with free variables in* $\widetilde{X}$*, and let* $\widetilde{P} \approx^{\mathrm{c}} \widetilde{E}\{\widetilde{P}/\widetilde{X}\}$*,* $\widetilde{Q} \approx^{\mathrm{c}} \widetilde{E}\{\widetilde{Q}/\widetilde{X}\}$*. Then* $\widetilde{P} \approx^{\mathrm{c}} \widetilde{Q}$*.*

The version of Milner's unique-solution theorem for $\approx$ further requires that all sums are guarded:

Theorem 3.6 (unique solution of equations for $\approx$)**.** *Let* E_i *(with only guarded sums) be guarded and sequential, and with free variables in* $\widetilde{X}$*. Let* $\widetilde{P} \approx \widetilde{E}\{\widetilde{P}/\widetilde{X}\}$*,* $\widetilde{Q} \approx \widetilde{E}\{\widetilde{Q}/\widetilde{X}\}$*, then* $\widetilde{P} \approx \widetilde{Q}$*.*

The proof of the last two theorems above exploits an invariance property on immediate derivatives of guarded and sequential expressions, and then extracts a bisimulation (up to bisimilarity) out of the solutions of the system. To see the need of the sequentiality condition, consider the equation (from [2]) $X \approx \boldsymbol{\nu}a\,(a.X \mid \overline{a})$ where X is guarded but not sequential. Any process that does not perform a–action is a solution, e.g. $\mathbf{0}$ and $b.\mathbf{0}$.

For more details on the formalisation of the above three theorems, see Section 4.10 for the univariable case and Section 5 for the multivariable case.

3.2. Expansions and Contractions

Milner's "unique solution of equations" theorem for $\approx$ (Theorem 3.6) brings a new proof technique for proving (weak) bisimilarities. However, it has limitations: the equations must be guarded and sequential. (Moreover, all sums where equation variables appear must be guarded sums.) This limits the usefulness of the technique, since the occurrences of other operators using equation variables, such as parallel composition and restriction, in general cannot be eliminated. The constraints in Theorem 3.6, however, can be weakened if we move from equations to a special kind of 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 are performed). The process Q, however, may be nondeterministic and may have other ways to do the same work, ways which could be slower (i.e., involving more τ-actions than those performed by P).

Definition 3.7 (contraction)**.** *A process relation* $\mathcal{R}$ *is a* (bisimulation) contraction *if, whenever* $P\,\mathcal{R}\,Q$*,*

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

Two processes P and Q are in the bisimilarity contraction*, written as $P \succeq_{\mathrm{bis}} Q$, if $P \mathcal{R} Q$ for some contraction $\mathcal{R}$. Sometimes we write $\preceq_{\mathrm{bis}}$ for the inverse of $\succeq_{\mathrm{bis}}$.*

In clause (1) of the above definition, Q is required to match the challenge transition of P 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, clause (2) entirely ignores efficiency on the challenges from Q: the final derivatives are required to be related by $\approx$, rather than by $\mathcal{R}$.

Bisimilarity contraction is coarser than bisimilarity expansion $\succeq_{\mathrm{e}}$ [12, 6], one of the most useful auxiliary relations in up-to techniques:

Definition 3.8 (expansion)**.** *A process relation $\mathcal{R}$ is an* expansion *if, whenever $P \mathcal{R} Q$,*

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

Two processes P and Q are in the bisimilarity expansion*, written as $P \succeq_{\mathrm{e}} Q$, if $P \mathcal{R} Q$ for some expansion $\mathcal{R}$.*

Bisimilarity expansion is widely used in proof techniques for bisimilarity. It intuitively refines bisimilarity by formalising the idea of "efficiency" between processes. Clause (1) is the same as for contraction, while in clause (2) expansion requires $P \stackrel{\mu}{\Longrightarrow} P'$, rather than $P \stackrel{\widehat{\mu}}{\Longrightarrow} P'$. Moreover, in clause (2) of Def. 3.7 the final derivatives are simply required to be bisimilar ($P' \approx Q'$). Intuitively, $P \succeq_{\mathrm{e}} Q$ holds if $P \approx Q$ and, in addition, *Q is always at least as efficient as P*.

Example 3.9. *We have $a \not\succeq_{\mathrm{bis}} \tau.a$. However, $a + \tau.a \succeq_{\mathrm{bis}} a$, as well as its converse, $a \succeq_{\mathrm{bis}} a + \tau.a$. Indeed, if $P \approx Q$ then $P \succeq_{\mathrm{bis}} P + Q$. The last two relations do not hold with $\succeq_{\mathrm{e}}$, which explains the strictness of the inclusion $\succeq_{\mathrm{e}} \subset \succeq_{\mathrm{bis}}$.*

Bisimilarity expansion and bisimilarity contraction are both preorders. Similarily with (weak) bisimilarity, both the expansion and the contraction preorders are preserved by all CCS operators except the summation. The proofs are similar to those for bisimilarity, see, e.g. [7] for details.

3.3. Systems of contractions

A *system of contractions* is defined as a system of equations, except that the contraction symbol $\succeq_{\mathrm{bis}}$ is used in the place of $=$ in Def. 3.1. Thus a system of contractions is a set $\{X_i \succeq_{\mathrm{bis}} E_i\}_{i \in I}$ where I is an indexing set and each E_i contains variables in $\widetilde{X}$.

Now we recall the "unique solution of contractions" theorem [7], which weakens the requirements of Milner's result (Theorem 3.6).

Lemma 3.10. *Suppose $\widetilde{P}$ and $\widetilde{Q}$ are solutions (for $\succeq_{\mathrm{bis}}$) of a system of weakly guarded contractions that uses guarded sums. For any context C that uses guarded sums, if $C[\widetilde{P}] \stackrel{\mu}{\Longrightarrow} R$, then there is a context C' that uses guarded sums such that $R \succeq_{\mathrm{bis}} C'[\widetilde{P}]$ and $C[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Longrightarrow} R' \approx C'[\widetilde{Q}]$ for some R'.*

Proof. (sketch from [7]) Let n be the length (i.e., the number of one-step transitions) of a transition $C[\widetilde{P}] \stackrel{\mu}{\Longrightarrow} R$, and let $C''[\widetilde{P}]$ and $C''[\widetilde{Q}]$ be the processes obtained from $C[\widetilde{P}]$ and $C[\widetilde{Q}]$ by unfolding the definition of $\succeq_{\mathrm{bis}}$ for n times. Thus in C'' each hole is underneath at least n prefixes, and therefore cannot contribute to an action in the first n transitions. Moreover, all involved contexts use guarded sums.

We have $C[\widetilde{P}] \succeq_{\mathrm{bis}} C''[\widetilde{P}]$ and $C[\widetilde{Q}] \succeq_{\mathrm{bis}} C''[\widetilde{Q}]$ from the precongruence property of $\succeq_{\mathrm{bis}}$ (we exploit here the syntactic constraints on sums). Moreover, since each hole of the context C'' is underneath at least n prefixes, applying the definition of $\succeq_{\mathrm{bis}}$ on the transition $C[\widetilde{P}] \stackrel{\mu}{\Longrightarrow} R$, we infer the existence of C' such that $C''[\widetilde{P}] \stackrel{\widehat{\mu}}{\Longrightarrow} C'[\widetilde{P}] \preceq_{\mathrm{bis}} R$ and $C''[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Longrightarrow} C'[\widetilde{Q}]$. Finally, again applying the definition of $\succeq_{\mathrm{bis}}$ on $C[\widetilde{Q}] \succeq_{\mathrm{bis}} C''[\widetilde{Q}]$, we derive $C[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Longrightarrow} R' \approx C'[\widetilde{Q}]$ for some R'. □

Theorem 3.11 (unique solution of contractions [7])**.** *Let E_i (and with guarded sums only) be weakly guarded, with free variables in $\widetilde{X}$, and let $\widetilde{P} \succeq_{\mathrm{bis}} \widetilde{E}\{\widetilde{P}/\widetilde{X}\}$, $\widetilde{Q} \succeq_{\mathrm{bis}} \widetilde{E}\{\widetilde{Q}/\widetilde{X}\}$. Then $\widetilde{P} \approx \widetilde{Q}$.*

Proof. We prove $P_i \approx Q_i$ (for each $i \in I$) by considering the following relation

$$\mathcal{R} \stackrel{\mathrm{def}}{=} \{(R, S) \mid R \approx C[\widetilde{P}], S \approx C[\widetilde{Q}] \text{ for some context } C \text{ (with only guarded sums)}\} \ .$$

Obviously we have $(P_i, Q_i) \in \mathcal{R}$ (by taking $C = E_i$, and the fact that $\succeq_{\mathrm{bis}}$ implies $\approx$). It remains to show that $\mathcal{R}$ is a bisimulation. Suppose $(R, S) \in \mathcal{R}$ via a context C. For any R' such that $R \xrightarrow{\mu} R'$, we have to find an S' with $S \stackrel{\widehat{\mu}}{\Longrightarrow} S'$ and $(R', S') \in \mathcal{R}$. From $R \approx C[\widetilde{P}]$, we derive $C[\widetilde{P}] \stackrel{\widehat{\mu}}{\Longrightarrow} R'' \approx R'$ for some R''. Then by Lemma 3.10, there exists C' with $R'' \succeq_{\mathrm{bis}} C'[\widetilde{P}]$ and $C[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Longrightarrow} S'' \approx C'[\widetilde{Q}]$ for some S''. From $S \approx C[\widetilde{Q}]$ and $C[\widetilde{Q}] \stackrel{\widehat{\mu}}{\Longrightarrow} S''$, by induction and definition of $\approx$, we find S' with $S \stackrel{\widehat{\mu}}{\Longrightarrow} S'$. This completes the proof, as we have $R' \approx C'[\widetilde{P}]$ and $S' \approx C'[\widetilde{Q}]$ by transitivity of $\approx$ and the fact that $\succeq_{\mathrm{bis}}$ implies $\approx$. (The other side from S follows in the same manner.) See Fig. 2 for a visual illustration. □

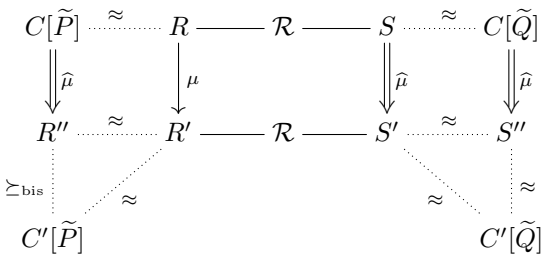

Figure 2: Proof illustration of Theorem 3.11 (showing $\mathcal{R}$ is a bisimulation)

3.4. Rooted contraction

Theorem 3.11 brings a new proof technique for bisimilarity, which is less restrictive than Milner's Theorem 3.6 (but with the additional costs of checking $\succeq_{\mathrm{bis}}$ in addition to $\approx$). However, compared with Milner's Theorem 3.4, there remains the limitation on the need of guarded sums. This is mainly due to the fact that $\approx$ is not a congruence and also $\succeq_{\mathrm{bis}}$ is not a precongruence. Inspired by rooted bisimilarity, to eliminate the restriction on guarded sums we refine the idea of contractions by moving to *rooted contractions*:

Definition 3.12. *Two processes P and Q are in* rooted contraction*, written as $P \succeq^{\mathrm{c}}_{\mathrm{bis}} Q$, if*

1. *$P \xrightarrow{\mu} P'$ implies that there is Q' with $Q \xrightarrow{\mu} Q'$ and $P' \succeq_{\mathrm{bis}} Q'$;*
2. *$Q \xrightarrow{\mu} Q'$ implies that there is P' with $P \stackrel{\mu}{\Longrightarrow} P'$ and $P' \approx Q'$.*

The above definition was found with the help of interactive theorem proving. The following two principles were adopted when manually searching for a possible definition: (1) the definition should not be coinductive, along the lines of rooted bisimilarity $\approx^{\mathrm{c}}$ (Def. 2.2); (2) the definition should be built on top of the existing *contraction* relation $\succeq_{\mathrm{bis}}$. Furthermore, we needed to prove that the definition being found indeed yields the coarsest precongruence contained in $\succeq_{\mathrm{bis}}$. The proof is similar with the analogous result for $\approx^{\mathrm{c}}$. See Section 4.8 for more details.

Theorem 3.13. *$\succeq^{\mathrm{c}}_{\mathrm{bis}}$ is a precongruence in CCS, and it is the coarsest precongruence contained in $\succeq_{\mathrm{bis}}$.*

For this new relation, the analogous of Lemma 3.10 and of Theorem 3.11 can now be stated without constraints on summation. The schema of the proofs is almost identical, because all properties of $\succeq^{\mathrm{c}}_{\mathrm{bis}}$ needed in this proof is its precongruence, which is indeed true for all weakly guarded contexts:

Lemma 3.14. *Let $\widetilde{P}$ and $\widetilde{Q}$ be solutions (for $\succeq^{\mathrm{c}}_{\mathrm{bis}}$) of a system of weakly guarded contractions. For any context C, if $C[\widetilde{P}] \stackrel{\mu}{\Longrightarrow} R$, then there is a context C' such that $R \succeq_{\mathrm{bis}} C'[\widetilde{P}]$ and $C[\widetilde{Q}] \stackrel{\mu}{\Longrightarrow} R' \approx C'[\widetilde{Q}]$ for some R'.*

The next lemma is actually Lemma 3.13 of [2, p. 102], and is needed to prove Milner's "unique solution of equations for $\sim$" (Theorem 3.4):

Lemma 3.15. *If the variables $\widetilde{X}$ are weakly guarded in E, and $E\{\widetilde{P}/\widetilde{X}\} \xrightarrow{\alpha} P'$, then P' takes the form $E'\{\widetilde{P}/\widetilde{X}\}$ (for some expression E'), and moreover, for any $\widetilde{Q}$, $E\{\widetilde{Q}/\widetilde{X}\} \xrightarrow{\alpha} E'\{\widetilde{Q}/\widetilde{X}\}$.*

Theorem 3.16 (unique solution of rooted contractions)**.** *Let E_i be weakly guarded with free variables in $\widetilde{X}$, and let $\widetilde{P} \succeq^{\mathrm{c}}_{\mathrm{bis}} \widetilde{E}\{\widetilde{P}/\widetilde{X}\}$, $\widetilde{Q} \succeq^{\mathrm{c}}_{\mathrm{bis}} \widetilde{E}\{\widetilde{Q}/\widetilde{X}\}$. Then $\widetilde{P} \approx^{\mathrm{c}} \widetilde{Q}$.*

Proof. We prove $P_i \approx^{\mathrm{c}} Q_i$ (for each $i \in I$) by considering the following relation

$$\mathcal{R} \stackrel{\mathrm{def}}{=} \{(R, S) \mid R \approx C[\widetilde{P}], S \approx C[\widetilde{Q}] \text{ for some context } C\} \ .$$

Following the same steps in the proof of Theorem 3.11 (using Lemma 3.14 in place of Lemma 3.10), we can prove that $(P_i, Q_i) \in \mathcal{R}$ and $\mathcal{R}$ is indeed a bisimulation. But this only shows $P_i \approx Q_i$. To further show $P_i \approx^{\mathrm{c}} Q_i$, we appeal to Lemma 2.3: for any P' such that $P_i \xrightarrow{\mu} P'$, we have to find a Q' with $Q_i \stackrel{\mu}{\Longrightarrow} Q'$ and $(R', S') \in \mathcal{R}$. From $P_i \succeq^{\mathrm{c}}_{\mathrm{bis}} E_i[\widetilde{P}]$, by definition of $\succeq^{\mathrm{c}}_{\mathrm{bis}}$ we derive $E_i[\widetilde{P}] \xrightarrow{\mu} P''$ for some P''. Then by Lemma 3.15 there exists a context E' with $P'' = E'[\widetilde{P}]$ and $E_i[\widetilde{Q}] \xrightarrow{\mu} E'[\widetilde{Q}]$. From $Q_i \succeq^{\mathrm{c}}_{\mathrm{bis}} E_i[\widetilde{Q}]$ and $E_i[\widetilde{Q}] \xrightarrow{\mu} E'[\widetilde{Q}]$, by definition of $\succeq^{\mathrm{c}}_{\mathrm{bis}}$ we have $Q_i \stackrel{\mu}{\Longrightarrow} Q'$ for some Q' and $Q' \approx E'[\widetilde{Q}]$. This completes the proof, as we have $P' \approx C[\widetilde{P}]$ (by the fact that $\succeq_{\mathrm{bis}}$ implies $\approx$) and $Q' \approx C[\widetilde{Q}]$. (The other side from Q_i follows in the same manner.) See Fig. 3 for a visual illustration. □

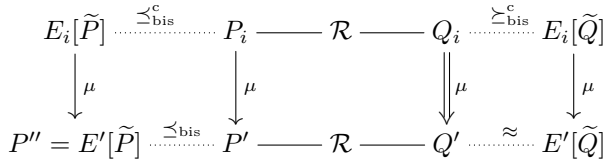

Figure 3: Proof illustration of Theorem 3.16 (showing $P_i \approx^{\mathrm{c}} Q_i$)

4. The formalisation

We highlight here a comprehensive formalisation of CCS in the HOL theorem prover (HOL4) [13, 14], with a focus towards the theory (and formal proofs) of the unique solution of equations/contractions theorems mentioned in Section 3 and 3.2. All proof scripts are available as part of HOL's official examples[2]. The work so far consists of about 24,000 lines (1MB) of code in total, in which about 5,000 lines were derived from the early work of Monica Nesi [15] on HOL88, with major modifications.

Higher Order Logic (HOL) [16] traces its roots back to the *Logic of Computable Functions (LCF)* [17, 18] by Robin Milner and others since 1972. It is a variant of Church's Simple Theory of Types (STT) [19], 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, while some other theorem provers in the HOL family (e.g. Isabelle/HOL) have certain extensions. Indeed, HOL has considerably simpler logical foundations than most other theorem provers. As a consequence, theories and proofs verified in HOL are easier to understand for people who are not familar with more advanced dependent type theories, e.g. the Calculus of Constructions implemented in Coq.

HOL4 is implemented in Standard ML, and the same programming language plays three different roles:

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

- The underlying implementation language for the core HOL engine;
- The language in which proof tactics are implemented;
- The interface language of the HOL proof scripts and interactive shell.

Moreover, using the same language HOL4 users can write complex automatic verification tools by calling HOL's theorem proving facilities. The formal proofs of the CCS theorems that we have carried out are mostly done in a manner that closely follows their paper proofs, with minimal automatic proof searching.

4.1. Higher Order Logic (HOL)

HOL is a formal system of typed logical terms. The types are expressions that denote sets (in the universe $\mathcal{U}$). HOL type system is much simpler than those based on dependent types and other type theories. There are four kinds of types in the HOL logic, as illustrated in Fig. 4 for its BNF grammar. In HOL, the standard atomic types *bool* and *ind* denote, respectively, the distinguished two-element set **2** and the distinguished infinite set **I**.

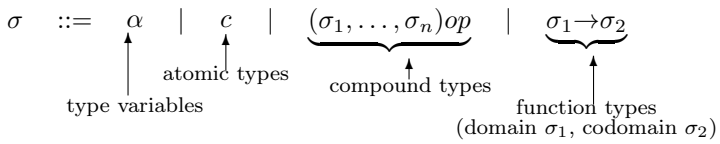

Figure 4: HOL's type grammar

HOL terms represent elements of the sets denoted by their types. There are four kinds of HOL terms, which can be described (in simplified forms) by the BNF grammar in Fig. 5. (See [16] for a complete description of HOL, including the primitive derivative rules to be mentioned below.)

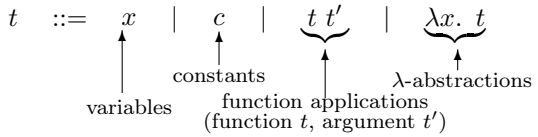

Figure 5: HOL's term grammar

The deductive system of HOL is specified by eight primitive derivative rules:

1. Assumption introduction (`ASSUME`);
2. Reflexivity (`REFL`);
3. β-conversion (`BETA_CONV`);
4. Substitution (`SUBST`);
5. Abstraction (`ABS`);
6. Type instantiation (`INST_TYPE`);
7. Discharging an assumption (`DISCH`);
8. Modus Ponens (`MP`).

All proofs are eventually reduced to applications of the above rules, which also give the semantics of two foundamental logical connectives, equality (=) and implication ($\Rightarrow$). The remaining logical connectives and

first-order quantifiers, including the logical true (T) and false (F), are further defined as λ-functions:

$$\vdash \mathrm{T} \stackrel{\text{def}}{=} ((\lambda x_{bool}.\ x) = (\lambda x_{bool}.\ x))$$
$$\vdash \forall \stackrel{\text{def}}{=} \lambda P_{\alpha\to bool}.\ P = (\lambda x.\ \mathrm{T})$$
$$\vdash \exists \stackrel{\text{def}}{=} \lambda P_{\alpha\to bool}.\ P(\varepsilon\, P)$$
$$\vdash \mathrm{F} \stackrel{\text{def}}{=} \forall b_{bool}.\ b$$
$$\vdash \neg \stackrel{\text{def}}{=} \lambda b.\ b \Rightarrow \mathrm{F}$$
$$\vdash \wedge \stackrel{\text{def}}{=} \lambda b_1\ b_2.\ \forall b.\ (b_1 \Rightarrow (b_2 \Rightarrow b)) \Rightarrow b$$
$$\vdash \vee \stackrel{\text{def}}{=} \lambda b_1\ b_2.\ \forall b.\ (b_1 \Rightarrow b) \Rightarrow ((b_2 \Rightarrow b) \Rightarrow b)$$
$$\vdash \mathrm{One\_One} \stackrel{\text{def}}{=} \lambda f_{\alpha\to\beta}.\ \forall x_1\ x_2.\ (f\ x_1 = f\ x_2) \Rightarrow (x_1 = x_2)$$
$$\vdash \mathrm{Onto} \stackrel{\text{def}}{=} \lambda f_{\alpha\to\beta}.\ \forall y.\ \exists x.\ y = f\ x$$
$$\vdash \mathrm{Type\_Definition} \stackrel{\text{def}}{=} \lambda P_{\alpha\to bool}\ rep_{\beta\to\alpha}.\ \mathrm{One\_One}\ rep\ \wedge\ (\forall x.\ Px = (\exists y.\ x = rep\ y))$$

The last logical constant, Type_Definition, can be used to define new HOL types as bijections of subsets of existing types [20]. HOL `Datatype` package [21, 22] automates this tedious process, and can be used for defining the types needed for CCS. Finally, the whole HOL *standard* theory is based on the following four axioms:[3]

`BOOL_CASES_AX`	$\vdash \forall b.\ (b = \mathrm{T})\ \vee\ (b = \mathrm{F})$
`ETA_AX`	$\vdash \forall f_{\alpha\to\beta}.\ (\lambda x.\ f\ x) = f$
`SELECT_AX`	$\vdash \forall P_{\alpha\to bool}\ x.\ P\ x \Rightarrow P(\varepsilon\ P)$
`INFINITY_AX`	$\vdash \exists f_{ind\to ind}.\ \mathrm{One\_One}\ f\ \wedge\ \neg(\mathrm{Onto}\ f)$

Usually the above four axioms are the only axioms allowed in conventional formalisation projects in HOL4: adding new axioms manually may break logical consistency.

4.2. The CCS formalisation

The CCS formalisation starts with type definitions for action, relabeling and then the processes. We use the type "β `Label`" for all labels (i.e. visible actions), where the type variable β corresponds to $\mathscr{L}$ (the set of names for labels) mentioned at the beginning of Section 2. (Thus the cardinality of "β `Label`" depends on its type variable: when β is finite or countable, "β `Label`" is countable.) All labels are divided into input and output ones. The type "β `Label`" is defined by HOL's `Datatype` package in the following syntax:

```
Datatype: Label = name 'b | coname 'b
End
```

Intuitively, "β `Label`" turns names in $\mathscr{L}$ into *input and output labels*. For instance, if the type β is instantiated to `string` (the type of all ASCII strings), then the HOL terms `name` "a" and `coname` "b" denote the input label a and output label $\overline{b}$, respectively. The type "β `Action`" is the union of all visible actions (input and output labels) and the invisible action τ (`tau`). For instace, the input *action* a and output *action* $\overline{b}$ of type "`string Action`" are denoted by `label` (`name` "a") and `label` (`coname` "b"), respectively. On the other hand, `label` (`name` 1) and `label` (`coname` 2) could be actions of type "`num Action`", where `num` is the type of natural numbers in HOL.

The type "(α, β) `CCS`", accounting for all CCS terms, has two type variables α and β corresponding to the set of agent variables $\mathscr{X}$ and the set of names $\mathscr{L}$, respectively. (Indeed the CCS syntax in Section 2 is parametric with respect to the choice of these two sets.) The type "(α, β) `CCS`" is defined inductively by the `Datatype` package (here "β `Relabeling`" is the type of all relabeling functions; we have also formalised relabeling, though it is not discussed below):

[3] HOL is strictly weaker than ZFC (the Zermelo-Frankel set theory with the Axiom of Choice), thus not all theorems valid in ZFC can be formalised in HOL. (See [16] for more details.)

```
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
End
```

The above definition allows us to write terms like `nil` and `sum P Q` in HOL4. Their correspondences with conventional CCS notations in the literature are given in Table 1, where most CCS operators have also more readable abbreviated forms, either for end users or for TEX outputs. (All formal theorems and definitions in this paper are generated from HOL4. Also, by default, all theorems are fully specialised with outermost universal quantifiers removed.)

CCS concept	Notation	HOL term	HOL abbrev.	TEX outputs
nil	$\mathbf{0}$	`nil`	`nil`	$\mathbf{0}$
prefix	$\mu.P$	`prefix u P`	`u..P`	$u.P$
summation	$P+Q$	`sum P Q`	`P + Q`	$P + Q$
parallel composition	$P \mid Q$	`par P Q`	`P || Q`	$P \mid Q$
restriction	$(\nu L)\ P$	`restr L P`	`(nu L) P`	$(\nu L)\ P$
recursion	$\mathtt{rec}\ A.P$	`rec A P`	`rec A P`	`rec` $A\ P$
relabeling	$P\,[rf]$	`relab P rf`	`relab P rf`	`relab` $P\ rf$
constant/variable	A	`var A`	`var A`	`var` A
invisible action	τ	`tau`	`tau`	τ
input action	a	`label (name a)`	`In(a)`	$\mathrm{in}\ a$
output action	$\overline{a}$	`label (coname a)`	`Out(a)`	$\overline{\mathrm{out}}\ a$
variable substitution	$E\{E'/X\}$	`CCS_Subst E E' X`	`[E'/X] E`	$[E'/X]\ E$
transition	$P \xrightarrow{\mu} Q$	`TRANS P u Q`	`P --u-> Q`	$P\ {-u}\!\rightarrow\ Q$
weak transition	$P \stackrel{\mu}{\Longrightarrow} Q$	`WEAK_TRANS P u Q`	`P ==u=> Q`	$P\ {=u}\!\Rightarrow\ Q$
ϵ–transition	$P \stackrel{\epsilon}{\Longrightarrow} Q$	`EPS P Q`	`EPS P Q`	$P \stackrel{\epsilon}{\Longrightarrow} Q$

Table 1: Syntax of some CCS concepts in HOL

The transition semantics of CCS processes strictly follows the SOS rules given in Fig. 1. However, they are not axioms but consequences of an *inductive relation* definition of `TRANS` by the `HOL_reln` function of HOL4 (see [22, p. 219] for more details). The successful invocation of the definitional principle returns three important theorems (`TRANS_rules`, `TRANS_ind` and `TRANS_cases`):

- `TRANS_rules` is a conjunction of implications (each corresponding to a SOS rule) that will be the same as the input term. In fact, the following formal versions of SOS rules are extracted from `TRANS_rules`:

$$\vdash\ u.E\ {-u}\!\rightarrow\ E \qquad \texttt{[PREFIX]}$$
$$\vdash\ E\ {-u}\!\rightarrow\ E_1 \implies E + E'\ {-u}\!\rightarrow\ E_1 \qquad \texttt{[SUM1]}$$
$$\vdash\ E\ {-u}\!\rightarrow\ E_1 \implies E' + E\ {-u}\!\rightarrow\ E_1 \qquad \texttt{[SUM2]}$$
$$\vdash\ E\ {-u}\!\rightarrow\ E_1 \implies E \mid E'\ {-u}\!\rightarrow\ E_1 \mid E' \qquad \texttt{[PAR1]}$$
$$\vdash\ E\ {-u}\!\rightarrow\ E_1 \implies E' \mid E\ {-u}\!\rightarrow\ E' \mid E_1 \qquad \texttt{[PAR2]}$$

$$\frac{E\ {-\texttt{label}\ l}\!\rightarrow\ E_1 \quad E'\ {-\texttt{label}\ (\texttt{COMPL}\ l)}\!\rightarrow\ E_2}{E \mid E'\ {-\tau}\!\rightarrow\ E_1 \mid E_2}\ \textsf{PAR3}$$

$$\frac{E\ {-u}\!\rightarrow\ E' \quad u = \tau \vee u = \texttt{label}\ l \wedge l \notin L \wedge \texttt{COMPL}\ l \notin L}{(\nu L)\ E\ {-u}\!\rightarrow\ (\nu L)\ E'}\ \textsf{RESTR}$$

$$\frac{E\ -u\rightarrow\ E'}{\texttt{relab}\ E\ rf\ -\texttt{relabel}\ rf\ u\rightarrow\ \texttt{relab}\ E'\ rf}\ \text{RELABELING}$$

$$\frac{[\texttt{rec}\ X\ E/X]\ E\ -u\rightarrow\ E_1}{\texttt{rec}\ X\ E\ -u\rightarrow\ E_1}\ \text{REC}$$

- `TRANS_ind` is the induction principle for the relation (see Section 5 for its exact statement and an application in the proof of Proposition 5.1).

- `TRANS_cases` is the so-called 'cases' or 'inversion' theorem for the relation, and is used to decompose an element in the relation into the possible ways of obtaining it by the rules:

$$\begin{aligned}
\vdash\ \forall\, a_0\ a_1\ a_2\,.\ & \\
& a_0\ -a_1\rightarrow\ a_2\ \Longleftrightarrow \\
& a_0 = a_1.a_2 \vee (\exists\, E\ E'.\ a_0 = E + E' \wedge E\ -a_1\rightarrow\ a_2) \vee \\
& (\exists\, E\ E'.\ a_0 = E' + E \wedge E\ -a_1\rightarrow\ a_2) \vee \\
& (\exists\, E\ E_1\ E'.\ a_0 = E \mid E' \wedge a_2 = E_1 \mid E' \wedge E\ -a_1\rightarrow\ E_1) \vee \\
& (\exists\, E\ E_1\ E'.\ a_0 = E' \mid E \wedge a_2 = E' \mid E_1 \wedge E\ -a_1\rightarrow\ E_1) \vee \\
& (\exists\, E\ l\ E_1\ E'\ E_2\,. \\
& \qquad a_0 = E \mid E' \wedge a_1 = \tau \wedge a_2 = E_1 \mid E_2 \wedge E\ -\texttt{label}\ l\rightarrow\ E_1 \wedge \\
& \qquad E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\rightarrow\ E_2) \vee \\
& (\exists\, E\ E'\ l\ L\,. \\
& \qquad a_0 = (\nu L)\ E \wedge a_2 = (\nu L)\ E' \wedge E\ -a_1\rightarrow\ E' \wedge \\
& \qquad (a_1 = \tau \vee a_1 = \texttt{label}\ l \wedge l \notin L \wedge \texttt{COMPL}\ l \notin L)) \vee \\
& (\exists\, E\ u\ E'\ rf\,. \\
& \qquad a_0 = \texttt{relab}\ E\ rf \wedge a_1 = \texttt{relabel}\ rf\ u \wedge a_2 = \texttt{relab}\ E'\ rf \wedge \\
& \qquad E\ -u\rightarrow\ E') \vee \exists\, E\ X\,.\ a_0 = \texttt{rec}\ X\ E \wedge [\texttt{rec}\ X\ E/X]\ E\ -a_1\rightarrow\ a_2
\end{aligned}$$

For instance, the following proposition requires `TRANS_cases`:

$$\vdash\ E + E'\ -u\rightarrow\ E''\ \Longleftrightarrow\ E\ -u\rightarrow\ E'' \vee E'\ -u\rightarrow\ E'' \qquad \texttt{[TRANS\_SUM\_EQ]}$$

In particular, the SOS rule `REC` (Recursion) says that if we substitute all occurrences of the variable A in P to $(\texttt{rec}\, A.\, P)$ and the resulting process has a transition to P' with an action u, then $(\texttt{rec}\, A.\, P)$ has the same transition. Here "$[\texttt{rec}\ X\ E/X]\ E$" is an abbreviation for "`CCS_Subst` E $(\texttt{rec}\ X\ E)\ X$", where `CCS_Subst` is a recursive function substituting all occurrences of a free variable with a CCS term. For most CCS operators `CCS_Subst` just recursively goes into a deeper level without changing anything, e.g.:

$$\vdash\ [E'/X]\ (E_1\ +\ E_2) = [E'/X]\ E_1\ +\ [E'/X]\ E_2 \qquad \texttt{[CCS\_Subst\_sum]}$$

The only two insteresting cases are those for agent variables and recursion:

$$\vdash\ [E'/X]\ (\texttt{var}\ Y) = \textbf{if}\ Y = X\ \textbf{then}\ E'\ \textbf{else}\ \texttt{var}\ Y \qquad \texttt{[CCS\_Subst\_var]}$$
$$\vdash\ [E'/X]\ (\texttt{rec}\ Y\ E) = \textbf{if}\ Y = X\ \textbf{then}\ \texttt{rec}\ Y\ E\ \textbf{else}\ \texttt{rec}\ Y\ ([E'/X]\ E) \qquad \texttt{[CCS\_Subst\_rec]}$$

Notice that variable substitutions only affect free variables. For instance, the variable Y in "$\texttt{rec}\, Y\, E$" is bound and therefore the substitution ignores it.

A useful facility exploiting the interplay between HOL4 and Standard ML (which follows an idea of Nesi [15]) is a recursive ML function that takes a CCS process and returns a theorem indicating all direct transitions of the process. (If the input process is infinitely branching, the function will not terminate, however.) For instance, we know that the process $(a.\,\mathbf{0} \mid \bar{a}.\,\mathbf{0})$ has three immediate derivatives given by the following transitions: $(a.\,\mathbf{0} \mid \bar{a}.\,\mathbf{0}) \xrightarrow{a} (\mathbf{0} \mid \bar{a}.\,\mathbf{0})$, $(a.\,\mathbf{0} \mid \bar{a}.\,\mathbf{0}) \xrightarrow{\bar{a}} (a.\,\mathbf{0} \mid \mathbf{0})$ and $(a.\,\mathbf{0} \mid \bar{a}.\,\mathbf{0}) \xrightarrow{\tau} (\mathbf{0} \mid \mathbf{0})$. To completely describe all possible transitions of the process, the following two facts have to be proved: (1) there are indeed the three transitions mentioned above; (2) there is no other transition. For large CCS processes

it is surprisingly tedious to manually derive all the possible transitions and prove the non-existence of other transitions. This shows the usefulness of appealing to an ML function `CCS_TRANS_CONV` that is designed to automate the whole process. For instance, taking the input $(a.\,\mathbf{0} \mid \bar{a}.\,\mathbf{0})$ the function returns the following theorem which describes all its one-step transitions:

$$\begin{array}{l}\vdash \text{in "a"}.\mathbf{0} \mid \overline{\text{out}}\ \text{"a"}.\mathbf{0} \xrightarrow{u} E \iff \\ \quad (u = \text{in "a"} \wedge E = \mathbf{0} \mid \overline{\text{out}}\ \text{"a"}.\mathbf{0} \vee u = \overline{\text{out}}\ \text{"a"} \wedge E = \text{in "a"}.\mathbf{0} \mid \mathbf{0}) \vee \\ \quad u = \tau \wedge E = \mathbf{0} \mid \mathbf{0}\end{array}$$

4.3. Bisimulation and Bisimilarity

A highlight of this CCS formalisation is the simplified definitions of bisimilarities using the new coinduction package of HOL4. Without this package, bisimilaries can still be defined in HOL, but proving their properties would be more tedious and complicated [2, p. 91]. Below we briefly describe how weak bisimulation and weak bisimilarity are defined in HOL. Strong bisimulation and strong bisimilarity, as well as other concepts such as expansion and contraction, can be defined in the same manner.

To define (weak) bisimilarity, first we need to define weak transitions of CCS processes. A (possibly empty) sequence of τ-transitions between two processes is defined as a new binary relation `EPS` ($\stackrel{\epsilon}{\Longrightarrow}$), which is the reflexive and transitive closure (RTC, denoted by a superscript *) of ordinary τ-transitions of CCS processes:

$$\texttt{EPS} \stackrel{\text{def}}{=} (\lambda E\ E'.\ E \xrightarrow{\tau} E')^* \qquad \texttt{[EPS\_def]}$$

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

$$E \stackrel{u}{\Rightarrow} E' \stackrel{\text{def}}{=} \exists E_1\ E_2.\ E \stackrel{\epsilon}{\Longrightarrow} E_1 \wedge E_1 \xrightarrow{u} E_2 \wedge E_2 \stackrel{\epsilon}{\Longrightarrow} E' \qquad \texttt{[WEAK\_TRANS]}$$

The definition of weak bisimulation is then based on weak and ϵ–transitions:

$$\begin{array}{l}\texttt{WEAK\_BISIM}\ \mathit{Wbsm} \stackrel{\text{def}}{=} \\ \quad \forall E\ E'. \\ \qquad \mathit{Wbsm}\ E\ E' \Longrightarrow \\ \qquad (\forall l. \\ \qquad\quad (\forall E_1.\ E \xrightarrow{\texttt{label}\ l} E_1 \Longrightarrow \exists E_2.\ E' \stackrel{\texttt{label}\ l}{\Rightarrow} E_2 \wedge \mathit{Wbsm}\ E_1\ E_2) \wedge \\ \qquad\quad \forall E_2.\ E' \xrightarrow{\texttt{label}\ l} E_2 \Longrightarrow \exists E_1.\ E \stackrel{\texttt{label}\ l}{\Rightarrow} E_1 \wedge \mathit{Wbsm}\ E_1\ E_2) \wedge \\ \qquad (\forall E_1.\ E \xrightarrow{\tau} E_1 \Longrightarrow \exists E_2.\ E' \stackrel{\epsilon}{\Longrightarrow} E_2 \wedge \mathit{Wbsm}\ E_1\ E_2) \wedge \\ \qquad \forall E_2.\ E' \xrightarrow{\tau} E_2 \Longrightarrow \exists E_1.\ E \stackrel{\epsilon}{\Longrightarrow} E_1 \wedge \mathit{Wbsm}\ E_1\ E_2 \qquad \texttt{[WEAK\_BISIM]}\end{array}$$

We can prove, for example, that the identity relation $(\lambda x\,y.\,x = y)$ is indeed a bisimulation, and that bisimulation is preserved by inversion, composition, and union operations. Weak bisimilarity can be then defined in HOL4 by the following code:

```
CoInductive WEAK_EQUIV :
    !(E :('a, 'b) CCS) (E' :('a, 'b) CCS).
       (!l.
         (!E1. TRANS E  (label l) E1 ==>
               (?E2. WEAK_TRANS E' (label l) E2 /\ WEAK_EQUIV E1 E2)) /\
         (!E2. TRANS E' (label l) E2 ==>
               (?E1. WEAK_TRANS E  (label l) E1 /\ WEAK_EQUIV E1 E2))) /\
       (!E1. TRANS E  tau E1 ==> (?E2. EPS E' E2 /\ WEAK_EQUIV E1 E2)) /\
       (!E2. TRANS E' tau E2 ==> (?E1. EPS E  E1 /\ WEAK_EQUIV E1 E2))
      ==> WEAK_EQUIV E E'
End
```

Like the case of `TRANS`, the successful invocation of the coinductive definitional principle returns three theorems (`WEAK_EQUIV_rules`, `WEAK_EQUIV_coind` and `WEAK_EQUIV_cases`):

- `WEAK_EQUIV_rules` is the same as the input term, which now becomes a theorem:

$$\begin{array}{l}\vdash (\forall l.\\ \quad\quad (\forall E_1.\ E\ -\texttt{label}\ l\rightarrow\ E_1 \implies \exists E_2.\ E'\ =\texttt{label}\ l\Rightarrow\ E_2\ \wedge\ E_1\ \approx\ E_2)\ \wedge\\ \quad\quad \forall E_2.\ E'\ -\texttt{label}\ l\rightarrow\ E_2 \implies \exists E_1.\ E\ =\texttt{label}\ l\Rightarrow\ E_1\ \wedge\ E_1\ \approx\ E_2)\ \wedge\\ \quad (\forall E_1.\ E\ -\tau\rightarrow\ E_1 \implies \exists E_2.\ E'\ \stackrel{\epsilon}{\Longrightarrow}\ E_2\ \wedge\ E_1\ \approx\ E_2)\ \wedge\\ \quad (\forall E_2.\ E'\ -\tau\rightarrow\ E_2 \implies \exists E_1.\ E\ \stackrel{\epsilon}{\Longrightarrow}\ E_1\ \wedge\ E_1\ \approx\ E_2) \implies\\ \quad E\ \approx\ E'\end{array}$$

- `WEAK_EQUIV_coind` is the coinduction principle for `WEAK_EQUIV` ($\approx$):

$$\begin{array}{l}\vdash (\forall a_0\ a_1.\\ \quad\quad WEAK\_EQUIV'\ a_0\ a_1 \implies\\ \quad\quad (\forall l.\\ \quad\quad\quad (\forall E_1.\\ \quad\quad\quad\quad a_0\ -\texttt{label}\ l\rightarrow\ E_1 \implies\\ \quad\quad\quad\quad \exists E_2.\ a_1\ =\texttt{label}\ l\Rightarrow\ E_2\ \wedge\ WEAK\_EQUIV'\ E_1\ E_2)\ \wedge\\ \quad\quad\quad \forall E_2.\\ \quad\quad\quad\quad a_1\ -\texttt{label}\ l\rightarrow\ E_2 \implies \exists E_1.\ a_0\ =\texttt{label}\ l\Rightarrow\ E_1\ \wedge\ WEAK\_EQUIV'\ E_1\ E_2)\ \wedge\\ \quad\quad (\forall E_1.\ a_0\ -\tau\rightarrow\ E_1 \implies \exists E_2.\ a_1\ \stackrel{\epsilon}{\Longrightarrow}\ E_2\ \wedge\ WEAK\_EQUIV'\ E_1\ E_2)\ \wedge\\ \quad\quad \forall E_2.\ a_1\ -\tau\rightarrow\ E_2 \implies \exists E_1.\ a_0\ \stackrel{\epsilon}{\Longrightarrow}\ E_1\ \wedge\ WEAK\_EQUIV'\ E_1\ E_2) \implies\\ \quad \forall a_0\ a_1.\ WEAK\_EQUIV'\ a_0\ a_1 \implies a_0\ \approx\ a_1\end{array}$$

- `WEAK_EQUIV_cases` is the so-called 'cases' or 'inversion' theorem for the relations, and is used to decompose an element in the relation into the possible ways of obtaining it by the rules:

$$\begin{array}{l}\vdash a_0\ \approx\ a_1\ \iff\\ \quad (\forall l.\\ \quad\quad (\forall E_1.\ a_0\ -\texttt{label}\ l\rightarrow\ E_1 \implies \exists E_2.\ a_1\ =\texttt{label}\ l\Rightarrow\ E_2\ \wedge\ E_1\ \approx\ E_2)\ \wedge\\ \quad\quad \forall E_2.\ a_1\ -\texttt{label}\ l\rightarrow\ E_2 \implies \exists E_1.\ a_0\ =\texttt{label}\ l\Rightarrow\ E_1\ \wedge\ E_1\ \approx\ E_2)\ \wedge\\ \quad (\forall E_1.\ a_0\ -\tau\rightarrow\ E_1 \implies \exists E_2.\ a_1\ \stackrel{\epsilon}{\Longrightarrow}\ E_2\ \wedge\ E_1\ \approx\ E_2)\ \wedge\\ \quad \forall E_2.\ a_1\ -\tau\rightarrow\ E_2 \implies \exists E_1.\ a_0\ \stackrel{\epsilon}{\Longrightarrow}\ E_1\ \wedge\ E_1\ \approx\ E_2\end{array}$$

The coinduction principle `WEAK_EQUIV_coind` says that any bisimulation is contained in the resulting relation. The purpose of `WEAK_EQUIV_cases` is to further assert that such resulting relation is indeed a fixed point. Thus `WEAK_EQUIV_coind` and `WEAK_EQUIV_cases` together make sure that bisimilarity is the greatest fixed point.

The original definition of `WEAK_EQUIV` now becomes a theorem:

$$\vdash E\ \approx\ E' \iff \exists Wbsm.\ Wbsm\ E\ E'\ \wedge\ \texttt{WEAK\_BISIM}\ Wbsm \qquad \texttt{[WEAK\_EQUIV]}$$

The formal definition of rooted bisimilarity ($\approx^{\mathrm{c}}$, `OBS_CONGR`) is not recursive and follows Definition 2.2:

$$\begin{array}{l}E\ \approx^{\mathrm{c}}\ E' \stackrel{\text{def}}{=}\\ \quad \forall u.\\ \quad\quad (\forall E_1.\ E\ -u\rightarrow\ E_1 \implies \exists E_2.\ E'\ =u\Rightarrow\ E_2\ \wedge\ E_1\ \approx\ E_2)\ \wedge\\ \quad\quad \forall E_2.\ E'\ -u\rightarrow\ E_2 \implies \exists E_1.\ E\ =u\Rightarrow\ E_1\ \wedge\ E_1\ \approx\ E_2 \qquad \texttt{[OBS\_CONGR]}\end{array}$$

Below is the formal version of Lemma 2.3 (`OBS_CONGR_BY_WEAK_BISIM`), which is needed later, in the proof of Theorem 3.16:

$$\vdash \texttt{WEAK\_BISIM}\ \mathit{Wbsm} \Longrightarrow$$
$$\forall E\ E'.$$
$$(\forall u.$$
$$(\forall E_1.\ E\ -u\rightarrow\ E_1 \Longrightarrow \exists E_2.\ E'\ =u\Rightarrow\ E_2 \wedge \mathit{Wbsm}\ E_1\ E_2) \wedge$$
$$\forall E_2.\ E'\ -u\rightarrow\ E_2 \Longrightarrow \exists E_1.\ E\ =u\Rightarrow\ E_1 \wedge \mathit{Wbsm}\ E_1\ E_2) \Longrightarrow$$
$$E \approx^{c} E'$$

Finally, on the relationship between (weak) bisimilarity and rooted bisimilarity, we have proved Deng's Lemma and Hennessy's Lemma (Lemma 4.1 and 4.2 of [9, p. 176, 178]):

$$\vdash \forall p\ q.$$
$$p \approx q \Longrightarrow$$
$$(\exists p'.\ p\ -\tau\rightarrow\ p' \wedge p' \approx q) \vee (\exists q'.\ q\ -\tau\rightarrow\ q' \wedge p \approx q') \vee p \approx^{c} q \qquad \texttt{[DENG\_LEMMA]}$$

$$\vdash \forall p\ q.\ p \approx q \iff p \approx^{c} q \vee p \approx^{c} \tau.q \vee \tau.p \approx^{c} q \qquad \texttt{[HENNESSY\_LEMMA]}$$

These are useful results in the theory of CCS (though we will not need them in the remainder of the paper).

4.4. Algebraic Laws

Having formalised the definitions of strong bisimulation and strong bisimilarity, we can derive *algebraic laws* for the bisimilarities. We only report a few algebraic laws on summation:

`STRONG_SUM_IDEMP:`	$\vdash E + E \sim E$
`STRONG_SUM_COMM:`	$\vdash E + E' \sim E' + E$
`STRONG_SUM_IDENT_L:`	$\vdash \mathbf{0} + E \sim E$
`STRONG_SUM_IDENT_R:`	$\vdash E + \mathbf{0} \sim E$
`STRONG_SUM_ASSOC_R:`	$\vdash E + E' + E'' \sim E + (E' + E'')$
`STRONG_SUM_ASSOC_L:`	$\vdash E + (E' + E'') \sim E + E' + E''$
`STRONG_SUM_MID_IDEMP:`	$\vdash E + E' + E \sim E' + E$
`STRONG_LEFT_SUM_MID_IDEMP:`	$\vdash E + E' + E'' + E' \sim E + E'' + E'$

The first five of them are proven by constructing appropriate bisimulations, and their formal proofs are written in a goal-directed manner [22, Chapter 4]. In constrast, the last three algebraic laws are derived in a forward manner by applications of previous proven laws (without directly using the SOS inference rules and the definition of bisimulation). These algebraic laws also hold for weak bisimilarity and rooted bisimilarity, as these are coarser than strong bisimilarity. For weak bisimilarity and rooted bisimilarity, the following algebraic laws, called τ-laws, hold:

`TAU1:`	$\vdash u.\tau.E \approx^{c} u.E$
`TAU2:`	$\vdash E + \tau.E \approx^{c} \tau.E$
`TAU3:`	$\vdash u.(E + \tau.E') + u.E' \approx^{c} u.(E + \tau.E')$
`TAU_STRAT:`	$\vdash E + \tau.(E' + E) \approx^{c} \tau.(E' + E)$
`TAU_WEAK:`	$\vdash \tau.E \approx E$

4.5. Expansion, Contraction and Rooted Contraction

We formalise and contraction along the lines of strong and weak bisimilarity:

$$\texttt{EXPANSION}\ \mathit{Exp} \stackrel{\text{def}}{=}$$
$$\forall E\ E'.$$
$$\mathit{Exp}\ E\ E' \Longrightarrow$$
$$(\forall l.$$
$$(\forall E_1.\ E\ -\texttt{label}\ l\rightarrow\ E_1 \Longrightarrow \exists E_2.\ E'\ -\texttt{label}\ l\rightarrow\ E_2 \wedge \mathit{Exp}\ E_1\ E_2) \wedge$$
$$\forall E_2.\ E'\ -\texttt{label}\ l\rightarrow\ E_2 \Longrightarrow \exists E_1.\ E\ =\texttt{label}\ l\Rightarrow\ E_1 \wedge \mathit{Exp}\ E_1\ E_2) \wedge$$
$$(\forall E_1.\ E\ -\tau\rightarrow\ E_1 \Longrightarrow \mathit{Exp}\ E_1\ E' \vee \exists E_2.\ E'\ -\tau\rightarrow\ E_2 \wedge \mathit{Exp}\ E_1\ E_2) \wedge$$

$$\forall E_2.\ E' \xrightarrow{\tau} E_2 \implies \exists E_1.\ E \stackrel{\tau}{\Longrightarrow} E_1 \wedge Exp\ E_1\ E_2 \qquad \texttt{[EXPANSION]}$$

$$\vdash P \succeq_{\mathrm{e}} Q \iff \exists Exp.\ Exp\ P\ Q \wedge \texttt{EXPANSION}\ Exp \qquad \texttt{[expands\_thm]}$$

$$\begin{array}{l}\texttt{CONTRACTION}\ Con \stackrel{\texttt{def}}{=} \\ \quad \forall E\ E'. \\ \qquad Con\ E\ E' \implies \\ \qquad (\forall l. \\ \qquad\quad (\forall E_1.\ E \xrightarrow{\texttt{label}\ l} E_1 \implies \exists E_2.\ E' \xrightarrow{\texttt{label}\ l} E_2 \wedge Con\ E_1\ E_2) \wedge \\ \qquad\quad \forall E_2.\ E' \xrightarrow{\texttt{label}\ l} E_2 \implies \exists E_1.\ E \stackrel{\texttt{label}\ l}{\Longrightarrow} E_1 \wedge E_1 \approx E_2) \wedge \\ \qquad (\forall E_1.\ E \xrightarrow{\tau} E_1 \implies Con\ E_1\ E' \vee \exists E_2.\ E' \xrightarrow{\tau} E_2 \wedge Con\ E_1\ E_2) \wedge \\ \qquad \forall E_2.\ E' \xrightarrow{\tau} E_2 \implies \exists E_1.\ E \stackrel{\epsilon}{\Longrightarrow} E_1 \wedge E_1 \approx E_2 \end{array} \qquad \texttt{[CONTRACTION]}$$

$$\vdash P \succeq_{\mathrm{bis}} Q \iff \exists Con.\ Con\ P\ Q \wedge \texttt{CONTRACTION}\ Con \qquad \texttt{[contracts\_thm]}$$

The contraction preorder ($\succeq_{\mathrm{bis}}$) contains the expansion preorder ($\succeq_{\mathrm{e}}$), and they are both contained in weak bisimilarity ($\approx$):

Proposition 4.1. *(Relationships between contraction preorder, expansion preorder and weak bisimilarity)*

1. *(Expansion preorder implies contraction preorder)*

$$\vdash \forall P\ Q.\ P \succeq_{\mathrm{e}} Q \implies P \succeq_{\mathrm{bis}} Q \qquad \texttt{[expands\_IMP\_contracts]}$$

2. *(Contraction preorder implies weak bisimilarity)*

$$\vdash \forall P\ Q.\ P \succeq_{\mathrm{bis}} Q \implies P \approx Q \qquad \texttt{[contracts\_IMP\_WEAK\_EQUIV]}$$

The proofs of properties for contraction are generally harder than those for expansion. This is mostly due to the fact that, although the contraction preorder ($\succeq_{\mathrm{bis}}$) is contained in bisimilarity ($\approx$), a contraction need not be a bisimulation. In another words, the following proposition does not hold:

$$\forall Con.\ \texttt{CONTRACTION}\ Con \implies \texttt{WEAK\_BISIM}\ Con$$

However it does hold that, if $\mathcal{R}$ is a contraction, then $\mathcal{R} \cup \approx$ is a bisimulation. For instance, we can prove `contracts_IMP_WEAK_EQUIV` by constructing a bisimulation $Wbsm$ containing two processes P and Q, given that they are in Con (a contraction):

```
        ∃ Wbsm. Wbsm P Q ∧ WEAK_BISIM Wbsm
------------------------------------
 0.  Con P Q
 1.  CONTRACTION Con
```

To complete the proof, one cannot choose Con for $Wbsm$ and show that Con itself is a bisimulation, but rather that $Con \cup \approx$ is a bisimulation. In contrast, in the corresponding lemma for expansion one can just take Con. Finally, the rooted contraction ($\succeq^{\mathrm{c}}_{\mathrm{bis}}$) is formalised as follows:

$$\begin{array}{l} E \succeq^{\mathrm{c}}_{\mathrm{bis}} E' \stackrel{\texttt{def}}{=} \\ \quad \forall u. \\ \qquad (\forall E_1.\ E \xrightarrow{u} E_1 \implies \exists E_2.\ E' \xrightarrow{u} E_2 \wedge E_1 \succeq_{\mathrm{bis}} E_2) \wedge \\ \qquad \forall E_2.\ E' \xrightarrow{u} E_2 \implies \exists E_1.\ E \stackrel{u}{\Longrightarrow} E_1 \wedge E_1 \approx E_2 \end{array} \qquad \texttt{[OBS\_contracts]}$$

4.6. The formalisation of "bisimulation up to"

"Bisimulation up to" is a family of powerful proof techniques for reducing the sizes of relations needed for defining bisimulations [23]. By definition, two processes are bisimilar iff there exists a bisimulation relation containing them. However, in practice this definition is sometimes hard to apply. Instead, to reduce the

sizes of the exhibited relations, one prefers to define relations which are bisimulations only when closed up under some specific and privileged relation, so to relieve the needed proof work. These techniques are usually called *"up-to" techniques*.

Recall that we often write $P\ \mathcal{R}\ Q$ to denote $(P,Q) \in \mathcal{R}$ for any binary relation $\mathcal{R}$. Moreover, $\sim \mathcal{S} \sim$ is the composition of three binary relations: $\sim$, $\mathcal{S}$ and $\sim$. Hence $P \sim \mathcal{S} \sim Q$ means that there exist P' and Q' such that $P \sim P'$, $P'\ \mathcal{S}\ Q'$ and $Q' \sim Q$.

Definition 4.2. *$\mathcal{S}$ is a "bisimulation up to $\sim$" if $P\ \mathcal{S}\ Q$ implies, for all μ,*

1. *Whenever $P \xrightarrow{\mu} P'$ then, for some Q', $Q \xrightarrow{\mu} Q'$ and $P' \sim \mathcal{S} \sim Q'$,*
2. *Whenever $Q \xrightarrow{\mu} Q'$ then, for some P', $P \xrightarrow{\mu} P'$ and $P' \sim \mathcal{S} \sim Q'$.*

Theorem 4.3. *If $\mathcal{S}$ is a "bisimulation up to $\sim$", then $\mathcal{S} \subseteq \sim$:*

$\vdash$ `STRONG_BISIM_UPTO` $Bsm \wedge Bsm\ P\ Q \Longrightarrow P \sim Q$ `[STRONG_EQUIV_BY_BISIM_UPTO]`

Hence, to prove $P \sim Q$, one only needs to find a bisimulation up to $\sim$ that contains (P,Q). For weak bisimilarity, the naive weak bisimulation up to weak bisimilarity is unsound: if one simply replaces all occurrences of $\sim$ in Def. 4.2 with $\approx$, the resulting "weak bisimulation up to" relation need not be contained in weak bisimilarity ($\approx$) [24]. There are a few ways to fix this problem, and one is the following:

Definition 4.4. *$\mathcal{S}$ is a "bisimulation up to $\approx$" if $P\ \mathcal{S}\ Q$ implies, for all μ,*

1. *Whenever $P \xrightarrow{\mu} P'$ then, for some Q', $Q \xRightarrow{\widehat{\mu}} Q'$ and $P' \sim \mathcal{S} \approx Q'$,*
2. *Whenever $Q \xrightarrow{\mu} Q'$ then, for some P', $P \xRightarrow{\widehat{\mu}} P'$ and $P' \approx \mathcal{S} \sim Q'$.*

Formally:

$$
\begin{array}{l}
\texttt{WEAK\_BISIM\_UPTO}\ Wbsm \stackrel{\text{def}}{=} \\
\quad \forall E\ E'. \\
\qquad Wbsm\ E\ E' \Longrightarrow \\
\qquad (\forall l. \\
\qquad\quad (\forall E_1. \\
\qquad\qquad E\ -\texttt{label}\ l\rightarrow\ E_1 \Longrightarrow \\
\qquad\qquad \exists E_2. \\
\qquad\qquad\quad E'\ =\texttt{label}\ l\Rightarrow\ E_2 \wedge \\
\qquad\qquad\quad (\texttt{WEAK\_EQUIV} \circ_r Wbsm \circ_r \texttt{STRONG\_EQUIV})\ E_1\ E_2) \wedge \\
\qquad\quad \forall E_2. \\
\qquad\qquad E'\ -\texttt{label}\ l\rightarrow\ E_2 \Longrightarrow \\
\qquad\qquad \exists E_1. \\
\qquad\qquad\quad E\ =\texttt{label}\ l\Rightarrow\ E_1 \wedge \\
\qquad\qquad\quad (\texttt{STRONG\_EQUIV} \circ_r Wbsm \circ_r \texttt{WEAK\_EQUIV})\ E_1\ E_2) \wedge \\
\qquad (\forall E_1. \\
\qquad\quad E\ -\tau\rightarrow\ E_1 \Longrightarrow \\
\qquad\quad \exists E_2.\ E' \stackrel{\epsilon}{\Longrightarrow} E_2 \wedge (\texttt{WEAK\_EQUIV} \circ_r Wbsm \circ_r \texttt{STRONG\_EQUIV})\ E_1\ E_2) \wedge \\
\qquad \forall E_2. \\
\qquad\quad E'\ -\tau\rightarrow\ E_2 \Longrightarrow \\
\qquad\quad \exists E_1.\ E \stackrel{\epsilon}{\Longrightarrow} E_1 \wedge (\texttt{STRONG\_EQUIV} \circ_r Wbsm \circ_r \texttt{WEAK\_EQUIV})\ E_1\ E_2
\end{array}
$$

Note that the HOL term corresponding to $\sim \mathcal{R} \approx$ is "`WEAK_EQUIV` $\circ_r$ R $\circ_r$ `STRONG_EQUIV`" where the order of $\sim$ and $\approx$ seems reverted. This is because, in HOL notation, the rightmost relation (`STRONG_EQUIV` or $\sim$) in the relational composition is applied first.

Theorem 4.5. *If $\mathcal{S}$ is a bisimulation up to $\approx$, then $\mathcal{S} \subseteq \approx$:*

$\vdash$ `WEAK_BISIM_UPTO` $Bsm \wedge Bsm\ P\ Q \Longrightarrow P \approx Q$ `[WEAK_EQUIV_BY_BISIM_UPTO]`

The above version of "bisimulation up to $\approx$" is not powerful enough for Milner's "unique solution of equations" theorem for $\approx$ (Theorem 3.6, see [25] for more details). The following version, with weak arrows, is used in the proof of Theorem 3.6:

Definition 4.6. *$\mathcal{S}$ is a "bisimulation up to $\approx$ with weak arrows" if $P\ \mathcal{S}\ Q$ implies, for all μ,*

1. *Whenever $P \stackrel{\mu}{\Longrightarrow} P'$ then, for some Q', $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $P' \approx \mathcal{S} \approx Q'$,*
2. *Whenever $Q \stackrel{\mu}{\Longrightarrow} Q'$ then, for some P', $P \stackrel{\widehat{\mu}}{\Longrightarrow} P'$ and $P' \approx \mathcal{S} \approx Q'$.*

Formally:

```
WEAK_BISIM_UPTO_ALT Wbsm =def=
  ∀E E'.
      Wbsm E E' ⟹
      (∀l.
          (∀E1.
              E =label l⇒ E1 ⟹
              ∃E2.
                  E' =label l⇒ E2 ∧
                  (WEAK_EQUIV ∘r Wbsm ∘r WEAK_EQUIV) E1 E2) ∧
          ∀E2.
              E' =label l⇒ E2 ⟹
              ∃E1.
                  E =label l⇒ E1 ∧
                  (WEAK_EQUIV ∘r Wbsm ∘r WEAK_EQUIV) E1 E2) ∧
      (∀E1.
          E =τ⇒ E1 ⟹
          ∃E2. E' =ϵ⇒ E2 ∧ (WEAK_EQUIV ∘r Wbsm ∘r WEAK_EQUIV) E1 E2) ∧
      ∀E2.
          E' =τ⇒ E2 ⟹
          ∃E1. E =ϵ⇒ E1 ∧ (WEAK_EQUIV ∘r Wbsm ∘r WEAK_EQUIV) E1 E2
```

Theorem 4.7. *If $\mathcal{S}$ is a bisimulation up to $\approx$ with weak arrows, then $\mathcal{S} \subseteq \approx$:*

```
⊢ WEAK_BISIM_UPTO_ALT Bsm ∧ Bsm P Q ⟹ P ≈ Q          [WEAK_EQUIV_BY_BISIM_UPTO_ALT]
```

4.7. Context, guardedness and (pre)congruence

CCS contexts are needed in defining (pre)congruence. To prevent doing variable substitutions, one can take univariable λ-expressions (of type "(α, β) `CCS` $\rightarrow$ (α, β) `CCS`") as *multi-hole CCS contexts*. This choice has a significant advantage over *one-hole contexts*, as each hole corresponds to one occurrence of the (same) variable in univariable CCS expressions or equations. Thus *contexts* can be used both in (pre)congruence definitions and in formulating the unique solution of equations theorems in the univariable case. The precise definition of CCS contexts is inductive:

```
⊢ CONTEXT (λt. t) ∧ (∀p. CONTEXT (λt. p)) ∧
  (∀a e. CONTEXT e ⟹ CONTEXT (λt. a.e t)) ∧
  (∀e1 e2. CONTEXT e1 ∧ CONTEXT e2 ⟹ CONTEXT (λt. e1 t + e2 t)) ∧
  (∀e1 e2. CONTEXT e1 ∧ CONTEXT e2 ⟹ CONTEXT (λt. e1 t | e2 t)) ∧
  (∀L e. CONTEXT e ⟹ CONTEXT (λt. (νL) (e t))) ∧
  ∀rf e. CONTEXT e ⟹ CONTEXT (λt. relab (e t) rf)          [CONTEXT_rules]
```

In the above definition (actually generated by `HOL_reln`), for any process p, $(\lambda\, t.\ p)$ is a valid context with no hole (similarly to an equation without variables).

Below is the formalisation of Def. 3.3. A context is *weakly guarded* (`WG`) if each hole is underneath a prefix:

$$\vdash (\forall p.\ \texttt{WG}\ (\lambda t.\ p)) \wedge (\forall a\ e.\ \texttt{CONTEXT}\ e \Longrightarrow \texttt{WG}\ (\lambda t.\ a.e\ t)) \wedge$$
$$(\forall e_1\ e_2.\ \texttt{WG}\ e_1 \wedge \texttt{WG}\ e_2 \Longrightarrow \texttt{WG}\ (\lambda t.\ e_1\ t + e_2\ t)) \wedge$$
$$(\forall e_1\ e_2.\ \texttt{WG}\ e_1 \wedge \texttt{WG}\ e_2 \Longrightarrow \texttt{WG}\ (\lambda t.\ e_1\ t \mid e_2\ t)) \wedge$$
$$(\forall L\ e.\ \texttt{WG}\ e \Longrightarrow \texttt{WG}\ (\lambda t.\ (\nu L)\ (e\ t))) \wedge$$
$$\forall rf\ e.\ \texttt{WG}\ e \Longrightarrow \texttt{WG}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf) \qquad \texttt{[WG\_rules]}$$

Notice the differences between a weakly guarded context and an ordinary context: $(\lambda t.\, t)$ is not weakly guarded as the variable t is directly exposed without any prefixed action, while $(\lambda t.\, a.\, e[t])$ is weakly guarded as long as $e[\cdot]$ is a context, which is not necessary weakly guarded.

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

$$\vdash (\forall p.\ \texttt{SG}\ (\lambda t.\ p)) \wedge (\forall l\ e.\ \texttt{CONTEXT}\ e \Longrightarrow \texttt{SG}\ (\lambda t.\ \texttt{label}\ l.e\ t)) \wedge$$
$$(\forall a\ e.\ \texttt{SG}\ e \Longrightarrow \texttt{SG}\ (\lambda t.\ a.e\ t)) \wedge$$
$$(\forall e_1\ e_2.\ \texttt{SG}\ e_1 \wedge \texttt{SG}\ e_2 \Longrightarrow \texttt{SG}\ (\lambda t.\ e_1\ t + e_2\ t)) \wedge$$
$$(\forall e_1\ e_2.\ \texttt{SG}\ e_1 \wedge \texttt{SG}\ e_2 \Longrightarrow \texttt{SG}\ (\lambda t.\ e_1\ t \mid e_2\ t)) \wedge$$
$$(\forall L\ e.\ \texttt{SG}\ e \Longrightarrow \texttt{SG}\ (\lambda t.\ (\nu L)\ (e\ t))) \wedge$$
$$\forall rf\ e.\ \texttt{SG}\ e \Longrightarrow \texttt{SG}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf) \qquad \texttt{[SG\_rules]}$$

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:

$$\vdash \texttt{SEQ}\ (\lambda t.\ t) \wedge (\forall p.\ \texttt{SEQ}\ (\lambda t.\ p)) \wedge (\forall a\ e.\ \texttt{SEQ}\ e \Longrightarrow \texttt{SEQ}\ (\lambda t.\ a.e\ t)) \wedge$$
$$\forall e_1\ e_2.\ \texttt{SEQ}\ e_1 \wedge \texttt{SEQ}\ e_2 \Longrightarrow \texttt{SEQ}\ (\lambda t.\ e_1\ t + e_2\ t) \qquad \texttt{[SEQ\_rules]}$$

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

$$\vdash \texttt{GCONTEXT}\ (\lambda t.\ t) \wedge (\forall p.\ \texttt{GCONTEXT}\ (\lambda t.\ p)) \wedge$$
$$(\forall a\ e.\ \texttt{GCONTEXT}\ e \Longrightarrow \texttt{GCONTEXT}\ (\lambda t.\ a.e\ t)) \wedge$$
$$(\forall a_1\ a_2\ e_1\ e_2.$$
$$\quad \texttt{GCONTEXT}\ e_1 \wedge \texttt{GCONTEXT}\ e_2 \Longrightarrow \texttt{GCONTEXT}\ (\lambda t.\ a_1.e_1\ t + a_2.e_2\ t)) \wedge$$
$$(\forall e_1\ e_2.\ \texttt{GCONTEXT}\ e_1 \wedge \texttt{GCONTEXT}\ e_2 \Longrightarrow \texttt{GCONTEXT}\ (\lambda t.\ e_1\ t \mid e_2\ t)) \wedge$$
$$(\forall L\ e.\ \texttt{GCONTEXT}\ e \Longrightarrow \texttt{GCONTEXT}\ (\lambda t.\ (\nu L)\ (e\ t))) \wedge$$
$$\forall rf\ e.\ \texttt{GCONTEXT}\ e \Longrightarrow \texttt{GCONTEXT}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf) \qquad \texttt{[GCONTEXT\_rules]}$$

$$\vdash (\forall p.\ \texttt{WGS}\ (\lambda t.\ p)) \wedge (\forall a\ e.\ \texttt{GCONTEXT}\ e \Longrightarrow \texttt{WGS}\ (\lambda t.\ a.e\ t)) \wedge$$
$$(\forall a_1\ a_2\ e_1\ e_2.\ \texttt{GCONTEXT}\ e_1 \wedge \texttt{GCONTEXT}\ e_2 \Longrightarrow \texttt{WGS}\ (\lambda t.\ a_1.e_1\ t + a_2.e_2\ t)) \wedge$$
$$(\forall e_1\ e_2.\ \texttt{WGS}\ e_1 \wedge \texttt{WGS}\ e_2 \Longrightarrow \texttt{WGS}\ (\lambda t.\ e_1\ t \mid e_2\ t)) \wedge$$
$$(\forall L\ e.\ \texttt{WGS}\ e \Longrightarrow \texttt{WGS}\ (\lambda t.\ (\nu L)\ (e\ t))) \wedge$$
$$\forall rf\ e.\ \texttt{WGS}\ e \Longrightarrow \texttt{WGS}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf) \qquad \texttt{[WGS\_rules]}$$

Several lemmas about the above concepts (CONTEXT, WG, SEQ, etc.) are needed for capturing properties about the relationships among these kinds of contexts and about their compositions. These proofs are usually tedious and long, due to multiple levels of inductions on the structure of the contexts.

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 is an equivalence, while the latter can just be a preorder:

$$\texttt{congruence}\ R \stackrel{\text{def}}{=}$$
$$\quad \texttt{equivalence}\ R \wedge \forall x\ y\ ctx.\ \texttt{CONTEXT}\ ctx \Longrightarrow R\ x\ y \Longrightarrow R\ (ctx\ x)\ (ctx\ y) \qquad \texttt{[congruence]}$$

$$\texttt{precongruence}\ R \stackrel{\text{def}}{=}$$
$$\quad \texttt{PreOrder}\ R \wedge \forall x\ y\ ctx.\ \texttt{CONTEXT}\ ctx \Longrightarrow R\ x\ y \Longrightarrow R\ (ctx\ x)\ (ctx\ y) \qquad \texttt{[precongruence]}$$

Both strong bisimilarity ($\sim$) and rooted bisimilarity ($\approx^c$) are congruence relations:

$\vdash$ `congruence STRONG_EQUIV` [STRONG_EQUIV_congruence]

$\vdash$ `congruence OBS_CONGR` [OBS_CONGR_congruence]

Although weak bisimilarity ($\approx$) is *not* a congruence with respect to `CONTEXT`, it is substitutive with respect to `GCONTEXT` as $\approx$ is preserved by guarded sums. Similarily, contraction ($\succeq_{\text{bis}}$) is substitutive with respect to `GCONTEXT`. Rooted contraction ($\succeq^{\text{c}}_{\text{bis}}$, or `OBS_contracts` in HOL), on the other hand, is indeed a precongruence:

$\vdash$ `precongruence OBS_contracts` [OBS_contracts_precongruence]

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

In this section we give a proof of the second part of Theorem 2.4, i.e. $\approx^{\text{c}}$ is the coarsest congruence contained in $\approx$. The general form of this theorem is the following one [8, 9, 2]:

Proposition 4.8. *Rooted bisimilarity ($\approx^{\text{c}}$) is the coarsest congruence contained in weak bisimilarity ($\approx$):*

$$\forall p\ q.\ p \approx^{\text{c}} q \iff (\forall r.\ p + r \approx q + r)\ . \tag{1}$$

From left to right (1) trivially holds, due to the substitutivity of $\approx^{\text{c}}$ for summation and the fact that $\approx^{\text{c}}$ implies $\approx$: (Thus we are only interested in (1) from right to left.)

$\vdash \forall p\ q.\ p \approx^{\text{c}} q \implies \forall r.\ p + r \approx q + r$ [COARSEST_CONGR_LR]

The formalisation of this theorem presents some delicate aspects. For instance, within our CCS syntax which supports only binary sums, one way to prove Proposition 4.8 is to add an hypothesis that the involved processes do not use all available labels. Indeed, this is the standard argument by Milner [2, p. 153]. Formalising this result, however, requires a detailed treatment of free and bound names (of labels) of CCS processes, with the restriction operator acting as a binder. In our actual formalisation, instead, we assume the weaker hypothesis that all *immediate weak* derivatives of p and q do not use all available labels. We call this the *free action* property:

`free_action` $p \stackrel{\text{def}}{=} \exists a.\ \forall p'.\ \neg(p\ =$`label` $a\Rightarrow\ p')$ [free_action_def]

Now we show how (1) is connected with the statement of Proposition 4.8, and prove it under the free action assumptions. The coarsest congruence contained in (weak) bisimilarity, namely *bisimilarity congruence* (`WEAK_CONGR` in HOL), is the *composition closure* (`CC`) of (weak) bisimilarity:

`CC` $R \stackrel{\text{def}}{=} (\lambda g\ h.\ \forall c.$ `CONTEXT` $c \implies R\ (c\ g)\ (c\ h))$ [CC_def]

`WEAK_CONGR` $\stackrel{\text{def}}{=}$ `CC WEAK_EQUIV` [WEAK_CONGR]

We do not need to put $R\ g\ h$ into the antecedents of `CC_def`, as this is anyhow obtained from the trivial context $(\lambda x.\ x)$. The next result shows that, for any binary relation R on CCS processes, the composition closure of R is always at least as fine as R (here $\subseteq_r$ stands for *relational subset*):

$\vdash \forall R.$ `CC` $R \subseteq_r R$ [CC_is_finer]

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

$\vdash \forall R\ R'.$ `congruence` $R' \wedge R' \subseteq_r R \implies R' \subseteq_r$ `CC` R [CC_is_coarsest]

$\vdash \forall R\ R'.$ `precongruence` $R' \wedge R' \subseteq_r R \implies R' \subseteq_r$ `CC` R [PCC_is_coarsest]

Given the central role of summation, we also consider the relation closure of bisimilarity with respect to summation, called *equivalence compatible with summation* (`SUM_EQUIV`):

`SUM_EQUIV` $\stackrel{\text{def}}{=} (\lambda p\ q.\ \forall r.\ p + r \approx q + r)$ [SUM_EQUIV]

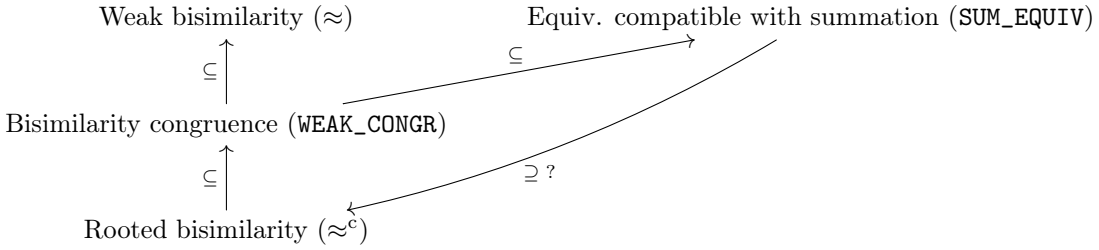

Figure 6: Relationships between several equivalences and ≈

Rooted bisimilarity $\approx^{\mathrm{c}}$ (as a congruence contained in ≈) is now contained in `WEAK_CONGR`, which in turn is trivially contained in `SUM_EQUIV`, as shown in Fig. 6. Thus, to prove Proposition 4.8, the crux is to prove that `SUM_EQUIV` is contained in $\approx^{\mathrm{c}}$, making all three relations ($\approx^{\mathrm{c}}$, `WEAK_CONGR` and `SUM_EQUIV`) coincide:

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

Here is the formalisation of (2) under free action hypothesis:

Theorem 4.9 (`COARSEST_CONGR_RL`)**.** *Under the free action hypothesis, $\approx^{\mathrm{c}}$ is coarsest congruence contained in ≈.*

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

With an almost identical proof, rooted contraction ($\succeq^{\mathrm{c}}_{\mathrm{bis}}$) is also the coarsest precongruence contained in the bisimilarity contraction ($\succeq_{\mathrm{bis}}$):

Theorem 4.10 (`COARSEST_PRECONGR_RL`)**.** *Under the free action hypothesis, $\succeq_{\mathrm{bis}}$ is the coarsest precongruence contained in $\succeq_{\mathrm{bis}}$.*

$$\vdash \forall p\ q.\ \texttt{free\_action}\ p \wedge \texttt{free\_action}\ q \implies (\forall r.\ p + r \succeq_{\mathrm{bis}} q + r) \implies p \succeq^{\mathrm{c}}_{\mathrm{bis}} q$$

The formal proofs of Theorem 4.9 and 4.10 precisely follow Milner [2, p. 153–154]. Although Milner requires a stronger hypothesis: $\mathrm{fn}(p) \cup \mathrm{fn}(q) \neq \mathscr{L}$ (here fn stands for *free names*), the actual proof essentially requires only the above free action property. Indeed, in the proof one only looks at the immediate weak derivatives of p and q, and only requires that there is an input or output label that never occurs as a label of the involved transitions.

4.9. Arbitrarily many non-bisimilar processes

As the type "(α, β) `CCS`" is parameterized with two type variables, if the type of all label names β has a small cardinality (a singleton in the extreme case), it is possible that the processes of Proposition 4.8 use all available labels, and thus the free action hypothesis does not hold. In this case, it is still possible to prove Proposition 4.8; however, due to some limitations of HOL itself we have to assume that the processes are *finite-state*, i.e. the set of all their derivatives is finite. The original proof, due to van Glabbeek [8], does not require finite-state CCS. Here is the main theorem:

Theorem 4.11 (`COARSEST_CONGR_FINITE`)**.** *For finite-state CCS, $\approx^{\mathrm{c}}$ is the coarsest congruence contained in ≈:*

$$\vdash \forall p\ q.\ \texttt{finite\_state}\ p \wedge \texttt{finite\_state}\ q \implies (p \approx^{\mathrm{c}} q \iff \forall r.\ p + r \approx q + r)$$

The precise definition of `finite_state` used in above theorem will be given later. We start with a core lemma (`PROP3_COMMON`) saying that, for any two processes p and q, if there exists a *stable* (i.e. without τ transitions) process which is not bisimilar with any weak derivative of p and q, then `SUM_EQUIV` indeed implies rooted bisimilarity ($\approx^{\mathrm{c}}$) [8, 26]:

$$\vdash \forall p\ q.$$
$$(\exists k.\ \texttt{STABLE}\ k \wedge (\forall p'\ u.\ p =u\Rightarrow p' \Longrightarrow \neg(p' \approx k)) \wedge \forall q'\ u.\ q =u\Rightarrow q' \Longrightarrow \neg(q' \approx k)) \Longrightarrow (\forall r.\ p + r \approx q + r) \Longrightarrow p \approx^{c} q \qquad \texttt{[PROP3\_COMMON]}$$

$$\texttt{STABLE}\ p \stackrel{\text{def}}{=} \forall u\ p'.\ p \xrightarrow{u} p' \Longrightarrow u \neq \tau \qquad \texttt{[STABLE]}$$

To prove Theorem 4.11, it only remains to construct such stable process k for any two finite-state processes p and q. For arbitrary CCS processes, this construction relies on arbitrary infinite sums of processes (not within our CCS syntax) and transfinite induction to obtain an arbitrary large sequence of processes that are all pairwise non-bisimilar, which was firstly introduced by Jan Willem Klop (see [8] for some historical notes). We have only partially formalised van Glabbeek's proof, mostly because our CCS syntax does not allow infinite summation (and it is not easy to extend it with this support). Another more important reason is that the typed logic implemented in various HOL systems (including Isabelle/HOL) is not strong enough to define a type for all possible ordinals [27] which is required in van Glabbeek's proof. As the consequence, the formalisation (Theorem 4.11) can only apply to finite-state CCS.

The above core lemma (`PROP3_COMMON`) requires the existence of a special CCS process, which is not weakly bisimilar to any weak derivative of the two root processes. There could be infinitely many such subprocesses, even on finitely branching processes. We can, however, consider the equivalence classes of CCS processes modulo weak bisimilarity. If there are infinitely many such classes, then it will be possible to choose one that is distinct from all the (finitely many) states in the transition graphs of the two given processes. This can be done by following Klop's contruction. We call the processes in this construction the "Klop processes":

Definition 4.12 (Klop processes)**.** *For each ordinal λ, and an arbitrary chosen action $a \neq \tau$, define a CCS process k_λ as follows:*

- $k_0 = 0$,
- $k_{\lambda+1} = k_\lambda + a.\,k_\lambda$ *and*
- *for λ a limit ordinal, $k_\lambda = \sum_{\mu<\lambda} k_\mu$ (meaning that k_λ is constructed from all graphs k_μ for $\mu < \lambda$ by identifying their root).*

When processes are finite-state, that is, the number of states in which a process may evolve by performing transitions is finite, we can use the following subset of Klop processes, defined as a recursive function (on natural numbers) in HOL4:

Definition 4.13. *(Klop processes as recursive function on natural numbers)*

$$\texttt{KLOP}\ a\ 0 \stackrel{\text{def}}{=} \mathbf{0}$$
$$\texttt{KLOP}\ a\ (\texttt{SUC}\ n) \stackrel{\text{def}}{=} \texttt{KLOP}\ a\ n + \texttt{label}\ a.\texttt{KLOP}\ a\ n \qquad \texttt{[KLOP\_def]}$$

Following the inductive structure of the above definition, and using the SOS rules (Sum_1) and (Sum_2), we can prove the following properties of Klop functions:

Proposition 4.14. *(Properties of Klop functions and processes)*

1. *(All Klop processes are stable)*

   $$\vdash \texttt{STABLE}\ (\texttt{KLOP}\ a\ n) \qquad \texttt{[KLOP\_PROP0]}$$

2. *(Any transition from a Klop process leads to a smaller Klop process, and conversely)*

   $$\vdash \texttt{KLOP}\ a\ n \xrightarrow{\texttt{label}\ a} E \iff \exists m.\ m < n \wedge E = \texttt{KLOP}\ a\ m \qquad \texttt{[KLOP\_PROP1]}$$

3. *(The weak version of the previous property)*

   $\vdash$ KLOP a n $=$label $a\Rightarrow$ E $\iff$ $\exists m.\ m < n \wedge E =$ KLOP a m [KLOP_PROP1']

4. *(All Klop processes are distinct according to strong bisimilarity)*

   $\vdash m < n \implies \neg$(KLOP a $m \sim$ KLOP a n) [KLOP_PROP2]

5. *(All Klop processes are distinct according to weak bisimilarity)*

   $\vdash m < n \implies \neg$(KLOP a $m \approx$ KLOP a n) [KLOP_PROP2']

6. *(Klop functions are one-one)*

   $\vdash$ ONE_ONE (KLOP a) [KLOP_ONE_ONE]

For any "label a", having a function "KLOP a" (of type "num $\to$ (α, β) CCS") defined on the natural numbers, we obtain a countable set of processes with all Klop processes built from the same label. Since the number of all Klop processes in this set is (countably) infinite, and since they are all pairwise non-bisimiar, we can always choose a number that is mapped to a Klop process that is non-bisimilar with any derivative of two given (finite-state) processes p and q, even when a is the only element of type β, i.e. the only label name in $\mathscr{L}$. This property is captured by appealing to the following set-theoreric lamma (see [26] for its proof):

Lemma 4.15. *Given an equivalence relation R defined on a type, and two sets A, B of elements in this type, if A is finite, B is infinite, and all elements in B belong to distinct equivalence classes, then there exists an element k in B which is not equivalent to any element in A:*

$\vdash$ equivalence $R \implies$
FINITE $A \wedge$ INFINITE $B \wedge (\forall x\ y.\ x \in B \wedge y \in B \wedge x \neq y \implies \neg R\ x\ y) \implies$
$\exists k.\ k \in B \wedge \forall n.\ n \in A \implies \neg R\ n\ k$ [INFINITE_EXISTS_LEMMA]

To reason about finite-state CCS, we also need to define the concept of "finite-state CCS" as a predicate on CCS processes:

Definition 4.16 (finite-state CCS)**.**

1. *A binary relation* ***Reach*** *is the RTC (reflexive and transitive closure) of a relation indicating the existence of a transition between two processes:*

   Reach $\stackrel{\text{def}}{=}$ $(\lambda E\ E'.\ \exists u.\ E\ -u\to\ E')^*$ [Reach_def]

2. *The set of all derivatives (****NODES****) of a process is the set of all processes reachable from it:*

   NODES $p \stackrel{\text{def}}{=} \{ q \mid$ Reach $p\ q \}$ [NODES_def]

3. *A process is* ***finite-state*** *if the set of all derivatives is finite:*

   finite_state $p \stackrel{\text{def}}{=}$ FINITE (NODES p) [finite_state_def]

We rely on various properties of the above definitions, such as the following one:

Proposition 4.17. *If p has a weak transition to q, then q is among the derivatives of p:*

$\vdash p\ =u\Rightarrow\ q \implies q \in$ NODES p [WEAK_TRANS_IN_NODES]

Using all the above results, now we can prove the following finite-state version of "Klop lemma":

Lemma 4.18 (Klop lemma for finite-state CCS)**.** *For any two finite-state CCS p and q, there is another process k, which is not weakly bisimilar with any weak derivative of p and q (i.e., any process reachable from p or q by means of transitions):*

$$\vdash \forall p\ q.$$
$$\texttt{finite\_state}\ p \wedge \texttt{finite\_state}\ q \Longrightarrow$$
$$\exists k.$$
$$\texttt{STABLE}\ k \wedge (\forall p'\ u.\ p =u\Rightarrow p' \Longrightarrow \neg(p' \approx k)) \wedge$$
$$\forall q'\ u.\ q =u\Rightarrow q' \Longrightarrow \neg(q' \approx k) \qquad \texttt{[KLOP\_LEMMA\_FINITE]}$$

Combining the above lemma with the core lemma (PROP3_COMMON) and Theorem 4.9 (COARSEST_CONGR_RL), yields the proof of Theorem 4.11 (COARSEST_CONGR_FINITE). The same proof idea can also be used with contraction and rooted contration.

4.10. Unique solution of equations

In this section we describe the formalisation of Milner's "unique solution of equations" theorems, limited to univariable equations. (The multivariable extension is described in Section 5.)

4.10.1. The version for strong bisimilarity

Using "bisimulation up to $\sim$" technique, we obtain the following key lemma, which states that, if X is weakly guarded in E, then the "first move" of E is independent of the agent substituted for X:

Lemma 4.19 (STRONG_UNIQUE_SOLUTION_LEMMA, Lemma 3.13 of [2], univariable version)**.** *If the variable X is weakly guarded in E, and $E\{P/X\} \xrightarrow{\alpha} P'$, then P' takes the form $E'\{P/X\}$ (for some expression E'), and moreover, for any Q, $E\{Q/X\} \xrightarrow{\alpha} E'\{Q/X\}$:*

$$\vdash \texttt{WG}\ E \Longrightarrow$$
$$\forall P\ a\ P'.\ E\ P\ -a\rightarrow\ P' \Longrightarrow \exists E'.\ \texttt{CONTEXT}\ E' \wedge P' = E'\ P \wedge \forall Q.\ E\ Q\ -a\rightarrow\ E'\ Q$$

Then, by structural induction on weakly guarded contexts (WG), we prove Theorem 3.4 in the univariable case. The formal proof basically follows the outline of its informal version [2, p. 102–103], which is tedious due to large amounts of case analyses on each CCS operator.

Theorem 4.20 (STRONG_UNIQUE_SOLUTION, univariable version of Theorem 3.4)**.** *Suppose the expression E contains at most the variable X, and let X be weakly guarded in E.*

$$\textit{If } P \sim E\{P/X\} \textit{ and } Q \sim E\{Q/X\} \textit{ then } P \sim Q. \tag{3}$$

$$\vdash \texttt{WG}\ E \wedge P \sim E\ P \wedge Q \sim E\ Q \Longrightarrow P \sim Q$$

4.10.2. The version for rooted bisimilarity

For the proof of Theorem 3.5, we did not use any "bisimulation up-to" technique. Instead, we have used a different technique based on Lemma 2.3, whose formal version is the following one (OBS_CONGR_BY_WEAK_BISIM):

$$\vdash \texttt{WEAK\_BISIM}\ \mathit{Wbsm} \Longrightarrow$$
$$\forall E\ E'.$$
$$(\forall u.$$
$$(\forall E_1.\ E\ -u\rightarrow\ E_1 \Longrightarrow \exists E_2.\ E' =u\Rightarrow E_2 \wedge \mathit{Wbsm}\ E_1\ E_2) \wedge$$
$$\forall E_2.\ E'\ -u\rightarrow\ E_2 \Longrightarrow \exists E_1.\ E =u\Rightarrow E_1 \wedge \mathit{Wbsm}\ E_1\ E_2) \Longrightarrow$$
$$E \approx^{c} E'$$

Using Lemma 2.3, the next two results are proved by directly constructing the required bisimulation: (see [26] for more details)

Lemma 4.21 (OBS_UNIQUE_SOLUTION_LEMMA)**.** *If the variable X is guarded and sequential in G, and $G\{P/X\} \xrightarrow{\alpha} P'$, then P' takes the form $H\{P/X\}$, for some H, and for any Q, we also have $G\{Q/X\} \xrightarrow{\alpha} H\{Q/X\}$. Moreover H is sequential, and if $\alpha = \tau$, then H is also guarded.*

$$\vdash \texttt{SG}\ G \wedge \texttt{SEQ}\ G \Longrightarrow$$
$$\forall P\ a\ P'.$$
$$G\ P\ -a\rightarrow\ P' \Longrightarrow$$
$$\exists H.\ \texttt{SEQ}\ H \wedge (a = \tau \Longrightarrow \texttt{SG}\ H) \wedge P' = H\ P \wedge \forall Q.\ G\ Q\ -a\rightarrow\ H\ Q$$

Theorem 4.22 (OBS_UNIQUE_SOLUTION, univariable version of Theorem 3.5)**.** *Let E be guarded and sequential expressions, and let $P \approx^c E\{P/X\}$, $Q \approx^c E\{Q/X\}$. Then $P \approx^c Q$.*

$$\vdash \texttt{SG}\ E \wedge \texttt{SEQ}\ E \wedge P \approx^c E\ P \wedge Q \approx^c E\ Q \Longrightarrow P \approx^c Q$$

4.10.3. The version for weak bisimilarity

Milner [2] only mentioned two "unique solution of equations" theorems, one for strong equivalence, the other for rooted bisimilarity. There is, however, another version for weak bisimilarity (Theorem 3.6) that shares a large portion of proof steps with the proof for rooted bisimilarity. As weak bisimilarity is not a congruence, we have to be more restrictive on the syntax of the equation, using only guarded sums. The related formal proofs are tedious but closely follow the informal proof [2, p. 158–159], with the exception of the use of "bisimulation up to with weak arrows" (Def. 4.6, instead of Def. 4.4).

Lemma 4.23 (WEAK_UNIQUE_SOLUTION_LEMMA)**.** *If the variable X is guarded and sequential (with only guarded sums) in G, and $G\{P/X\} \xrightarrow{\alpha} P'$, then P' takes the form $H\{P/X\}$, for some expression H, and for any Q, we have $G\{Q/X\} \xrightarrow{\alpha} H\{Q/X\}$. Moreover H is sequential, and if $\alpha = \tau$ then H is also guarded.*

$$\vdash \texttt{SG}\ G \wedge \texttt{GSEQ}\ G \Longrightarrow$$
$$\forall P\ a\ P'.$$
$$G\ P\ -a\rightarrow\ P' \Longrightarrow$$
$$\exists H.\ \texttt{GSEQ}\ H \wedge (a = \tau \Longrightarrow \texttt{SG}\ H) \wedge P' = H\ P \wedge \forall Q.\ G\ Q\ -a\rightarrow\ H\ Q$$

Theorem 4.24 (WEAK_UNIQUE_SOLUTION, univariable version of Theorem 3.6)**.** *Let E be guarded and sequential expressions, and let $P \approx E\{P/X\}$, $Q \approx E\{Q/X\}$. Then $P \approx Q$.*

$$\vdash \texttt{SG}\ E \wedge \texttt{GSEQ}\ E \wedge P \approx E\ P \wedge Q \approx E\ Q \Longrightarrow P \approx Q$$

4.11. Unique solution of contractions

A delicate point in the formalisation of the results about unique solution of contractions is the proof of Lemma 3.14 and lemmas alike. In particular, we use 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*, i.e., sequences of transitions. (A further motivation is to set the ground for future extensions of this formalisation work to *trace equivalence* in place of bisimilarity.)

A trace is formally 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). Then, given any labeled transition relation R (of type "$\alpha \rightarrow \beta \rightarrow \alpha \rightarrow$ bool"), LRTC R is a relation representing traces over R (of type "$\alpha \rightarrow \beta$ list $\rightarrow \alpha \rightarrow$ bool"):

$$\texttt{LRTC}\ R\ a\ l\ b \stackrel{\text{def}}{=}$$
$$\forall P.$$
$$(\forall x.\ P\ x\ []\ x) \wedge$$
$$(\forall x\ h\ y\ t\ z.\ R\ x\ h\ y \wedge P\ y\ t\ z \Longrightarrow P\ x\ (h::t)\ z) \Longrightarrow$$
$$P\ a\ l\ b \qquad \texttt{[LRTC\_DEF]}$$

The trace relation for CCS, TRACE (of the type "(α, β) CCS $\rightarrow \beta$ Action list $\rightarrow$ (α, β) CCS $\rightarrow$ bool"), is then obtained by combining LRTC and the TRANS ($\xrightarrow{\mu}$) relation defining the SOS rules:

$$\texttt{TRACE} \stackrel{\text{def}}{=} \texttt{LRTC TRANS} \qquad \texttt{[TRACE\_def]}$$

In a trace $P \xrightarrow{acts} Q$, the list of actions $acts$ may be empty, in which case there is no transition, and the initial and final processes are the same, i.e. $P = Q$. If there is at most one visible action μ in $acts$, and all other actions are τ's, then the trace represents a weak transition $P \stackrel{\mu}{\Longrightarrow} Q$. For this, we have to distinguish two cases in the list of actions: no label and unique label. The definition of "no label" for a list of actions is as follows, where MEM tests if an element is a member of a list:

$$\texttt{NO\_LABEL}\ L \stackrel{\text{def}}{=} \neg\exists\, l.\ \texttt{MEM}\ (\texttt{label}\ l)\ L \qquad \texttt{[NO\_LABEL\_def]}$$

The definition of "unique label" can be done in many ways. The following definition (due to a suggestion from Robert Beers) avoids any counting or filtering in the list. It says that a label is unique in a list of actions if there is no label in the rest of list (here $+\!\!+$ is the concatenation of lists):

$$\texttt{UNIQUE\_LABEL}\ u\ L \stackrel{\text{def}}{=} \exists\, L_1\ L_2.\ L_1 +\!\!+ [u] +\!\!+ L_2 = L \wedge \texttt{NO\_LABEL}\ L_1 \wedge \texttt{NO\_LABEL}\ L_2 \qquad \texttt{[UNIQUE\_LABEL\_def]}$$

Finally, the relationship between traces and weak transitions is stated and proved in the following theorem. It says that a weak transition $P \stackrel{u}{\Longrightarrow} P'$ 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 $u \neq \tau$):

$$\begin{array}{l} \vdash\ P\ {=}u{\Rightarrow}\ P' \iff \\ \quad \exists\, acts. \\ \qquad \texttt{TRACE}\ P\ acts\ P' \wedge \neg\texttt{NULL}\ acts \wedge \\ \qquad \textbf{if}\ u = \tau\ \textbf{then}\ \texttt{NO\_LABEL}\ acts\ \textbf{else}\ \texttt{UNIQUE\_LABEL}\ u\ acts \end{array} \qquad \texttt{[WEAK\_TRANS\_AND\_TRACE]}$$

Now the formal version of Lemma 3.10 (UNIQUE_SOLUTION_OF_CONTRACTIONS_LEMMA):

$$\begin{array}{l} \vdash\ (\exists E.\ \texttt{WGS}\ E \wedge P \succeq_{\text{bis}} E\ P \wedge Q \succeq_{\text{bis}} E\ Q) \Longrightarrow \\ \quad \forall C. \\ \qquad \texttt{GCONTEXT}\ C \Longrightarrow \\ \qquad (\forall l\ R. \\ \qquad\quad C\ P\ {=}\texttt{label}\ l{\Rightarrow}\ R \Longrightarrow \\ \qquad\quad \exists C'. \\ \qquad\qquad \texttt{GCONTEXT}\ C' \wedge R \succeq_{\text{bis}} C'\ P \wedge \\ \qquad\qquad (\texttt{WEAK\_EQUIV} \circ_r (\lambda x\ y.\ x\ {=}\texttt{label}\ l{\Rightarrow}\ y))\ (C\ Q)\ (C'\ Q)) \wedge \\ \qquad \forall R. \\ \qquad\quad C\ P\ {=}\tau{\Rightarrow}\ R \Longrightarrow \\ \qquad\quad \exists C'.\ \texttt{GCONTEXT}\ C' \wedge R \succeq_{\text{bis}} C'\ P \wedge (\texttt{WEAK\_EQUIV} \circ_r \texttt{EPS})\ (C\ Q)\ (C'\ Q) \end{array}$$

Traces are actually used in the proof of above lemma via the following lemma (unfolding_lemma4):

$$\begin{array}{l} \vdash\ \texttt{GCONTEXT}\ C \wedge \texttt{WGS}\ E \wedge \texttt{TRACE}\ ((C \circ \texttt{FUNPOW}\ E\ n)\ P)\ xs\ P' \wedge \texttt{LENGTH}\ xs \leq n \Longrightarrow \\ \quad \exists C'.\ \texttt{GCONTEXT}\ C' \wedge P' = C'\ P \wedge \forall Q.\ \texttt{TRACE}\ ((C \circ \texttt{FUNPOW}\ E\ n)\ Q)\ xs\ (C'\ Q) \end{array}$$

which roughly says that, for any context C and weakly guarded context E, if $C[\,E^n[P]\,] \xrightarrow{xs} P'$ and the length of actions $xs \leqslant n$, then P' has the form $C'[P]$. Traces are used in reasoning about the number of intermediate actions in weak transitions. For instance, from Def. 3.7, it is easy to see that a weak transition either becomes shorter or remains the same when moving between $\succeq_{\text{bis}}$-related processes. This property is essential in the proof of Lemma 3.10. We show only one such lemma, for the case of visible weak transitions:

$$\begin{array}{l} \vdash\ P \succeq_{\text{bis}} Q \Longrightarrow \\ \quad \forall xs\ l\ P'. \\ \qquad \texttt{TRACE}\ P\ xs\ P' \wedge \texttt{UNIQUE\_LABEL}\ (\texttt{label}\ l)\ xs \Longrightarrow \\ \qquad \exists xs'\ Q'. \\ \qquad\quad \texttt{TRACE}\ Q\ xs'\ Q' \wedge P \succeq_{\text{bis}} Q \wedge \texttt{LENGTH}\ xs' \leq \texttt{LENGTH}\ xs \wedge \\ \qquad\quad \texttt{UNIQUE\_LABEL}\ (\texttt{label}\ l)\ xs' \end{array} \qquad \texttt{[contracts\_AND\_TRACE\_label]}$$

With all above lemmas, we can thus finally prove Theorem 3.11:

$$\vdash\ \texttt{WGS}\ E \wedge P \succeq_{\text{bis}} E\ P \wedge Q \succeq_{\text{bis}} E\ Q \Longrightarrow P \approx Q \qquad \texttt{[UNIQUE\_SOLUTION\_OF\_CONTRACTIONS]}$$

4.12. Unique solution of rooted contractions

The formal proof of "unique solution of rooted contractions theorem" (Theorem 3.16, univariable version) shares the same initial proof steps as Theorem 3.11. It then requires a few more steps to handle rooted bisimilarity in the conclusion. The two proofs are similar, mostly because the main property needed from contraction and rooted contraction is precongruence. Below is the formal version of Theorem 3.16:

$\vdash$ `WG` $E \wedge P \succeq^{c}_{bis} E\ P \wedge Q \succeq^{c}_{bis} E\ Q \Longrightarrow P \approx^{c} Q$ `[UNIQUE_SOLUTION_OF_ROOTED_CONTRACTIONS]`

Having proved the precongruence property of rooted contraction ($\succeq^{c}_{bis}$), now we can use weakly guarded expressions in the above theorem, which has no more constraints on summation (as shown by the appearance of `WG` rather than `WGS`).

Having removed the constraints on summation, this result is close to Milner's original 'unique solution of equations' theorem for *strong* bisimilarity (Theorem 3.4) — the same weakly guarded context (`WG`) is required. In contrast, Milner's "unique solution of equations" theorem for rooted bisimilarity ($\approx^{c}$) (Theorem 3.5), has more rigid constraints, as the *equations* must be both guarded and sequential.

5. The multivariable formalisation

In this section we attack the case of multiple equations (the 'multivariable' case), for two major theorems earlier discussed: Milner's "unique solution of equations (for $\sim$)" (Theorem 3.4) and the "unique solution of rooted contractions" (Theorem 3.16). The formalisation supports finitely many equations/contractions and equation variables.[4] We chose these two theorems because of their relevance, and because they well illustrate the work needed in the multivariable case.

The central problem of the multivariable formalisation is the representation of multivariable CCS equations (expressions and contexts). In the univariable case, λ-functions are used for representing univariable CCS equations, and variable substitutions are simply applications of λ-functions to CCS terms. This idea, however, cannot be extended to the multivariable case, as we do not have a fixed number of variables to deal with.

In the literature ([9, p. 102], e.g.), the variables X_i of a system of equations $\{X_i = E_i\}_{i \in I}$ are usually considered as *process variables* outside the CCS syntax. Then, an equation body E_i is an *open* expression built with CCS operators plus these equation variables. In a formalisation, this means either defining a whole new datatype (with the new equation variables), in which each CCS operator must be duplicated, or adding equation variables as a new primitive to the existing CCS type. In either cases the *disjointness* between equation variables and agent variables is syntactically guaranteed. Both solutions are rather combersome, and require a non-trivial modification of the existing univariable formalisation.

In our work, we have followed Milner's original approach [10], using the *same* alphabet for both agent variables and equation variables. This approach allows a large reuse of the univariable formalisation. For instance, the variable substitution in the SOS rule `REC` can be reused for substituting equation variables. However, care is needed in the proofs of many fundamental lemmas, mostly due to variable capture issues between free equation variables and bound agent variables.

5.1. Free and bound variables

As mentioned at the beginning of Section 3, within our CCS syntax an agent variable may occur outside the recursion in which it is bound. Thus the same variable can appear both *free* and *bound* in a CCS term. We denote the set of bound variables of a given CCS term E (the variables that are bound in a recursion subexpression of E) as $\mathsf{bv}(E)$ (or `BV` E in HOL), and the set of free variables as $\mathsf{fv}(E)$ (or `FV` E in HOL). Both `BV` and `FV` have the type "(α, β) `CCS` $\rightarrow$ α $\rightarrow$ `bool`", i.e. functions taking CCS terms and returning (finite) sets of variables (of type α). For their definition the interesting cases are those of recursion and agent variables, shown below (here `DELETE` and `INSERT` are set-theoretic operators of HOL's `pred_set` theory):

[4] The original theorems hold for even infinitely many equations and equation variables.

$\texttt{FV}\ (\texttt{var}\ X) \stackrel{\text{def}}{=} \{X\}$	$\texttt{FV}\ (\texttt{rec}\ X\ p) \stackrel{\text{def}}{=} \texttt{FV}\ p\ \texttt{DELETE}\ X$
$\texttt{BV}\ (\texttt{var}\ X) \stackrel{\text{def}}{=} \emptyset$	$\texttt{BV}\ (\texttt{rec}\ X\ p) \stackrel{\text{def}}{=} X\ \texttt{INSERT BV}\ p$

Furthermore, E is a process, written $\texttt{IS\_PROC}\ E$, if it does not contain any free variable, i.e. $\mathsf{fv}(E) = \emptyset$:

$$\texttt{IS\_PROC}\ E \stackrel{\text{def}}{=} \texttt{FV}\ E = \emptyset \qquad \texttt{[IS\_PROC\_def]}$$

And a list of CCS processes can be asserted by ALL_PROC defined upon IS_PROC:

$$\texttt{ALL\_PROC}\ Es \stackrel{\text{def}}{=} \texttt{EVERY IS\_PROC}\ Es \qquad \texttt{[ALL\_PROC\_def]}$$

The set of free and bound variables of a term need not be disjoint, e.g. in $X + \texttt{rec}X.E$. More importantly, when going from a CCS expression to its sub-expressions, the set of free variables may increase, while the set of bound variables either remains the same or decreases. For instance, $\mathsf{fv}(\texttt{rec}\ X.(\mu.X)) = \emptyset$, while $\mathsf{fv}(\mu.X) = \{X\}$. This property of $\mathsf{fv}(\cdot)$ brings some difficulities in transition inductions. As an evidence, we prove the following fundamental property of $\mathsf{fv}(\cdot)$ [10, p. 1209]:

Proposition 5.1. *The derivatives of a process are themselves processes, i.e. if* $E \xrightarrow{\mu} E'$ *and* $\mathsf{fv}(E) = \emptyset$, *then* $\mathsf{fv}(E') = \emptyset$. *Formally:*

$$\vdash \forall E\ u\ E'.\ E\ -u\rightarrow\ E' \wedge \texttt{IS\_PROC}\ E \Longrightarrow \texttt{IS\_PROC}\ E' \qquad \texttt{[TRANS\_PROC]}$$

Proof. We prove a stronger result, asserting that the set of free variables in a CCS process does not increase in its derivatives:

$$\vdash \forall E\ u\ E'.\ E\ -u\rightarrow\ E' \Longrightarrow \texttt{FV}\ E' \subseteq \texttt{FV}\ E \qquad \texttt{[TRANS\_FV]}$$

In [10], the proof of the above property is commented as "an easy action induction". As a matter of fact, in the HOL proof "action induction" (also known as *transition induction*) becomes a form of higher-order application of the following *induction principle* (generated together with the SOS rules), which essentially says that TRANS is the smallest relation satisfying the SOS rules:

$$\begin{array}{l}
\vdash \forall P. \\
\quad (\forall E\ u.\ P\ (u.E)\ u\ E) \wedge (\forall E\ u\ E_1\ E'.\ P\ E\ u\ E_1 \Longrightarrow P\ (E + E')\ u\ E_1) \wedge \\
\quad (\forall E\ u\ E_1\ E'.\ P\ E\ u\ E_1 \Longrightarrow P\ (E' + E)\ u\ E_1) \wedge \\
\quad (\forall E\ u\ E_1\ E'.\ P\ E\ u\ E_1 \Longrightarrow P\ (E \mid E')\ u\ (E_1 \mid E')) \wedge \\
\quad (\forall E\ u\ E_1\ E'.\ P\ E\ u\ E_1 \Longrightarrow P\ (E' \mid E)\ u\ (E' \mid E_1)) \wedge \\
\quad (\forall E\ l\ E_1\ E'\ E_2. \\
\qquad P\ E\ (\texttt{label}\ l)\ E_1 \wedge P\ E'\ (\texttt{label}\ (\texttt{COMPL}\ l))\ E_2 \Longrightarrow \\
\qquad P\ (E \mid E')\ \tau\ (E_1 \mid E_2)) \wedge \\
\quad (\forall E\ u\ E'\ l\ L. \\
\qquad P\ E\ u\ E' \wedge (u = \tau \vee u = \texttt{label}\ l \wedge l \notin L \wedge \texttt{COMPL}\ l \notin L) \Longrightarrow \\
\qquad P\ ((\nu L)\ E)\ u\ ((\nu L)\ E')) \wedge \\
\quad (\forall E\ u\ E'\ rf.\ P\ E\ u\ E' \Longrightarrow P\ (\texttt{relab}\ E\ rf)\ (\texttt{relabel}\ rf\ u)\ (\texttt{relab}\ E'\ rf)) \wedge \\
\quad (\forall E\ u\ X\ E_1.\ P\ ([\texttt{rec}\ X\ E/X]\ E)\ u\ E_1 \Longrightarrow P\ (\texttt{rec}\ X\ E)\ u\ E_1) \Longrightarrow \\
\quad \forall a_0\ a_1\ a_2.\ a_0\ -a_1\rightarrow\ a_2 \Longrightarrow P\ a_0\ a_1\ a_2 \qquad \texttt{[TRANS_ind]}
\end{array}$$

The above long theorem is of the form $\forall P.\ X \Longrightarrow \forall E\ u\ E'.\ E\ -u\rightarrow\ E' \Longrightarrow P\ E\ u\ E'$ (with a_0, a_1, a_2 renamed), where the outermost universal quantifier P is a higher-order proposition (taking E, μ and E'), and X is another higher-order proposition. The proof goal is actually in the form of $\forall E\ u\ E'.\ E\ -u\rightarrow\ E' \Longrightarrow P\ E\ u\ E'$, where $P = (\lambda E\ u\ E'.\ \texttt{FV}\ E' \subseteq \texttt{FV}\ E)$. Thus, if we can prove X under this specific P, by Modus Ponens (MP) the original proof is completed. Now the goal can be reduced to several conjunct subgoals, each corresponding to one SOS rule. For instance, in the subgoal for SUM1 we need to prove $\mathsf{fv}(E_1) \subseteq \mathsf{fv}(E) \Longrightarrow \mathsf{fv}(E_1) \subseteq \mathsf{fv}(E + E')$, which holds as $\mathsf{fv}(E) \subseteq \mathsf{fv}(E + E') = \mathsf{fv}(E) \cup \mathsf{fv}(E')$. Eventually we have only the following goal left (the term above the dash line is the goal, those below the line are assumptions):

```
    FV E' ⊆ FV E DELETE X
------------------------------------
 0. FV E' ⊆ FV ([rec X E/X] E)
```

Note that FV E DELETE X = FV (rec X E), thus the current proof goal is indeed the consequence of action induction on the recursion operator. Here the problem is that we know nothing about FV ([rec X E/X] E). To further proceed, we need to prove some other (easier) lemmas. First, the next lemma can be proven by induction on E and a few basic set-theoretic facts:

$\vdash \forall X\ E\ E'.$ FV ($[E'/X]$ E) $\subseteq$ FV $E \cup$ FV E' [FV_SUBSET]

Now, if we take E' = rec $X.E$ in the above lemma, we get FV ([rec X E/X] E) $\subseteq$ FV $E \cup$ FV (rec X E) = FV $E \cup$ (FV E DELETE X) = FV E, i.e. the following lemma:

$\vdash \forall X\ E.$ FV ([rec X E/X] E) $\subseteq$ FV E [FV_SUBSET_REC]

Thus we can enrich the assumptions of the current proof goal with above lemma, and obtain FV E' $\subseteq$ FV E by the transitivity of $\subseteq$:

```
    FV E' ⊆ FV E DELETE X
------------------------------------
 0. FV E' ⊆ FV ([rec X E/X] E)
 1. FV ([rec X E/X] E) ⊆ FV E
 2. FV E' ⊆ FV E
```

Knowing FV E' $\subseteq$ FV E we cannot prove FV E' $\subseteq$ FV E DELETE X. However, if we knew $X \notin$ FV E', then FV E' DELETE X = FV E', and then FV E' $\subseteq$ FV $E \implies$ FV E' DELETE $X \subseteq$ FV E DELETE X, no matter if $X \in$ FV E or not, and the proof would complete. Thus it remains to show that

```
    X ∉ FV E'
------------------------------------
 0. FV E' ⊆ FV ([rec X E/X] E)
 1. FV ([rec X E/X] E) ⊆ FV E
 2. FV E' ⊆ FV E
```

Now we try the proof by contradiction (*reductio ad absurdum*): if the goal does not hold, i.e. $X \in$ FV E', then by assumption 0 we have $X \in$ FV ([rec X E/X] E):

```
    F
------------------------------------
 0. FV E' ⊆ FV ([rec X E/X] E)
 1. FV ([rec X E/X] E) ⊆ FV E
 2. FV E' ⊆ FV E
 3. X ∈ FV E'
 4. X ∈ FV ([rec X E/X] E)
```

But this is impossible, because all free occurrences of X in E now become bound in the form of rec $X.E$. In fact, the following lemma can be proven by induction on E:

$\vdash \forall X\ E\ E'.\ X \notin$ FV ([rec X E'/X] E) [NOTIN_FV_lemma]

Adding the above lemma (taking $E' = E$) into the assumption list immediately causes a contradiction with assumption 4, and the proof finally completes. □

Proposition 5.1 is essential in the proofs of the multivariable versions of all unique-solution theorems. On the other hand, the analogous result for bound variables is indeed just a simple transition induction. (The easy proof is omitted.)

Proposition 5.2. *if* $E \xrightarrow{\mu} E'$ *then* $\mathsf{bv}(E') \subseteq \mathsf{bv}(E)$*, or formally:*

$\vdash \forall E\ u\ E'.\ E\ -u\rightarrow\ E' \Longrightarrow \texttt{BV}\ E' \subseteq \texttt{BV}\ E$ [TRANS_BV]

5.2. Multivariable substitutions

There are two natural ways to implement the multivariable substitution $E\{\widetilde{P}/\widetilde{X}\}$, which replaces each free occurrence of the variables X_i in E with the corresponding P_i: (1) iterately applying the existing univariable `CCS_Subst`; (2) defining a new multivariable substitution function which substitutes all variables $\widetilde{X}$ in parallel. Note that there is the possibility that $\widetilde{P}$ syntactically again contains variables from $\widetilde{X}$, thus different orders of iterated substitutions may lead to different results. We thus prefer to define a multivariable version of `CCS_Subst` called `CCS_SUBST`, based on HOL `finite_map` theory [22].

A finite map (of type $\alpha \mapsto \beta$) is like a function (of type $\alpha \rightarrow \beta$) having only finitely many elements in its domain. In HOL, an empty finite map is denoted as `FEMPTY`. If *fm* is a finite map, its domain (as a set of keys) is denoted as `FDOM` *fm*. Applying *fm* on a certain key, say *k*, is denoted by *fm* ' *k*.

The function `CCS_SUBST` takes a finite map *fm* (of type $\alpha \mapsto (\alpha, \beta)$ `CCS`) and a CCS expression, and returns another CCS expression in which all occurrences of variables in the finite domain of *fm* are substituted with the corresponding value in *fm*. Initially, such a finite map can be built from a list of variables *Xs* and the corresponding substituted terms *Ps* by a helper function `fromList` (whose details are omitted here). `CCS_SUBST` (`fromList` *Xs Ps*) *E* is usually abbreviated as $[Ps/Xs]\ E$. Using finite maps, the substitution mechanism is order-independent. For most CCS operators, `CCS_SUBST` recursively calls itself on subterms. The most interesting cases are at agent variables and recursion:

$$\texttt{CCS\_SUBST}\ fm\ (\texttt{var}\ X) \stackrel{\text{def}}{=} \textbf{if}\ X \in \texttt{FDOM}\ fm\ \textbf{then}\ fm\ \texttt{'}\ X\ \textbf{else}\ \texttt{var}\ X \qquad \texttt{[CCS\_SUBST\_var]}$$

$$\texttt{CCS\_SUBST}\ fm\ (\texttt{rec}\ X\ E) \stackrel{\text{def}}{=} \textbf{if}\ X \in \texttt{FDOM}\ fm\ \textbf{then}\ \texttt{rec}\ X\ (\texttt{CCS\_SUBST}\ (fm \setminus X)\ E)\ \textbf{else}\ \texttt{rec}\ X\ (\texttt{CCS\_SUBST}\ fm\ E) \qquad \texttt{[CCS\_SUBST\_rec]}$$

Variable substitution only occurs on free variables. When the term *E* of "`CCS_SUBST` *fm E*" is in the form of $\texttt{rec}\ X.\ E'$, if *X* (a bound variable) is in the domain of *fm*, `CCS_SUBST` must continue the substitution on E' using a reduced map — without *X* — so that all occurences of *X* in E' are correctly bypassed. Since `CCS_SUBST` only runs once on each subterm, the possible free variables in *Ps* are never substituted.

Below we present some key lemmas about `CCS_SUBST`, omitting the proofs:[5]

Lemma 5.3 (`CCS_SUBST_elim`)**.** *If the free variables of E is disjoint with Xs, a substitution of Xs in E does not change E:*

$\vdash \texttt{DISJOINT}\ (\texttt{FV}\ E)\ (\texttt{set}\ Xs) \wedge \texttt{LENGTH}\ Ps = \texttt{LENGTH}\ Xs \Longrightarrow [Ps/Xs]\ E = E$

Lemma 5.4 (`CCS_SUBST_self`)**.** *Substituting each free variable X in E with* `var` *X (itself) does not change E:*

$\vdash \texttt{ALL\_DISTINCT}\ Xs \Longrightarrow [\texttt{MAP var}\ Xs/Xs]\ E = E$

The next lemma plays an important role. It essentially swaps the order of two substitutions:

Lemma 5.5 (`CCS_SUBST_nested`)**.** *Under certain conditions (to get rid of substitution orders), two nested substitutions can be converted into a single substitution where the targets are substituted first:*

$$\vdash \texttt{ALL\_DISTINCT}\ Xs \wedge \texttt{LENGTH}\ Ps = \texttt{LENGTH}\ Xs \wedge \texttt{LENGTH}\ Es = \texttt{LENGTH}\ Xs \wedge \texttt{DISJOINT}\ (\texttt{BV}\ C)\ (\texttt{set}\ Xs) \Longrightarrow [Ps/Xs]\ ([Es/Xs]\ C) = [\texttt{MAP}\ [Ps/Xs]\ Es/Xs]\ C$$

[5] Hereafter, some new logical constants from HOL's `pred_set` and `list` theories are used (see [22] for more details.): `DISJOINT` denotes set disjointness; "`set` *Xs*" is the set converted from the list *Xs*; `ALL_DISTINCT` says that the elements of a list are distinct; `MAP` is the mapping function from one list to another; `EVERY` means each element of a list satisfies a predicate; `ZIP` creates a new list from two lists of the same length, and each element of the new list is a pair of elements from the given lists.

The next three lemmas show the relative correctness of CCS_SUBST with respect to CCS_Subst:

Lemma 5.6 (CCS_SUBST_sing). *If there is only one single variable X in the map (with the target expression E'), then* CCS_SUBST *behaves exactly the same as (the univariable)* CCS_Subst*:*

$$\vdash [[E']/[X]]\ E\ =\ [E'/X]\ E$$

Lemma 5.7 (CCS_SUBST_reduce). *Under certain conditions (to get rid of substitution orders), a multivariable substitution of variables $X::Xs$ can be reduced to a multivariable substitution of variables Xs and an univariable substitution of X:*

$$\begin{array}{l}\vdash \neg\texttt{MEM}\ X\ Xs\ \wedge\ \texttt{ALL\_DISTINCT}\ Xs\ \wedge\ \texttt{LENGTH}\ Ps\ =\ \texttt{LENGTH}\ Xs\ \wedge\\ \quad \texttt{EVERY}\ (\lambda\, e.\ X\notin \texttt{FV}\ e)\ Ps\implies\\ \quad \forall E.\ [P::Ps/X::Xs]\ E\ =\ [P/X]\ ([Ps/Xs]\ E)\end{array}$$

Lemma 5.8 (CCS_SUBST_FOLDR). *Under certain conditions (to get rid of substitution orders), a multivariable substitution can be reduced to repeated applications of univariable substitutions of each variable:*

$$\begin{array}{l}\vdash \texttt{ALL\_DISTINCT}\ Xs\ \wedge\ \texttt{LENGTH}\ Ps\ =\ \texttt{LENGTH}\ Xs\ \wedge\\ \quad \texttt{EVERY}\ (\lambda(x,p).\ \texttt{FV}\ p\subseteq\{x\})\ (\texttt{ZIP}\ (Xs,Ps))\implies\\ \quad [Ps/Xs]\ E\ =\ \texttt{FOLDR}\ (\lambda(x,y)\ e.\ [y/x]\ e)\ E\ (\texttt{ZIP}\ (Xs,Ps))\end{array}$$

Finally, the following two lemmas precisely estimate the free and bound variables of a substituted term:

Lemma 5.9 (BV_SUBSET_BIGUNION). *The bound variables of $[Ps/Xs]\ E$ are contained in the union of the bound variables in E and Ps.*

$$\begin{array}{l}\vdash \texttt{ALL\_DISTINCT}\ Xs\ \wedge\ \texttt{LENGTH}\ Ps\ =\ \texttt{LENGTH}\ Xs\ \wedge\ \texttt{DISJOINT}\ (\texttt{BV}\ E)\ (\texttt{set}\ Xs)\implies\\ \texttt{BV}\ ([Ps/Xs]\ E)\subseteq \texttt{BV}\ E\cup \texttt{BIGUNION}\ (\texttt{IMAGE BV}\ (\texttt{set}\ Ps))\end{array}$$

Lemma 5.10 (FV_SUBSET_BIGUNION_PRO). *The free variables of $[Ps/Xs]\ E$ are contained in the union of the free variables in E and Ps, excluding Xs.*

$$\begin{array}{l}\vdash \texttt{ALL\_DISTINCT}\ Xs\ \wedge\ \texttt{LENGTH}\ Ps\ =\ \texttt{LENGTH}\ Xs\ \wedge\ \texttt{DISJOINT}\ (\texttt{BV}\ E)\ (\texttt{set}\ Xs)\implies\\ \texttt{FV}\ ([Ps/Xs]\ E)\subseteq \texttt{FV}\ E\ \texttt{DIFF set}\ Xs\cup \texttt{BIGUNION}\ (\texttt{IMAGE FV}\ (\texttt{set}\ Ps))\end{array}$$

In some of the above lemmas, the condition DISJOINT (BV E) (set Xs) (the bound variables of E and the free substitution variables are disjoint) is added to ease the proofs. The condition is not necessary and could be eliminated, at the price of some extra effort.

5.3. Multivariable (weakly guarded) contexts

As mentioned in Section 4.7, univariable contexts and their guarded companions are defined as predicates over λ-expressions of type "$(\alpha,\ \beta)$ CCS $\to$ $(\alpha,\ \beta)$ CCS". These predicates, such as CONTEXT (multi-hole contexts), WG (weak guarded contexts), SG (guarded contexts) and SEQ (sequential contexts), are all defined inductively, i.e. they are built from some "holes" in a bottom-up manner. For instance, WG $(\lambda\, t.\ a.t \mid P)$ holds because WG $(\lambda\, t.\ a.t)$ holds as a base case of the inductive definition of WG:

$$\vdash \forall a.\ \texttt{WG}\ (\lambda\, t.\ a.t) \qquad \texttt{[WG1]}$$

For any agent variable X, we have by β-reduction: $(\lambda\, t.\ l.t + P)\ (\texttt{var}\ X) = l.\texttt{var}\ X + P$ (or "$l.\,X + P$" in textbook notation), and the expression is weakly guarded (Def. 3.3).

It could be possible to inductively define multivariable contexts and their guarded variants, but care is needed for variables that occur within a recusion. We have preferred a different solution, in which the definitions of multivariable contexts is based on the existing univariable definitions. For this, intuitively, we replace a single variable with a hole (viewing the other variables as constants), so to obtain a single-variable context on which predicates like CONTEXT can be used. For example, to see the weak guardedness of $a.\,X + b.\,X + c.\,Y$, we substitute each occurrence of a variable with a "hole" and consider the resulting term as a multi-hole context (i.e., a λ-function); then all such multi-hole contexts should satisfy WG, i.e. the following results hold:

$\vdash$ WG $(\lambda\, t.\ a.t\ +\ b.t\ +\ c.\mathtt{var}\ Y)$
$\vdash$ WG $(\lambda\, t.\ a.\mathtt{var}\ X\ +\ b.\mathtt{var}\ X\ +\ c.t)$

Note that $a.t + b.t + c.\mathtt{var}\ Y = [t/X]\ (a.\mathtt{var}\ X + b.\mathtt{var}\ X + c.\mathtt{var}\ Y)$ can be expressed as a univariable substitution of the original term. This idea leads to the following definitions:

$$\mathtt{context}\ Xs\ E \stackrel{\mathsf{def}}{=} \mathtt{EVERY}\ (\lambda\, X.\ \mathtt{CONTEXT}\ (\lambda\, t.\ [t/X]\ E))\ Xs \qquad \texttt{[context\_def]}$$

$$\mathtt{weakly\_guarded}\ Xs\ E \stackrel{\mathsf{def}}{=} \mathtt{EVERY}\ (\lambda\, X.\ \mathtt{WG}\ (\lambda\, t.\ [t/X]\ E))\ Xs \qquad \texttt{[weakly\_guarded\_def]}$$

The above definitions take an extra list of variables Xs and assert the CCS expression E with respect to this list. This allows us to formalise the concepts of contexts and guardedness independently of the free variables of E. Then a logical term "`weakly_guarded` (`SET_TO_LIST` (`FV` E)) E" can be used to assert the weak guardedness of E with respect to all its free variables.[6] (The multivariable guardedness and sequentiality can also be defined similarily, using their univariable companions `SG` and `SEQ`.)

The most important property of multivariable contexts is that strong bisimilarity and other (pre)congruence relations are preserved by them, for instance:

Lemma 5.11 (`STRONG_EQUIV_subst_context`)**.** *If two tuples of processes $\widetilde{P}$ and $\widetilde{Q}$ are strongly bisimilar*[7]*, then for any context E, $E\{\widetilde{P}/\widetilde{X}\}$ and $E\{\widetilde{Q}/\widetilde{X}\}$ are strongly bisimilar:*

$\vdash$ ALL_DISTINCT $Xs \wedge$ LENGTH $Ps =$ LENGTH $Xs \wedge Ps \sim Qs \Longrightarrow$
$\forall\, E.$ context $Xs\ E \Longrightarrow [Ps/Xs]\ E \sim [Qs/Xs]\ E$

Similar properties also hold if $\sim$ is replaced with rooted bisimilarity ($\approx^{\mathrm{c}}$) and rooted contraction ($\succeq^{\mathrm{c}}_{\mathrm{bis}}$). It does not hold for weak bisimilarity ($\approx$) and the contraction preorder ($\succeq_{\mathrm{bis}}$), as they are not (pre)congruence relations.

Another important property of contexts is their composability: if we substitute some free variables in a context with some other contexts, the resulting term is still a valid context (with respect to the same set of variables):

$\vdash$ ALL_DISTINCT $Xs \wedge$ context $Xs\ C \wedge$ EVERY (context Xs) $Es \wedge$
LENGTH $Es =$ LENGTH $Xs \Longrightarrow$
context Xs $([Es/Xs]\ C)$ [context_combin]

Not every term within the CCS syntax is a context, as context (or equation) variables are not allowed to occur within recursion. The converse holds however: if the set of free variables of a CCS term is disjoint from a list of variables, then the term is indeed a context with respect to such list of variables:

$\vdash$ DISJOINT (FV E) (set Xs) $\Longrightarrow$ context $Xs\ E$ [disjoint_imp_context]

On the other hand, for any context (with respect to $\widetilde{X}$) of the form $\mathtt{rec}\, Y.\, E$, the set of free variables of E excluding Y is disjoint with $\widetilde{X}$:

$\vdash$ context Xs (rec Y E) $\Longrightarrow$ DISJOINT (FV E DELETE Y) (set Xs) [context_rec]

In the above lemma we cannot conclude DISJOINT (FV E) (set Xs), because Y may appear in both sets. We also cannot conclude context $Xs\ E$, because in $\mathtt{rec}\, Y.\, E$ the bound variable Y may occur freely within another nested recursion. For instance, variable Y is free in $\mathtt{rec}\, Y.\, E$, for $E = \mathtt{rec}\, Z.\, (a.\, Y + b.\, Z) + c.\, Y$.

For weakly guarded contexts, besides their usual properties as in the univariable case (i.e., weak guardedness, `WG`), we also need their composability with respect to multivariable contexts: if we substitute some free variables in a context C with some weakly guarded contexts, the resulting context is weakly guarded (with respect to the same set of variables):

[6] Here `SET_TO_LIST` converts a finite set to a list of the same elements. It turns out that we never need this, because in all unique-solution theorems a list of variables Xs is fixed and then all equations are required to contain free variables in Xs.

[7] A HOL term like $Ps \sim Qs$ means that two lists of CCS processes are componentwise bisimilar ($P_i \sim Q_i$).

$\vdash$ `ALL_DISTINCT` $Xs \wedge$ `context` $Xs\ C \wedge$ `weakly_guarded` $Xs\ Es \wedge$
`LENGTH` $Es =$ `LENGTH` $Xs \Longrightarrow$
`weakly_guarded` $Xs\ ([Es/Xs]\ C)$ `[weakly_guarded_combin]`

5.4. Multivariable equations and solutions

With the formal definitions of multivariable substitution and multivariable (weakly guarded) contexts, now we are ready to formally define multivariable CCS equations/contractions and their (unique) solutions. Consider a system of equation $\{X_i = E_i\}_{i \in I}$ (Def. 3.1), or its expanded form (here we suppose $I = [1, n] \in \mathbb{N}$, i.e. a finite list):

$$\begin{cases} X_1 = E_1[\widetilde{X}] \\ X_2 = E_2[\widetilde{X}] \\ \cdots \\ X_n = E_n[\widetilde{X}] \end{cases}$$

Consider its two essential ingredients:

- $\widetilde{X} = (X_1, X_2, \ldots, X_n)$: a list of equation variables;
- $\widetilde{E} = (E_1, E_2, \ldots, E_n)$: a list of CCS contexts with possible occurrences of free variables in $\widetilde{X}$.

The two lists $\widetilde{X}$ and $\widetilde{E}$ should have the same length, and the variables in $\widetilde{X}$ should be distinct. Furthermore, $\widetilde{E}$ does not contain free variables that are not in $\widetilde{X}$. Finally, for each E_i, the set of its bound variables should be *disjoint* from $\widetilde{X}$. These requirements yield the fomal definition of multivariable CCS equation:

`CCS_equation` $Xs\ Es \stackrel{\text{def}}{=}$
`ALL_DISTINCT` $Xs \wedge$ `LENGTH` $Es =$ `LENGTH` $Xs \wedge$
`EVERY` $(\lambda e.$ `FV` $e \subseteq$ `set` $Xs)\ Es \wedge$ `EVERY` $(\lambda e.$ `DISJOINT` (`BV` e) (`set` Xs)) Es

Now consider (formally) what is a solution $\widetilde{P}$ of CCS equations. First of all, the definition should be parametrized on a binary CCS relation $\mathcal{R}$ such as $\sim$ and $\succeq^{\text{c}}_{\text{bis}}$, so that a single definition can be used to represent solutions of all kinds of CCS equations. Then, $\widetilde{P}\ \mathcal{R}\ \widetilde{E}\{\widetilde{P}/\widetilde{X}\}$ should hold:

$$\begin{cases} P_1\ \mathcal{R}\ E_1\{\widetilde{P}/\widetilde{X}\} \\ P_2\ \mathcal{R}\ E_2\{\widetilde{P}/\widetilde{X}\} \\ \cdots \\ P_n\ \mathcal{R}\ E_n\{\widetilde{P}/\widetilde{X}\} \end{cases}$$

Furthermore, each P_i should be a pure process, i.e. having no free variable. Finally, the set of bound variables is disjoint with $\widetilde{X}$. (This disjointness requirement is optional but makes many proofs easier.) Putting all together, below is the formal definition of a solution of multivariable CCS equations:[8]

`CCS_solution` $R\ Xs\ Es\ Ps \stackrel{\text{def}}{=}$
`ALL_PROC` $Ps \wedge$ `EVERY` $(\lambda e.$ `DISJOINT` (`BV` e) (`set` Xs)) $Ps \wedge$
`LIST_REL` $R\ Ps$ (`MAP` $[Ps/Xs]\ Es$)

Note that the two logical terms "`CCS_solution` $R\ Xs\ Es\ Ps$" and "$Ps \in$ `CCS_solution` $R\ Xs\ Es$" in HOL have the same meaning (they are equivalent). The latter form suggests that "`CCS_solution` $R\ Xs\ Es$" is actually a set containing all solutions of "`CCS_equation` $Xs\ Es$". Then the unique-solution theorems can be understood thus: any two elements in the solution set are bisimilar.

[8] If R is a binary relation of CCS processes, `LIST_REL` R is the same binary relation but for lists of CCS processes with the same length. implicitly implies that the two lists A and B have the same length.

5.5. Unique solution of equations/contractions (the multivariable version)

With all above machineries of multivariable contexts and substitutions, the multivariable case of Lemma 3.15 is formalised below (`strong_unique_solution_lemma`):

$$\begin{array}{l}\vdash \texttt{weakly\_guarded}\ Xs\ E\ \wedge\ \texttt{FV}\ E \subseteq \texttt{set}\ Xs\ \wedge\ \texttt{DISJOINT}\ (\texttt{BV}\ E)\ (\texttt{set}\ Xs) \Longrightarrow \\ \quad \forall\, Ps\,. \\ \qquad \texttt{LENGTH}\ Ps\ =\ \texttt{LENGTH}\ Xs \Longrightarrow \\ \qquad \forall\, u\ P'\,. \\ \qquad\quad [Ps/Xs]\ E\ -u\rightarrow\ P' \Longrightarrow \\ \qquad\quad \exists\, E'\,. \\ \qquad\qquad \texttt{context}\ Xs\ E'\ \wedge\ \texttt{FV}\ E' \subseteq \texttt{set}\ Xs\ \wedge\ \texttt{DISJOINT}\ (\texttt{BV}\ E')\ (\texttt{set}\ Xs)\ \wedge \\ \qquad\qquad P'\ =\ [Ps/Xs]\ E'\ \wedge \\ \qquad\qquad \forall\, Qs\,.\ \texttt{LENGTH}\ Qs\ =\ \texttt{LENGTH}\ Xs \Longrightarrow [Qs/Xs]\ E\ -u\rightarrow\ [Qs/Xs]\ E'\end{array}$$

The multivariable version of Theorem 3.4 is formalised thus:

$$\begin{array}{l}\vdash \texttt{CCS\_equation}\ Xs\ Es\ \wedge\ \texttt{weakly\_guarded}\ Xs\ Es\ \wedge \\ Ps\ \in\ \texttt{CCS\_solution STRONG\_EQUIV}\ Xs\ Es\ \wedge \\ Qs\ \in\ \texttt{CCS\_solution STRONG\_EQUIV}\ Xs\ Es \Longrightarrow \\ Ps\ \sim\ Qs \qquad\qquad \texttt{[strong\_unique\_solution\_thm]}\end{array}$$

Now the multivariable version of Theorem 3.16:

$$\begin{array}{l}\vdash \texttt{CCS\_equation}\ Xs\ Es\ \wedge\ \texttt{weakly\_guarded}\ Xs\ Es\ \wedge \\ Ps\ \in\ \texttt{CCS\_solution OBS\_contracts}\ Xs\ Es\ \wedge \\ Qs\ \in\ \texttt{CCS\_solution OBS\_contracts}\ Xs\ Es \Longrightarrow \\ Ps\ \approx^{\mathrm{c}}\ Qs \qquad\qquad \texttt{[unique\_solution\_of\_rooted\_contractions]}\end{array}$$

In summary, while each step related to multivariable substitutions is more difficult than that for the univariable case, the structure of the overall proofs is the same; various proof steps can even be copied from the univariable case.

6. Related work on formalisation

Monica Nesi did the first CCS formalisations for both pure and value-passing CCS [15, 28] using early versions of the HOL theorem prover.[9] Her main focus was on implementing decision procedures (as a ML program, e.g. [29]) for automatically proving bisimilarities of CCS processes. Her work has been a basis for ours [26]. However, the differences are substantial, especially in the way of defining bisimilarities. We greatly benefited from new features and standard libraries in recent versions of HOL4, and our formalisation has covered a larger spectrum of the (pure) CCS theory.

Bengtson, Parrow and Weber did a substantial formalisation work on CCS, π-calculi and ψ-calculi using Isabelle/HOL and its nominal logic, with the main focus on the handling of name binders [30, 31]. More details can be found in Bengtson's PhD thesis [32]. For CCS, he has formalized basic properties for strong/weak equivalence (congruence, basic algebraic laws); the CCS syntax does not have constants or recursion, using instead replication. Other formalisations in this area include the early work of T.F. Melham [33] and O.A. Mohamed [34] in HOL, Compton [35] in Isabelle/HOL, Solange[10] in Coq and Chaudhuri et al. [36] in Abella, the latter focuses on "bisimulation up-to" techniques (for strong bisimilarity) for CCS and π-calculus. Damien Pous [37] also formalised up-to techniques and some CCS examples in Coq. Formalisations less related to ours include Kahsai and Miculan [38] for the spi calculus in Isabelle/HOL, and Hirschkoff [39] for the π-calculus in Coq.

[9] Part of this work can now be found at `https://github.com/binghe/HOL-CCS/tree/master/CCS-Nesi`.

[10] `https://github.com/coq-contribs/ccs`

7. Conclusions and future work

In this paper, besides introducing rooted contraction and its unique solution theorems, we have carried out a comprehensive formalisation of the theory of CCS in the HOL4 theorem prover. In particular, the formalisation supports four methods for establishing (strong and weak) bisimilarities:

1. constructing a bisimulation (the standard bisimulation proof method);
2. constructing a "bisimulation up-to";
3. employing algebraic laws;
4. constructing a system of equations or contractions (i.e., the 'unique-solution' method)

The formalisation has focused on the theory of unique solution of equations and contractions, both in the univariable and in the multivariable cases. It has 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. Indeed we have considered several "unique-solution" results (for various equivalences and preorders); they share many parts of the proofs, but present a few delicate and subtle differences in a few points. In a paper-and-pencil proof, checking all details would be long and error-prone, especially in cases where the proofs are similar to each other or when there are long case analyses to be carried out. We believe that the formalisation of all definitions and theorems are easy to read and to understand, as they are close to their original statement in textbooks.

For some future work, formalising other equivalences and preorders could also be considered, notably trace-based equivalences, as well as more refined process calculi such as value-passing CCS. A different and possibly challenging research line is the formalisation of a different approach [40, 41] 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 the anonymous reviewers and several people from the HOL community, including (in alphabetical order) Robert Beers, Jeremy Dawson, Ramana Kumar, Michael Norrish, Konrad Slind, and Thomas Türk. Sangiorgi has been supported by MIUR-PRIN project 'Analysis of Program Analyses' (ASPRA, ID: 201784YSZ5_004), and by the European Research Council (ERC) Grant DLV-818616 DIAPASoN. The paper was written in memory of Michael J. C. Gordon (1948–2017), the creator of the HOL theorem prover.

[1] C. Tian, D. Sangiorgi, Unique Solutions of Contractions, CCS, and their HOL Formalisation, in: J. A. Pérez, S. Tini (Eds.), Proceedings Combined 25th International Workshop on Expressiveness in Concurrency and 15th Workshop on Structural Operational Semantics, Beijing, China, September 3, 2018, Vol. 276 of Electronic Proceedings in Theoretical Computer Science, Open Publishing Association, 2018, pp. 122–139. doi:10.4204/EPTCS.276.10.

[2] R. Milner, Communication and concurrency, PHI Series in computer science, Prentice-Hall, 1989.

[3] J. C. M. Baeten, T. Basten, M. A. Reniers, Process Algebra: Equational Theories of Communicating Processes, Cambridge University Press, 2010. doi:10.1017/CBO9781139195003.

[4] A. W. Roscoe, The theory and practice of concurrency, Prentice Hall, 1998.
URL http://www.cs.ox.ac.uk/people/bill.roscoe/publications/68b.pdf

[5] A. W. Roscoe, Understanding Concurrent Systems, Springer, 2010. doi:10.1007/978-1-84882-258-0.

[6] D. Sangiorgi, Equations, contractions, and unique solutions, SIGPLAN Not. 50 (1) (2015) 421–432. doi:10.1145/2775051.2676965.

[7] D. Sangiorgi, Equations, contractions, and unique solutions, ACM Trans. Comput. Logic 18 (1) (2017) 4:1–4:30. doi:10.1145/2971339.

[8] R. J. van Glabbeek, A characterisation of weak bisimulation congruence, in: Processes, Terms and Cycles: Steps on the Road to Infinity, Springer, 2005, pp. 26–39. doi:10.1007/11601548_4.

[9] R. Gorrieri, C. Versari, Introduction to Concurrency Theory, Transition Systems and CCS, Springer, Cham, 2015. doi:10.1007/978-3-319-21491-7.

[10] R. Milner, Operational and algebraic semantics of concurrent processes, Handbook of Theoretical Comput. Sci. (1990).

[11] D. Sangiorgi, Introduction to Bisimulation and Coinduction, Cambridge University Press, 2011. doi:10.1017/CBO9780511777110.

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

[13] M. J. C. Gordon, T. F. Melham, Introduction to HOL: A Theorem Proving Environment for Higher Order Logic, Cambrige University Press, New York, NY, 1993.

[14] K. Slind, M. Norrish, A brief overview of HOL4, in: International Conference on Theorem Proving in Higher Order Logics, Springer, 2008, pp. 28–32. doi:10.1007/978-3-540-71067-7_6.
URL http://ts.data61.csiro.au/publications/nicta_full_text/1482.pdf
[15] M. Nesi, A formalization of the process algebra CCS in high order logic, Tech. Rep. UCAM-CL-TR-278, University of Cambridge, Computer Laboratory (1992).
URL http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-278.pdf
[16] HOL4 contributors, The HOL System LOGIC (Kananaskis-13 release) (Aug. 2019).
URL http://sourceforge.net/projects/hol/files/hol/kananaskis-13/kananaskis-13-logic.pdf
[17] M. J. C. Gordon, A. J. Milner, C. P. Wadsworth, Edinburgh LCF: A Mechanised Logic of Computation, Vol. 78 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 1979. doi:10.1007/3-540-09724-4.
[18] R. Milner, Logic for computable functions: description of a machine implementation, Tech. rep., Stanford Univ., Dept. of Computer Science (1972).
URL http://www.dtic.mil/dtic/tr/fulltext/u2/785072.pdf
[19] A. Church, A formulation of the simple theory of types, The journal of symbolic logic 5 (2) (1940) 56–68. doi:10.2307/2266170.
[20] T. F. Melham, Automating Recursive Type Definitions in Higher Order Logic, in: Current Trends in Hardware Verification and Automated Theorem Proving, Springer, New York, NY, 1989, pp. 341–386. doi:10.1007/978-1-4612-3658-0_9.
[21] T. F. Melham, A Package For Inductive Relation Definitions In HOL, in: 1991., International Workshop on the HOL Theorem Proving System and Its Applications, IEEE Computer Society Press, Davis, CA, USA, 1991, pp. 350–357. doi:10.1109/HOL.1991.596299.
[22] HOL4 contributors, The HOL System DESCRIPTION (Kananaskis-13 release) (Aug. 2019).
URL http://sourceforge.net/projects/hol/files/hol/kananaskis-13/kananaskis-13-description.pdf
[23] D. Pous, D. Sangiorgi, Bisimulation and coinduction enhancements: A historical perspective, Formal Asp. Comput. 31 (6) (2019) 733–749. doi:10.1007/s00165-019-00497-w.
URL https://doi.org/10.1007/s00165-019-00497-w
[24] D. Pous, D. Sangiorgi, Bisimulation and coinduction enhancements: A historical perspective, Formal Aspects of Computing (2019). doi:10.1007/s00165-019-00497-w.
[25] D. Sangiorgi, R. Milner, The problem of "Weak Bisimulation up to", in: LNCS 630 - CONCUR '92 - Concurrency Theory, Springer, 1992, pp. 32–46. doi:10.1007/BFb0084781.
[26] C. Tian, A Formalization of Unique Solutions of Equations in Process Algebra, Master's thesis, AlmaDigital, Bologna (Dec. 2017).
URL http://amslaurea.unibo.it/14798/
[27] M. Norrish, B. Huffman, Ordinals in HOL: Transfinite arithmetic up to (and beyond) ω_1, in: International Conference on Interactive Theorem Proving, Springer, 2013, pp. 133–146. doi:10.1007/978-3-642-39634-2_12.
[28] M. Nesi, Formalising a Value-Passing Calculus in HOL, Formal Aspects of Computing 11 (2) (1999) 160–199. doi:10.1007/s001650050046.
[29] R. Cleaveland, J. Parrow, B. Steffen, The concurrency workbench: A semantics-based tool for the verification of concurrent systems, ACM Transactions on Programming Languages and Systems (TOPLAS) 15 (1) (1993) 36–72. doi:10.1145/151646.151648.
[30] J. Bengtson, J. Parrow, A completeness proof for bisimulation in the pi-calculus using isabelle, Electronic Notes in Theoretical Computer Science 192 (1) (2007) 61–75. doi:10.1016/j.entcs.2007.08.017.
[31] J. Parrow, J. Bengtson, Formalising the pi-calculus using nominal logic, Logical Methods in Computer Science 5 (2009). doi:10.2168/LMCS-5(2:16)2009.
[32] J. Bengtson, Formalising process calculi, Ph.D. thesis, Acta Universitatis Upsaliensis (2010).
URL http://uu.diva-portal.org/smash/record.jsf?pid=diva2%3A311032
[33] T. F. Melham, A Mechanized Theory of the Pi-Calculus in HOL, Nord. J. Comput. 1 (1) (1994) 50–76.
URL http://core.ac.uk/download/pdf/22878407.pdf
[34] O. Aït Mohamed, Mechanizing a π-calculus equivalence in HOL, in: E. Thomas Schubert, P. J. Windley, J. Alves-Foss (Eds.), Higher Order Logic Theorem Proving and Its Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 1–16. doi:10.1007/3-540-60275-5_53.
[35] M. Compton, Embedding a fair CCS in Isabelle/HOL, in: Theorem Proving in Higher Order Logics: Emerging Trends Proceedings, 2005, p. 30. doi:10.1.1.105.834.
URL https://web.comlab.ox.ac.uk/techreports/oucl/RR-05-02.pdf#page=36
[36] K. Chaudhuri, M. Cimini, D. Miller, A lightweight formalization of the metatheory of bisimulation-up-to, in: Proceedings of the 2015 Conference on Certified Programs and Proofs, ACM, 2015, pp. 157–166. doi:10.1145/2676724.2693170.
[37] D. Pous, New up-to techniques for weak bisimulation, Theoretical Computer Science 380 (2007) 164–180. doi:10.1016/j.tcs.2007.02.060.
[38] T. Kahsai, M. Miculan, Implementing spi calculus using nominal techniques, in: Conference on Computability in Europe, Springer, 2008, pp. 294–305. doi:10.1007/978-3-540-69407-6_33.
[39] D. Hirschkoff, A full formalisation of π-calculus theory in the calculus of constructions, in: International Conference on Theorem Proving in Higher Order Logics, Springer, 1997, pp. 153–169. doi:10.1007/BFb0028392.
[40] A. Durier, D. Hirschkoff, D. Sangiorgi, Divergence and unique solution of equations, in: R. Meyer, U. Nestmann (Eds.), 28th International Conference on Concurrency Theory (CONCUR 2017), Vol. 85 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017, pp. 11:1–11:16. doi:10.4230/LIPIcs.CONCUR.2017.11.

[41] A. Durier, D. Hirschkoff, D. Sangiorgi, Eager functions as processes, in: 33nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, IEEE Computer Society, 2018, pp. 364–373. doi:10.1145/3209108.3209152.