Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in INFORMATICA

A Formalization of Unique Solutions of Equations in Process Algebra

Relatore:
Chiar.mo Prof.
Davide Sangiorgi

Presentata da:
Chun Tian

Sessione II
Anno Accademico 2016/2017

Abstract

In this thesis, a comprehensive formalization of Milner's *Calculus of Communicating Systems* (also known as CCS) has been done in HOL theorem prover (HOL4), based on an old work in HOL88. This includes all classical properties of strong/weak bisimulation equivalences and observation congruence, a theory of congruence for CCS, various versions of "bisimulation up to" techniques, and several deep theorems, namely the "coarsest congruence contained in $\approx$", and three versions of the "unique solution of equations" theorem in Milner's book.

This work is further extended to support recent developments in Concurrency Theory, namely the "contraction" relation and the related "unique solutions of contractions" theorem found by Prof. Davide Sangiorgi, University of Bologna. As a result, a rather complete theory of "contraction" (and a similar relation called "expansion") for CCS is also formalized in this thesis. Further more, a new variant of contraction called "observational contraction" was found by the author during this work, based on existing contraction relation. It's formally proved that, this new relation is preserved by direct sums of CCS processes, and has a more elegant form of the "unique solutions of contractions" theorem without any restriction on the CCS grammar.

The contribution of this thesis project is at least threefold: First, it can be seen as a formal verification of the core results in Prof. Sangiorgi's paper, and it provides all details for the informal proof sketches given in the paper. Second, a large piece of old proof scripts from the time of Hol88 (1990s) has been ported to HOL4 and made available to all its users. Third, it's a proof engineering research by itself on the correct formalization of process algebra, because the work has made extensive uses of some new features (e.g. coinductive relation) provided in recent versions of HOL4 (Kananaskis-11 and later). The uses of HOL4's rich theory libraries is also a highlight of this project, and it has successfully minimized the development efforts in this work. As a result, this project serves as a sample application of HOL4, which seems a natural choice for the formalization of process algebra.

For the author himself, after this thesis project he has become a skilled user of interactive theorem proving techniques, fully prepared at the level of formal methods in his future research activities in Computer Science.

Acknowledgments

The author want to give special thanks to Prof. *Davide Sangiorgi* for his supervision of this thesis and the initial proposal of the thesis topic on formalizing his recent research results beside the classical ones. Without Prof. Sangiorgi's kind approval and guidances the author wouldn't be able to graduate by doing some formalizations using his favorite software (HOL4).

Thanks to Prof. *Monica Nesi* (University of L'Aquila) for finding and sending her old HOL88 proof scripts on the old CCS formalization (during 1992-1995) to the author in 2016 and 2017. By studying these proof scripts, the author has actually learnt how to use HOL theorem prover, and the existing work has provided a good working basis for quickly reaching to the formalization of deep theorems in the frontier of Concurrency Theory.

Thanks to Prof. *Roberto Gorrieri*, who has taught his *Concurrent System and Models* course twice to the author (in Spring 2016 and 2017), It's all from these courses that the author has learnt Concurrency Theory and CCS. Prof. Gorrieri has also supervised the authors' previous two projects (one for exam, the other for internship) on the porting of the old CCS formalization of Monica Nesi from HOL88 to HOL4. These work now serves as the working basis of this thesis project.

Thanks to Prof. *Andrea Asperti*, who has taught the general idea of Interactive Theorem Proving (ITP) techniques to the author through several seminars. (Prof. Asperti used to teach formal methods but since 2015 he has changed to teach Machine Learning under the same course title) Although it's based on a different theorem prover (Matita, originally created by himself), the author wouldn't be able to learn HOL4 without necessary background knowledges learnt from Prof. Asperti, who has also approved the authors' exam project on a formalization of Lambek Calculus (mostly ported from Coq).

Also thanks to many people from HOL community (Michael Norrish, Thomas Tuerk, Ramana Kumar, Konrad Slind, etc.) for their kind help on resolving issues and doubts that the author has met during this thesis and previous related projects. All these people are professors and senior researchers in other universities. Without their help this thesis project won't be finished in reasonable time.

At last, thanks to Prof. *Mike J. Gordon* (University of Cambridge, the creator of HOL Theorem Prover), who has unfortunately passed away in the evening of August 22, 2017. He ever kindly encouraged the author's work in several private emails, and the author initially learnt the existence of the CCS formalization on HOL, from an introduction paper [1] written by Prof. Gordon.

The author also wants to thank his parents and all his friends in China, Italy and other countries, for all their supports in the past years. The author won't be able to start and focus on his degree study (and finally finish it) without their supports.

The paper is written in LaTeX using LNCS template, with theorems generated automatically by HOL's TeX exporting module (`EmitTeX`) from the working proof scripts.[1]

[1] The author has also submited many bug reports and small Pull Requests to help improving HOL on this part. But still there're known issues which are not reported yet.

Contents

Two-page Summary in Italian

In questo progetto di tesi di laurea, viene proposta una formalizzazione quasi completa del *Calculus of Communicating Systems* (**CCS**) di Robin Milner realizzata in HOL theorem prover (HOL4). L'implementazione proposta si basa sul riutilizzo di una formalizzazione già esistente realizzata in HOL88 da Monica Nesi. Tale formalizzazione è stata quindi riadattata per essere riutilizzata in HOL4 per l'aggiunta di nuove funzionalità.

La formalizzazione include proprietà classiche della bisimulazione, sia forte che debole, e la congruenza fondata sull'osservazione. Viene inoltre proposta una teoria di congruenza per CCS. Sono incluse inoltre varie tecniche di "bisimulazione fino a" (bisimulation up to) e numerosi teoremi profondi, come ad esempio:

1. Il lemma di Hennessy:

$$\vdash p \approx q \iff p \approx^c q \lor p \approx^c \tau\,.\,.\,q \lor \tau\,.\,.\,p \approx^c q$$

2. Il lemma di Deng:

$$\begin{array}{l} \vdash p \approx q \implies \\ \quad (\exists p'.\ p \ {-\tau\rightarrow}\ p' \land p' \approx q) \lor (\exists q'.\ q \ {-\tau\rightarrow}\ q' \land p \approx q') \lor \\ \quad p \approx^c q \end{array}$$

3. Il teorema "coarsest congruence contained in $\approx$":

$$\begin{array}{l} \vdash \texttt{finite\_state}\ p \land \texttt{finite\_state}\ q \implies \\ \quad (p \approx^c q \iff \forall r.\ p + r \approx q + r) \end{array}$$

4. Tre versioni del teorema della "unica soluzione degli equazioni" nel libro di Milner [2]:

$$\begin{array}{l} \vdash \texttt{WG}\ E \implies \forall P\ Q.\ P \sim E\ P \land Q \sim E\ Q \implies P \sim Q \\ \vdash \texttt{SG}\ E \land \texttt{GSEQ}\ E \implies \forall P\ Q.\ P \approx E\ P \land Q \approx E\ Q \implies P \approx Q \\ \vdash \texttt{SG}\ E \land \texttt{SEQ}\ E \implies \forall P\ Q.\ P \approx^c E\ P \land Q \approx^c E\ Q \implies P \approx^c Q \end{array}$$

Successivamente abbiamo esteso questo lavoro a sostegno dello sviluppo della recente Teoria di Concorrente, ossia della relazione *contrazione* ($\succeq_{\mathrm{bis}}$) e del suo teorema *"l'unica soluzione degli equazioni"*, proposta dal Professore Davide Sangiorgi dell'Universit'a di Bologna:

$\vdash$ `WGS` $E \implies \forall P\ Q.\ P \succeq_{bis}\ E\ P \land Q \succeq_{bis}\ E\ Q \implies P \approx Q$

Durante la dimostrazione della teorema "l'unica soluzione degli equazioni", una teoria completa della relazione "contrazione" (e un'altra relazione "expansione") per CCS è stata sviluppata nel progetto. Inoltre, una nuova versione della relazione "contrazione" è stata proposta dall'autore. La nuova relazione, basata sula relazione "contrazione" di Sangiorgi, si chiama "la contrazione fondato sull'osservazione" (*observational contraction*, $\succeq^c_{\text{bis}}$). Questa nuova relazione ha le proprietà migliore rispetto alla contrazione normale, infatti il suo teorema "l'unica soluzione dei equazioni" è in grado di accettare processi composti da somme dirette:

$\vdash$ `WG` $E \implies \forall P\ Q.\ P \succeq^c_{bis}\ E\ P \land Q \succeq^c_{bis}\ E\ Q \implies P \approx Q$

Così non ci sarà bisogno di limitare la grammatica CCS con solo le somme guardinghe, com'è trovato nell'articolo di Sangiorgi e gli altri.

La contribuzione di questa tesi di laurea è al meno triplo:

1. Il lavoro ottimo del professor Sangiorgi viene verificato parzialmente adesso. Coloro che vogliono sapere tutti i dettagli delle dimostrazioni nell'articolo [3] possono leggere la nostra formalizione, se si capisca la logica di higher order (HOL).

2. Una parte grande di codici vecchi da HOL88 (1990s) (scritto dal prof.ssa Monica Nesi) viene trovato, salvato e portato nell'ultima versione di HOL theorem prover. Ora questi codici vivano nel deposito codice di HOL officiale (così è disponibile ai utenti tutti). Non puo perdersi di più.

3. È una ricerca sul modo meglio della formalizzare di *process algebra*, e il tipico oggeto (CCS). Abbiamo usato tantissimo *new feature* nell'HOL4 per minimizzare il lavoro della formalizzazione. Sono inclusi la capacità per definire la relazione co-induttiva, e le teorie esistente (`relation`, `list`, etc.) nell'HOL4. Molti codici vecchi viene eliminati, mentre tutti i teoremi sono ancora lì.

Per l'autore stesso, questo progetto è da maggior parte una pratica per migliorare le sue capacità di usare il theorem prover e il metodo formale per risolvere i problemi in informatica.

Chapter 1

Introduction

In Concurrency Theory [4], the initial research motivation was the modeling of concurrent and reactive systems. Such systems can usually be abstracted into finite or infinite number of *states*, together with labelled *transitions* between pairs of states. From a mathematical view, these states and transitions form a *edge-labeled directed graph*, in which each edge represents a possible transition of states in the system. As a model, we gives such graphs a special name: *Labeled Transition Systems* (LTS), which is originally introduced by R. Keller [5].

The mainline research of Concurrency Theory concerns with a particular class of mathematical models for concurrent communicating processes, namely the class of *algebraic calculi* (process algebra). There're different approaches to the algebraic treatment of processes, the main distinction among them are the constructions by which processes are assembled. Milner's CCS is the simplest treatment with least number of such constructions, while it's still very powerful, in the sense that it's Turing-complete, although Turing-completeness is not enough to ensure the solvability of all the problems in Concurrency Theory. [4]

On the relationship between CCS and LTS, there seems to be two approaches: we can either treat CCS as a compact, algebraic representation of LTS, or treat LTS as the sequential (or interleaving) semantic model of CCS[1]. We think the former approach is not quite fair, because LTS itself has no computational power at all, while CCS can do computations due to its ability of synchronizations, even though this power has been limited between pairs of processes.

In this thesis, we have taken the following approach: we put CCS at the central position and define all equivalence concepts on CCS only, while LTS itself doesn't appear explicitly in the formalization. As the result, it's impossible to formalize the theorems involving both CCS and LTS, e.g. the *Representability Theorem.* But it's should be possible to extend the current work with LTS included as a dedicated type. [2]

[1] According to Gorrieri's book [6], each process algebra has at least three different semantics: sequential semantic (LTS), parallel semantics (Step Transition System, STS) and distributed semantics (Petri nets).

[2] The author has recently found that, it's actually possible and meaningful to embed LTS into CCS as a special constructor, without hurting almost all existing theorems. However he has finaly

Although titled as “A Formalization of Unique Solutions of Equations in Process Algebra”, this thesis project actually contains a quite complete formalization of Milner’s Calculus of Communicating Systems (CCS). The precise CCS class is Finitary and Pure CCS (i.e. no Value-Passing) with explicit relabeling operator. The work has covered almost everything in Milner’s classical textbook [2], and we have also formalized some recent developments in Concurrency Theory, namely the “contraction” relation and its “unique solution of contractions” theorem, introduced in the paper [3] of Prof. Davide Sangiorgi.

Nowadays many researchers in Concurrency Theory have turned to other more powerful process algebras, e.g. π-calculus and ψ-calculus. Even in scope of CCS, the research interests were shifted to CCS extensions like Multi-CCS and Value-Passing CCS. However, the original theory of pure CCS is easy to understand, thus it’s still the main content of many Concurrency Theory courses. On the other side, many more powerful process algebras were based on CCS, thus understanding CCS should be a good step for students who want to study more powerful process algebras in the future.

But even for pure CCS there’re still something quite deep when doing its formalization. For example, the use of “Process constants” in CCS didn’t appears in tother powerful process algebras like π-calculus [7]. It’s actually a challenging work to correctly formalize the “process constants” in CCS. Many authors has completely ignored process constants, while they still claim to successfully formalized CCS. This is the case for the CCS formalization we found using Coq and Isabelle/HOL [8]. What they have formalized is only a sub-language of CCS, in which all theorems for equivalence relations still hold.

The CCS formalization of Monica Nesi [9] has included process constants. In fact, this work has some advanced features beyond theorem proving in the usual sense: besides having proved many theorems, Monica Nesi has also implemented two decision procedures as ML functions: one for generating transitions from any CCS process, the other for automatically checking the equivalences between two processes. Different with similar functionalities in dedicated software, the results generated by theorem provers are trusted, because the outputs are also theorems. This has almost touched the area of CCS-based model checking with self-contained trustness.

In the work of Monica Nesi, there’s a creative treatment of process constants. In standard literature, a general CCS process can be defined by the following grammar: (suppose p and q are already CCS processes, a and b are actions)

$$p ::= \mathrm{nil} \quad | \quad \alpha.p \quad | \quad p + q \quad | \quad p \,\|\, q \quad | \quad (\nu a)p \quad | \quad p[b/a] \quad | \quad C \tag{1.1}$$

here C is a process constant denoting another CCS process, in which the C itself or

decided to not include this work into the thesis, because such additions may decrease the purity of CCS formalization, making people think that certain theorems may not be provable without having LTS ranged over process variables.

some other constants may appear. For example:

$$\begin{cases} A \overset{def}{=} (\nu a)(b.\mathrm{nil}\,|A) \\ B \overset{def}{=} b.B + a.A \end{cases} \tag{1.2}$$

But this form cannot be formalized directly, because in theorem provers a process must be represented as single term with all information. The solution of Monica Nesi is to use process variables and a recursion construction instead:

$$p ::= \mathrm{nil} \;\mid\; \alpha.p \;\mid\; p + q \;\mid\; p \,\|\, q \;\mid\; (\nu a)p \;\mid\; p[b/a] \;\mid\; \mathrm{var}\, X \;\mid\; \mathrm{rec}\, X\, p \tag{1.3}$$

Suppose B is the root process, with the new grammar it's possible to represent this process by the following term:

$$\mathrm{rec}\,\mathbb{B}\,(b.(\mathrm{var}\,B) + a.(\mathrm{rec}\,\mathbb{A}\,(\nu a)(b.\mathrm{nil}\,|\mathrm{var}\,A))) \tag{1.4}$$

This term contains both previous definitions of constants A and B. With a little more imagination, we can see the similarity between above term and a similar λ-calculus term (suppose other CCS operators were available):

$$\lambda\mathbb{B}.\,(b.B + a.(\lambda\mathbb{A}.\,(\nu a)(b.\mathrm{nil}\,|A))) \tag{1.5}$$

Now it comes to the interesting part: given above CCS grammars, the following "process" is also valid:

$$\mathrm{rec}\,\mathbb{A}.\,(a.A + b.C) \tag{1.6}$$

or in the grammar with constants:

$$A \overset{def}{=} a.A + b.C \tag{1.7}$$

Here the constant C is undefined, but the entire definition is still valid according to the CCS grammar. In standard literature, process terms containing undefined constants do not have fully given semantics, c.f. Definition 3.2 (and the remarks after it) in Gorrieri's book:

Definition 1.0.1. (Defined constant and fully defined term) A process constant A is *defined* if it possesses a defining equation: $A \overset{def}{=} p$. A process term q is *fully defined* if all the constants in it are defined.

The requirement that a term must be fully defined is due to the fact that its semantics cannot be fully given otherwise. For instance, term $a.A$ can execute a, but after that we do not know what to do if A is not equipped with a defining equation.

The approach is different in our formalization of CCS: we think every process term has determined semantics, even when it contains undefined constants! The rules are simple: *undefined constants have no transitions.* In another word, an undefined constant A works like a nil, they simply have no transitions. However, we didn't explicitly define such a rule, instead it's a natural consequence derived from the transition semantics.

It remains to understand what's exactly an undefined process variable. Inspired by [10], we have realized that, an undefined constant is actually a free variable in the process term. Formally speaking, now we took the following approach, which is better than those in current standard literature:

1. CCS terms may contain countable many process variables: $A, B, C, \ldots$;
2. Whenever such a variable A is wrapper by a rec construction with the same variable, i.e. $\mathrm{rec}\, A. \ldots A \ldots$, it's a *bounded variable*, or *(process) constant*;
3. Other variables are *free* (process) variables;
4. We call a CCS term with free variables *CCS expressions*;
5. A CCS term without free variables is *CCS process.*
6. Free variables have no transitions.
7. Variable substitutions in CCS terms p can only apply to those free variables, and a free variable cannot be substituted with a term q containing free variables having the same name with any surrounding constant.

These rules are very similar with λ-calculus. For example, in $(\lambda x.x+y)$, x is bounded, and y is free. We can substitute y with another *lambda*-term, but this term shouldn't contain x as free variable (otherwise we may have to rename the bound variable x into x').

With this approach, there's actually a straightforward way to formalize CCS equations with multiple variables, at least for finitely many variables: $\tilde{X} = \tilde{E}[\tilde{X}]$, in which the right side of each single equation $E_i[\tilde{X}]$ can be seen as a repeated variable substitution process: $E_i\{p_1/X_1\}\{p_2/X_2\}...\{p_n/X_n\}$, where $p_1, p_2, \ldots, p_n$ are solution of the equation as a list of CCS processes (without free variables). Thus in theory it's actually possible to formalize the "unique solution of equations" theorem with equations having multiple variables.

Above approach is currently not fully implemented, although we do have defined functions for retrieving the set of free and bound variables in CCS terms, and multi-variable substitutions. In fact, above approach was realized too late during the thesis, thus we chose to only consider single-variable equations, which actually coincide with semantics contexts of CCS as unary λ functions, taking one CCS process and returning another.

This choice has guaranteed the quick complete of this thesis project in reasonable time. But there's one drawback: it's impossible to express (and prove) the "Completeness of contractions" (Theorem 3.13 of [3]), because it requires an infinite system of contractions. A formalized theory of "contraction" wouldn't be complete with this completeness theorem, because it's the theorem where the "expansion" (a closely related relation) doesn't hold. But currently we have no choice but to leave it as not formalized.

1.1 Limitations

There's limitation in our formalization: the CCS summation operator is only binary, while in Milner's original CCS definition, infinite sums of processes are supported. As the result, only finite summation can be expressed in our definition of CCS, and the resulting CCS language is finitary.

Infinite sums of processes over a arbitrary infinite set of processes turns out to be impossible for higher order logic, i.e. having the following datatype:

```
Datatype ‘CCS = summ (CCS set) | ...‘;
```

The reason was explained by Michael Norrish (the HOL maintainer):

> "You can't define a type that recurses under the set "constructor" (your `summ` constructor has (CCS set) as an argument). Ignoring the num set argument, you would then have an injective function (the `summ` constructor itself) from sets of CCS values into single CCS values. This ultimately falls foul of Cantor's proof that the power set is strictly larger than the set."

What's certainly allowed in higher order logic is the summation over functions taking numerical indexes (numbers or even ordinals) returning CCS processes, i.e. something like:

```
Datatype ‘CCS = summ (num -> CCS) | ...‘;
```

However, currently the datatype package in HOL4 cannot support this kind of "nesting recursive definitions". In theory it's possible to define the desired datatype manually by constructing a bijection from another existing type, but this work is hard and goes beyond the author's ability.[3] The long-term plan is to implement (by the author) the idea of Andrei Popescu [12] based on Category Theory. It has been implemented in Isabelle/HOL. The author would like to have the same datatype defining ability in HOL4, and once this goal is achieved, the current CCS formalization will be also updated to support infinite sums.

It's also worth noting that, Milner's CCS has several different forms, in which the more comprehensive one was introduced in [10] as the last chapter of the *Handbook of Theoretical Computer Science, Volume B*. In this paper, Robin Milner has defined the set $\mathscr{E}$ of processes expressions $E, F, \ldots$ (also called terms) as the smallest set including $\mathscr{E}$ and the following expressions–when E, E_i are already in $\mathscr{E}$:

$\alpha.E$, a *Prefix* ($\alpha \in Act$),
$\sum_{i \in I} E_i$, a *Summation*,
$E_0 \| E_1$, a *Composition*,
$E \setminus L$, a *Restriction* ($L \subseteq \mathscr{L}$),
$E[f]$, a *Relabeling* (f is a relabeling function),
$\mathrm{fix}_j\{X_i = E_i \colon i \in I\}$, a *Recursion* ($j \in I$).

[3] In the work of Monica Nesi on formalizing Value-Passing CCS [11], there're supports of infinite sums in the CCS datatype, however currently we don't know how it's actually implemented, as the related proof scripts are not available.

In the final form (Recursion) the variables X_i are *bound* variables. For any expression E, its *free* variables $fv(E)$ are those while occur unbound in E. We say E is *closed* if $fv(E) = \emptyset$; in this case we say E is a *process*, and we shall use P, Q, R to range over the processes $\mathscr{P}$.

In this way, "expressions" (as the core part of an equation) now becomes first-class definitions in the theory of CCS, and a normal CCS process is nothing but an expression without any free variable. Further more, we quote,"for simplicity purpose we sometimes use an alternative formulation of Recursion: instead of the **fix** construction we introduce *process constants* $A, B, \ldots$ *into the language. We then admit a set* $\tilde{A} \overset{\text{def}}{=} \tilde{P}$ *of* defining equations *for the constants.*" So the process constants are actually defined by processes equations.

Unfortunately, almost all CCS textbooks (including the Milner's own book [2]) have adopted the "simple" approach in which process constants are part of the CCS grammar, and there's no **fix** construction.

Our work is derived from the work of Monica Nesi, and we didn't touch the definition of her CCS datatype except for the uses of type variables. In this thesis, we focused on the formalization of various versions of the "unique solution of equations/contractions" theorem, limited for the case of single-variable equations, because in this special case an expression is also a context, i.e. lambda functions taking one CCS process returning another. And the concepts of guardedness can also be easily and recursively defined by lambda functions. In this way, we actually ignored all free variables in CCS terms and didn't treat them as equation variables at all. And the resulting "unique solution of equations" theorems have very simple and elegant statements.

The proof scripts (about 20,000 lines) of this thesis is currently available in the '`example/CCS`' directory of HOL4 source code repository [4].

The structure of this thesis paper is as follows: (TODO) In Chapter ..

1.2 Related work

Beside the early CCS formalization by Monica Nesi, the author found only two other CCS formalizations. One is done in Isabelle/HOL (based on Nominal datatypes) in 2010, by Jesper Bengtson as part of his PhD thesis project [8]. He also formalized π-calculus and ψ-calculus, all available in Isabelle's Archive of Formal Proofs (AFP) [5]. Since he has done 3 big formal systems in one project, it's expected that, for each of them only very basic results can be done. The part of CCS has indeed formalized classic properties for strong/weak equivalence and observational congruence, however his CCS grammar doesn't include process constants. And his proved theorems also look quite different with their original forms in CCS textbooks.

The other formalization we found is based on Coq, done by Solange Coupet-Grimal in 1995, which is part of the "coq-contribs" archive (Coq's official user con-

[4] `https://github.com/HOL-Theorem-Prover/HOL`

[5] `https://www.isa-afp.org`

tributed code) [6]. This is actually a formalization of transition systems without any algebraic operator, and all has been covered is still the very basic properties of strong/weak equivalence and observational congruence.

Monica Nesi's early work [9] on pure CCS covered basically the same thing, i.e. classic properties for strong/weak equivalence and observational congruence. However, in her work there's a decision procedure written as ML program (i.e. a function): given any CCS process, this program can generate a theorem which precisely captures all possible (direct) transitions leading from the process. Monica Nesi also described another decision procedure for checking the bisimulation equivalence between two CCS processes. She also formalized Hennessy-Milner Logic (HML), although no decision procedures were implemented. HOL proof scripts are simply ML programs (for HOL88 this was Classic ML, now it's Standard ML), so is the HOL theorem prover itself. So it's very natural for its end users to write ML programs to extend the automatic proof searching abilities during the formalization of related theorems. But this same thing is not obvious for Isabelle, Coq and many other theorem provers, where the language for expressing theorems and proofs are usually domain languages unrelated with the underlying programming language for implementing the theorem prover itself.

As the audience shall see, the coverage of this project is so far the largest among all existing CCS formalizations. And our definitions, proved theorems have almost the same look as in standard CCS textbooks (thus easily understandable for students and professors in Concurrency Theory). Almost all proofs have clear, human readable steps for replay and learning purposes.

1.3 Availability

It's usually very hard to maintain a stable online URL to guarentee the availability of the work done in this thesis project for many years. The author's strategy is to put the entire work into the official code repository of HOL theorem prover, which is currently hosted at GitHub[7], under the folder "`examples/CCS`". This also means that, every HOL user who has recently installed latest HOL4 from its source code cloned by Git, or is using a formal HOL4 release since Kananaskis 12 (not released at current moment) should have a copy of our proof scripts in his computer under the same sub-folder related HOL top-level folder.

But things may change in the future, and the proof scripts found in HOL's code base may be further upgraded by the author or others. Thus it seems also necessary to maintain a copy which is exactly the same as the one mentioned in this thesis paper. Currently such a copy is also in Github[8] but the author cannot promise its availability for even next 10 years. Whenever this link is not available any more, please find the "official" version in HOL theorem prover.

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

[7] `https://github.com/HOL-Theorem-Prover/HOL`

[8] `https://github.com/binghe/informatica-public/tree/master/thesis`

Chapter 2

HOL Theorem Prover

The HOL interactive theorem prover is a proof assistant for higher-order logic: a programming environment in which logical theorems can be proved, with proof tools implemented. Built-in decision procedures and theorem provers can automatically establish many simple theorems (users may have to prove the hard theorems interactively!) An oracle mechanism gives access to external programs such as SMT and BDD engines. HOL is particularly suitable as a platform for implementing combinations of deduction, execution and property checking.

The approach to mechanizing formal proof used in HOL is due to Robin Milner [13], who also headed the team that designed and implemented the language ML.

HOL has several different major versions in history, the latest version is usually called HOL4. A relationship between HOL4 and other HOL versions and derived systems can be seen from Figure 2.1.

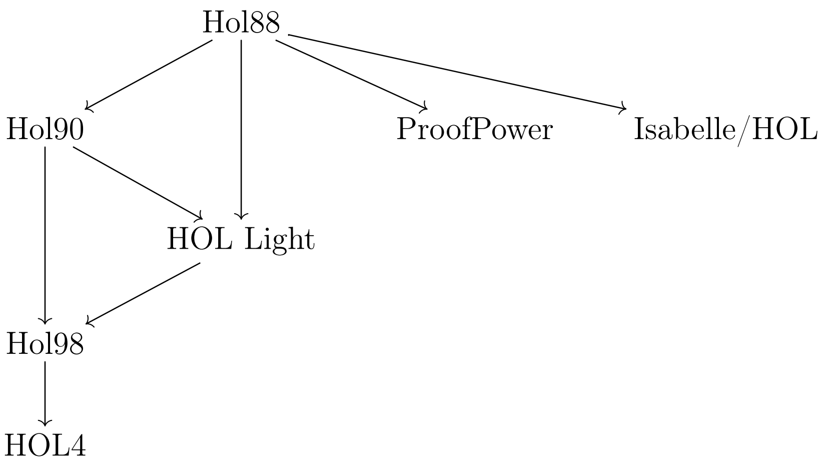

Figure 2.1: HOL4 and its relatives

HOL4 was implemented in Standard ML. This programming language plays three roles here:

1. It serves as the underlying implementation language for the core HOL engine;
2. It's used to implement tactics (and tacticals) for automatic theorem proving;
3. It's used as the command language of the interactive HOL system.

These roles can be done in separated languages. For example, in the Coq proof-assistant [1], the proof-assistant itself is written in OCaml, but the language for expressing theorems (and their proofs) is another language called *Gallina*, while the tactic language is again another different language called *Ltac*. But in HOL-4, all these three languages are the same (except for the inner language of logic, which is derived from Classic ML). This fact also allows us to use HOL-4 as a general programming platform and write an entire model checking software which uses the theorem prover as a library. Such advantage is not possible for most other theorem provers, at least not that straightforward.

2.1 Higher Order Logic

Higher Order Logic (or "HOL Logic") means simply typed λ-calculus plus Hibert choice operator, axiom of infinity, and rank-1 polymorphim (type variables). In this section we briefly introduce the HOL Logic, and please refer to official documents [14] for a complete picture.

The HOL syntax contains syntactic categories of types and terms whose elements are intended to denote respectively certain sets and elements of sets. This model is given in terms of a fixed set of sets $\mathcal{U}$, which will be called the universe and which is assumed to have the following properties:

Inhab Each element of $\mathcal{U}$ is a non-empty set.

Sub If $X \in \mathcal{U}$ and $\emptyset \neq Y \subseteq X$, then $Y \in \mathcal{U}$.

Prod If $X \in \mathcal{U}$ and $Y \in \mathcal{U}$, then $X \times Y \in \mathcal{U}$. The set $X \times Y$ is the cartesian product, consisting of ordered pairs (x, y) with $x \in X$ and $y \in Y$.

Pow If $X \in \mathcal{U}$, then the powerset $\wp(X) = \{Y : Y \subseteq X\}$ is also an element of $\mathcal{U}$.

Infty $\mathcal{U}$ contains a distinguished infinite set I.

Choice There is a distinguished element $\mathrm{ch} \in \prod_{X \in \mathcal{U}} X$. The element of the product $\prod_{X \in \mathcal{U}} X$ are (dependently typed) functions: the for all $X \in \mathcal{U}$, X is non-empty by **Inhab** and $\mathrm{ch}(X) \in X$ witnesses this.

Although HOL also supports non-standard models, above set-theoretic model is the basis of all builtin theories of HOL theorem prover and is usually adopted by HOL users when they claim to have formalized something in HOL.

[1] `https://coq.inria.fr`

The above assumptions on U are strictly weaker than those imposed on a universe of sets by the axioms of ZFC (Zermelo-Frankel set theory with the Axiom of Choice), principally because $\mathcal{U}$ is not required to satisfy any form of the Axiom of Replacement. Thus it's possible to prove the existence of a set $\mathcal{U}$ with the above properties from the axioms of ZFC[2], and it's possible in principal to give a completely formal version within ZFC set theory of the semantics of the HOL logic.

On the other side, some mathematics theories cannot be formalized in above standard model of HOL. One notable example is the full theory of ordinal numbers in the von Neumann hierarchy and the cardinal numbers. However, HOL can support the formalization of a large portation of mathematics, including Lebesgue Measures and Probability Theory. We have also strong witenesses seen in another theorem prover in HOL family (i.e. Isabelle/HOL).

There are some notable consequences of above assumptions (**Inhab**, **Sub**, **Prod**, **Pow**, **Infty** and **Choice**). Important one is the concept of functions. In set theory, functions are identified with their graphs, which are certain sets of order pairs. Thus the set $X \to Y$ of all functions from a set X to a set Y is a subset of $\wp(X \times Y)$; and it is a non-empty set when Y is non-empty. So **Sub**, **Prod** and **Pow** together imply that $\mathcal{U}$ also satisfies

Fun If $X \in \mathcal{U}$ and $Y \in \mathcal{U}$, then $X \to Y \in \mathcal{U}$.

By iterating **Prod**, one has that the certasian product of any finite, non-zero number of sets in $\mathcal{U}$ is again in $\mathcal{U}$. But $\mathcal{U}$ also contains the cartesian product of no sets, which is to say that it contains a one-element set (by virtue of **Sub** applied to any set in $\mathcal{U}$–**Infty** guarantees there is one); for definiteness, a particular one-element set will be singled out:

Unit $\mathcal{U}$ contains a distinguished one-element set $1 = \{0\}$.

Similarly, because of **Sub** and **Infty**, $\mathcal{U}$ contains two-element sets, one of which will be singled out:

Bool $\mathcal{U}$ contains a distinguished two-element set $2 = \{0, 1\}$.

2.1.1 Types and Terms

The types of the HOL logic are expressions that denote sets (in the universe $\mathcal{U}$). HOL's 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. 2.2 for its BNF grammar. Noticed that, in HOL the standard atomic types *bool* and *ind* denote, respectively, the distinguished two-element set 2 and the distinguished infinite set I.

The terms of the HOL logic are expressions that denote elements of the sets denoted by types. There're four kinds of terms in the HOL logic. There can be described approximately by the BNF grammar in Fig. 2.3.

[2] For example, one could take $\mathcal{U}$ to consist of all non-empty sets in the von Neumann cumulative hierarchy formed before stage $\omega + \omega$.

$$\sigma \quad ::= \quad \underset{\substack{\uparrow\\ \text{type variables}}}{\alpha} \quad | \quad \underset{\substack{\uparrow\\ \text{atomic types}}}{c} \quad | \quad \underbrace{(\sigma_1,\ldots,\sigma_n)op}_{\substack{\uparrow\\ \text{compound types}}} \quad | \quad \underbrace{\sigma_1\to\sigma_2}_{\substack{\uparrow\\ \text{function types}\\ \text{(domain } \sigma_1\text{, codomain } \sigma_2\text{)}}}$$

Figure 2.2: HOL's type grammar

$$t \quad ::= \quad \underset{\substack{\uparrow\\ \text{variables}}}{x} \quad | \quad \underset{\substack{\uparrow\\ \text{constants}}}{c} \quad | \quad \underbrace{t\ t'}_{\substack{\uparrow\\ \text{function applications}\\ \text{(function } t\text{, argument } t'\text{)}}} \quad | \quad \underbrace{\lambda x.\ t}_{\substack{\uparrow\\ \lambda\text{-abstractions}}}$$

Figure 2.3: HOL's term grammar

So HOL is a deductive system for simply typed λ-calculus, as its term syntax has shown, with each term in HOL associated with a unique simple type.

2.1.2 The HOL deductive system

The deductive system of the HOL logic is specified by the following eight rules of inference. (The identifiers in square brackets are the names of the ML functions in the HOL system that implement the corresponding inference rules.)

1. Assumption introduction [`ASSUME`]

$$\frac{}{t \vdash t}$$

2. Reflexivity [`REFL`]

$$\frac{}{\vdash t = t}$$

3. Beta-conversion [`BETA_CONV`]

$$\frac{}{\vdash (\lambda x.\ t_1)t_2 = t_1[t_2/x]}$$

- Where $t_1[t_2/x]$ is the result of substituting t_2 for x in t_1, with suitable renaming of variables to prevent free variables in t_2 becoming bound after substitution.

4. Substitution [`SUBST`]

$$\frac{\Gamma_1 \vdash t_1 = t_1' \qquad \cdots \qquad \Gamma_n \vdash t_n = t_n' \qquad\qquad \Gamma \vdash t[t_1,\ldots,t_n]}{\Gamma_1 \cup \cdots \cup \Gamma_n \cup \Gamma \vdash t[t_1',\ldots,t_n']}$$

- Where $t[t_1,\ldots,t_n]$ denotes a term t with some free occurrences of subterms t_1, ... , t_n singled out and $t[t_1',\ldots,t_n']$ denotes the result of replacing each selected occurrence of t_i by t_i' (for $1 \leq i \leq n$), with suitable renaming of variables to prevent free variables in t_i' becoming bound after substitution.

5. Abstraction [`ABS`]

$$\frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash (\lambda x.\ t_1) = (\lambda x.\ t_2)}$$

- Provided x is not free in Γ.

6. Type instantiation [`INST_TYPE`]

$$\frac{\Gamma \vdash t}{\Gamma[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n] \vdash t[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n]}$$

- Where $t[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n]$ is the result of substituting, in parallel, the types $\sigma_1, \ldots, \sigma_n$ for type variables $\alpha_1, \ldots, \alpha_n$ in t, and where $\Gamma[\sigma_1, \ldots, \sigma_n/\alpha_1, \ldots, \alpha_n]$ is the result of performing the same substitution across all of the theorem's hypotheses.

- After the instantiation, variables free in the input can not become bound, but distinct free variables in the input may become identified.

7. Discharging an assumption [`DISCH`]

$$\frac{\Gamma \vdash t_2}{\Gamma - \{t_1\} \vdash t_1 \Rightarrow t_2}$$

- Where $\Gamma - \{t_1\}$ is the set subtraction of $\{t_1\}$ from Γ.

8. Modus Ponens [`MP`]

$$\frac{\Gamma_1 \vdash t_1 \Rightarrow t_2 \qquad\qquad \Gamma_2 \vdash t_1}{\Gamma_1 \cup \Gamma_2 \vdash t_2}$$

In HOL, all the proofs are finally reduced to sequences of applications of above eight primitive inference rules. What's also implicitly given by above rules are the following two foundamental operators:

Equality Equality (`= : 'a -> 'a -> bool`) is an infix operator.

Implication Implication ($\Rightarrow$: `bool -> bool -> bool`) is the *material implication* and is an infix operator that is right-associative, i.e. $x \Rightarrow y \Rightarrow z$ parses to the same term as $x \Rightarrow (y \Rightarrow z)$.

Noticed that, unlike in other formal systems like Propositional Logic, logical implication ($\Rightarrow$) in HOL is not defined by other operators (e.g. $p \to q := \neg p \vee q$), instead it's a primitive constant whose meanings are given completely by above eight primitive deduction rules (`DISCH` and `MP`, to be precise).

2.1.3 Standard Theory of HOL

It's quite amazing (at least to the author) that, all the rest logical constants of Propositional Logic and First-order Logic (universal and existential quantifiers) can be embeded in λ-calculus, with equality and logical implication:

$$\begin{array}{ll}
\vdash & \mathrm{T} = ((\lambda x_{bool}.\ x) = (\lambda x_{bool}.\ x)) \\
\vdash & \forall = \lambda P_{\alpha \to bool}.\ \ P = (\lambda x.\ \mathrm{T}) \\
\vdash & \exists = \lambda P_{\alpha \to bool}.\ \ P(\varepsilon\ P) \\
\vdash & \mathrm{F} = \forall b_{bool}.\ \ b \\
\vdash & \neg = \lambda b.\ \ b \Rightarrow \mathrm{F} \\
\vdash & \wedge = \lambda b_1\ b_2.\ \forall b.\ (b_1 \Rightarrow (b_2 \Rightarrow b)) \Rightarrow b \\
\vdash & \vee = \lambda b_1\ b_2.\ \forall b.\ (b_1 \Rightarrow b) \Rightarrow ((b_2 \Rightarrow b) \Rightarrow b)
\end{array}$$

with the following special notations:

Notation	Meaning
$\forall x_\sigma.\ t$	$\forall(\lambda x_\sigma.\ t)$
$\forall x_1\ x_2\ \cdots\ x_n.\ t$	$\forall x_1.\ (\forall x_2.\ \cdots\ (\forall x_n.\ t)\ \cdots)$
$\exists x_\sigma.\ t$	$\exists(\lambda x_\sigma.\ t)$
$\exists x_1\ x_2\ \cdots\ x_n.\ t$	$\exists x_1.\ (\exists x_2.\ \cdots\ (\exists x_n.\ t)\ \cdots)$
$t_1\ \wedge\ t_2$	$\wedge\ t_1\ t_2$
$t_1\ \vee\ t_2$	$\vee\ t_1\ t_2$

Thus in HOL, an universal quantified term like $\forall x.P(x)$ is represented as the functional application of the constant $\forall$ as a λ-function and another λ-function $\lambda x.P(x)$. At first glance, these definitions seem a little strange. However, it can be manually verified that, they're indeed consistent with the usual semantics in Propositional Logic and First-order Logic. For the new constant (ε) used in the definition of existential quantifier ($\exists$), HOL gives it the following meaning:

Choice If t is a term having type $\sigma \to$ bool, then `@x.t x` (or, equivalently, `$@t`) denotes *some* member of the set whose characteristic function is t. If the set is empty, then `@x.t x` denotes an arbitrary member of the set denoted by σ. The constant `@` is a higher order version of Hilbert's ε-operator; it is related to the constant ι in Church's formulation of higher order logic.

However so far above statement is just a wish: its behavior is not given by any primitive deduction rules, nor have we its definition upon other more primitive objects. To actually have the choice operator we need an axiom. There're totally four such axioms in HOL's standard model:

`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` $\vdash \exists f_{ind \to ind}.\ \mathrm{One\_One}\ f\ \wedge\ \neg(\mathrm{Onto}\ f)$

Among these axioms, `BOOL_CASES_AX` represents the Law of Excluded Middle, `ETA_AX` is the η-conversion rule in λ-calculus (η-conversion cannot be derived from β-conversion). `SELECT_AX` gives Hilbert's ε-operator the actual meaning, it has the position as the Axiom of Choice in ZFC. The last axiom `INFINITY_AX` represents the Axiom of Infinity in ZFC, the two other constants appeared in the axiom have the following definitions:

$$\begin{aligned}&\vdash \text{One\_One} = \lambda f_{\alpha\to\beta}.\ \forall x_1\ x_2.\ (f\ x_1 = f\ x_2) \Rightarrow (x_1 = x_2)\\ &\vdash \text{Onto} = \lambda f_{\alpha\to\beta}.\ \forall y.\ \exists x.\ y = f\ x\end{aligned}$$

This is how the concept of "infinity" is formalized: if there's a function $f\colon I \to I$ which is one-one but not onto, then the set I has no choice but contains infinite elements. (And starting with this infinite set, the set of natual numbers, real numbers, ... can be built in HOL)

The last primitive definition is for defining new primitive types in HOL:

$$\begin{aligned}\vdash \text{Type\_Definition} = \ &\lambda P_{\alpha\to bool}\ rep_{\beta\to\alpha}.\ \text{One\_One}\ rep \ \ \wedge\\ &\quad(\forall x.\ P\ x \ = \ (\exists y.\ x = rep\ y))\end{aligned}$$

In HOL, new primitive types are defined by mapping a subset of existing types into a new type. Such mappings must be bijections. The constant Type_Definition is used for asserting such mappings, then new types can be used legally as primitive types like `bool` and `ind`. Notable types defined in this way are `num` (natural numbers), `sum` (sum types), `prod` (product types), `unit` (singleton type), etc.

The next level of types is customized inductive datatypes like structures in C-like languages. HOL's Datatype package was created by Thomas Melham [15], it's not part of the HOL Logic itself but a package for automatically creation of new inductive and recursive datatypes with supporting theorems. In this project, the `CCS` datatype is defined by this package.

2.2 Relations in HOL

2.2.1 Notations

From the view of mathematics, a n-ary *relation* R is nothing but a *set* of n-tuples, and we say a term (or just a sentence) $R\ x_1\ x_2\ \ldots\ x_n$ is true if and only if $(x_1, x_2, \ldots, x_n) \in R$. In particolar, an unary relation is also called a *predicate* which usually denote the property of objects. Binary relations are usually represented in infix ways, e.g., instead of $R\ x\ y$ we write $x\ R\ y$. Notable examples of binary relations include equality ($x = y$), inequality ($x < y$) and set relations like $A \subset B$ or even $x \in A$.

In general cases, a (binary) relation needs not to be transitive at all, nor each parameter has the same type. From the view of higher order logic, or simple-typed lambda calculus, however, there's a difference between terms like

$$R\ x_1\ x_2\ \ldots\ x_n,$$

and

$$R\ (x_1, x_2, \ldots, x_n),$$

in which the latter one equals to $(x_1, x_2, \ldots, x_n) \in R$ in predicate-based set theory [16].

In HOL, the former term has the types like

$$\alpha \to \beta \to \cdots \to \mathit{bool},$$

while the latter term has the types like

$$(\alpha \ \# \ \beta \ \# \ \cdots) \to \mathit{bool}$$

where $\alpha, \beta, \ldots$ represent type variables (for each parameter of the relation) and $\#$ is the operator for building Cartesian product types. The two types are equivalent (but not the same) for precisely defining any relation. The process of converting a relation from the former type to latter type is called *curry*, and the reverting process is called *uncurry*.

Relations used in logic, formal methods and theorem proving areas are usually defined in curried forms, because it has extra advantages that, a relation term with parameters *partially given* could still has a meaningful type. On the other side, mathematicians usually use the uncurried forms. In the rest of this paper, we will ignore the differences between curried and uncurried forms, and the reader should make implicit type translations in context-switching between mathematics and theorem proving talks.

2.2.2 Ways to define relations

To precisely define a relation (which is essentially a set), we can explicitly enumerating all its elements (when the number of them is finite), e.g.

$$R = \{(1,2), (2,3), (3,4), \ldots\}$$

or define the set by its characteristic function f (with the proof of the soundness of such functions given first, as required by ZFC), e.g.

$$R = \{(x,y) \colon f(x,y) = \text{true}\}$$

But mostly above two ways are not enough, instead, we have to define a relation by a set of rules in the following two forms:

$$\frac{\emptyset}{R\ x_1\ x_2\ \ldots\ x_n} \qquad\qquad \frac{\text{<hypothesis>}}{R\ x_1\ x_2\ \ldots\ x_n}$$

Rules in above left form should be understood as a element $(x_1, x_2, \ldots, x_n)$ belongs to the relation (as set) *unconditionally* (thus we usually call such rules as *base rules* or *axioms*), while rules in above right form should be understood in either *forward* form

$$\forall x_1, x_2, \ldots x_n.\ \text{<hypothesis>} \Rightarrow (x_1, x_2, \ldots, x_n) \in R,$$

and/or *backward* form

$$\forall x_1, x_2, \ldots x_n.\ (x_1, x_2, \ldots, x_n) \in R \Rightarrow \text{the same <hypothesis> in existence form,}$$

depending on different situations. In general, a hypothesis used in the above rules could be any term, in which the relation R itself can also be used, even when it's not fully defined yet. In such cases, we say the relation is defined *recursively*, and the previous two rule-defining methods cannot handle such relation definitions.

But for relations which must be defined recursively, in general we can't simply define it, say R, in the following way:

$$\forall x_1, x_2, \ldots, x_n.\ (x_1, x_2, \ldots, x_n) \in R \text{ if and only if } x_1, x_2, \ldots, x_n \text{ satisfy all the rules.}$$

In other words, the relation R is defined as the closure of all those rules.

The problem is, the resulting relation R of above definition may not be unique (even with *axiom of extension* (first axiom of ZFC) assumed always), thus the whole definition cannot be used to precisely describe a mathematic object in general.

To see the reasons, we can construct a function $F(R)$ from *any* rule set. The function takes a relation and returns another relation of the same type. The way to construct such a function is to just replace all occurences of R in the conclusion (i.e. under the line) part, into $F(R)$ and keep the (possible) occurences of R in hypothesis unchanged. So the rules now look like this:

$$\frac{\emptyset}{F(R)\ x_1\ x_2\ \ldots\ x_n} \qquad\qquad \frac{\text{<hypothesis>}}{F(R)\ x_1\ x_2\ \ldots\ x_n}$$

It's important to notice that, the definition of function F (as above rules) is *not* recursive. Given any R as input (assuming the type correctness, of course), $F(R)$ is simply the set of tuples indicated in all base rules, unioned with all tuples which satisfy the hypothesis of any other rules. In particolar, if $R = F(R)$, then we go back to the previous rules, this is to say, R is closed under those rules if and only if R is a fixed point of the function F.

Now we discuss the characteristics of function F. The first thing to notice is: no matter what hypothesis we have in each rule, the function F defined in this way, is always *monotone* (or *order preserving*), i.e. $\forall R_1\ R_2.\ R_1 \subset R_2 \Rightarrow F(R_1) \subset F(R_2)$. The reason is simple, if we consider more than tuples in R_1 with the hypothesis, we can only add more (or at least the same) than tuples in $F(R_1)$ into $F(R_2)$, there's absolutely no chance to eliminate any existing tuples in $F(R_1)$ from $F(R_2)$. (The conclusion is still true if all non-axiom rules were understood in backward)

On the other side, the set $\mathcal{L}$ of all relations R (of the same type) forms a *complete lattice*, because $\mathcal{L}$ has always a top element (the relation containing all possible tuples) and a bottom element (empty relation). Thus F is an order-preserving self-maping function on a complete lattice. According to Knester-Tarski Fixed Point Theorem [17], F has always a least fixed point and a greatest fixed point. To see the two fixed points do not coincide in general, it's enough to take a sample relation which satisfy only one rule:

$$\frac{n+1 \in R}{n \in R}$$

where n has the type of natural numbers (0, 1, ...). Then it's trivial to see that, the following function F

$$\frac{n+1 \in R}{n \in F(R)}$$

has a least fixed point $\emptyset$ (no numbers) and a greatest fixed point $\mathbb{N}$ (all numbers). Noticed that, it's not true that all relations between the least and greatest fixed points are also fixed points (in current case there's no other fixed points), but in general there may be other fixed points beside the least and greatest ones.

Thus, beside the for-sure existence of fixed points, the number of fixed points and the usefulness between least and greatest fixed points, are totally decided by specific relation defining rules. Most of time, one of least and greatest fixed points is trivial: either the least fixed point is the bottom element or the greatest fixed point is the top element, then only the other one is meaningful to be defined.

2.2.3 Inductive and co-inductive relation definitions

Fixing a rule set, depending on the desired relation object, sometimes we want the least fixed point, sometimes we want the greatest fixed point. Whenever the least fixed point is used, such a relation definition is called *inductive* relation definition, and the other one is called *co-inductive* relation definition.

Noticed that, in many relation definitions, the fixed points are not explicitly mentioned, while we still have precise (unique) definition for the desired relation object. For instance, the following defintion of an unary relation (or predicate, or set) is a typical inductive relation definition:

Definition 2.2.1. The set R is the *smallest* set such that

1. $0 \in R$;

2. forall n, if $n \in R$ then $n + 2 \in R$.

It's not hard to imagine that, the set R is the set of all even numbers, i.e. $R = \{0, 2, 4, 6, \ldots\}$.

Instead, if we required a *greatest* set which is closed under the same rules, we could have the entire set of natural numbers $\{0, 1, 2, \ldots\}$, even when the axiom rule $0 \in R$ is absent.

Sometimes the greatest and least fixed points coincide, and as a result there's only a single fixed point for certain rules. For instance, let A be a set, the following rules can be used to inductively define all finite lists with elements from A:

$$\frac{\emptyset}{\text{nil} \in \mathcal{L}} \qquad\qquad \frac{s \in \mathcal{L} \quad a \in A}{\langle a \rangle \bullet s \in \mathcal{L}}$$

According to [18] (p. 36), if above rules were used co-inductively, the resulting relation will be the set all *finite and infinite* lists. But if we limit the type of lists to only finite lists (e.g. the list data structure defined inductively), then the co-inductive defintion and inductive defintion coincide. This fact can be proved in HOL4 by the following scripts:

```
val (List_rules, List_ind, List_cases) = Hol_reln
   ‘(!l. (l = []) ==> List l) /\
    (!l h t. (l = h::t) /\ List t ==> List l)‘;

val (coList_rules, coList_coind, coList_cases) = Hol_coreln
   ‘(!l. (l = []) ==> coList l) /\
    (!l h t. (l = h::t) /\ coList t ==> coList l)‘;

val List_imp_coList = store_thm (
   "List_imp_coList", ‘‘!l. List l ==> coList l‘‘,
    HO_MATCH_MP_TAC List_ind
 >> RW_TAC bool_ss [coList_rules]);

val coList_imp_List = store_thm (
   "coList_imp_List", ‘‘!l. coList l ==> List l‘‘,
    Induct_on ‘l‘
 >| [ RW_TAC bool_ss [List_rules, coList_rules],
      STRIP_TAC
   >> ONCE_REWRITE_TAC [coList_cases]
   >> ONCE_REWRITE_TAC [List_cases]
   >> REPEAT STRIP_TAC
   >| [ ASM_REWRITE_TAC [],
        SIMP_TAC list_ss []
     >> ‘t = l‘ by PROVE_TAC [CONS_11]
     >> PROVE_TAC [] ] ]);

val List_eq_coList = store_thm (
   "List_eq_coList", ‘‘!l. coList l = List l‘‘,
    PROVE_TAC [List_imp_coList, coList_imp_List]);
```

Here we have defined two unary relations (`List` and `coList`), using exactly the same rules (except for the relation names):

$$\begin{array}{l}\vdash (\forall l.\ (l = [\,]) \implies \texttt{List}\ l) \ \wedge \\ \quad \forall l\ h\ t.\ (l = h::t) \wedge \texttt{List}\ t \implies \texttt{List}\ l\end{array}$$

Besides the rules, there's also a "case theorem" generated with the relation definition. This theorem is also the same for both `List` and `coList`, because it just expressed the same rules from the backward:

$$\vdash \texttt{List}\ a_0 \iff (a_0 = [\,]) \vee \exists h\ t.\ (a_0 = h::t) \wedge \texttt{List}\ t$$

The only differences between the two relations are the following theorems:

```
List_ind:
 ⊢ (∀ l. (l = []) ⟹ List′ l) ∧
   (∀ l h t. (l = h::t) ∧ List′ t ⟹ List′ l) ⟹
   ∀ a0. List a0 ⟹ List′ a0
coList_coind:
 ⊢ (∀ a0.
      coList′ a0 ⟹ (a0 = []) ∨ ∃ h t. (a0 = h::t) ∧ coList′ t) ⟹
   ∀ a0. coList′ a0 ⟹ coList a0
```

From the shapes of above two theorems, it's not hard to see that, the purpose of the induction theorem for `List` is to restrict it to the least possible closure, while the purpose of the co-induction theorem for `coList` is to restrict it to the greatest possible closure. Thus, for each defined relation, their characteristic can be fully decided by three generated theorems:

1. The original rules;

2. The same rules expressed from backward;

3. The induction or co-induction theorem.

Most of time, it's necessary to define inductive relation, because the induction theorem is necessary to prove many important results, and those results won't be proved if the same rules were defined co-inductively. And if for some reasons the inductive and co-inductive definitions coincide, then in theory both induction co-induction theorems can be used to prove other results. (although they cannot be generated together)

What's more intersting is the following game: for a specific relation defined from a group of rules, if either induction theorem or co-induction theorem were never used to prove any results in the whole theory, we want to know, if the relation *becomes larger*, when switching from the an inductive defintion to co-inductive defintion. For above List examples, such changes didn't make the resulting relation any larger, since we can prove their equivalence.

2.3 Writing proofs in HOL

In this section, we use some examples to illustrate the general techniques for proving theorems in HOL. CCS syntax and SOS (Structural Operational Semantics) rules appeared in this section will be re-introduced formally in next chapter.

2.3.1 Forward and backward proofs

For simple derivations (as proofs of CCS transition theorems) that we already know the "proof", to formally represent the proof in HOL4, we can directly *construct* the proof in forward way.

$$(\text{Sum}_1)\,\dfrac{(\text{Pref})\,\dfrac{}{a.b.0 \xrightarrow{a} b.0}}{a.b.0 + b.a.0 \xrightarrow{a} b.0}$$

To proof the derivation theorem:

```
⊢ In “a”..In “b”..nil + In “b”..In “a”..nil
  −In “a”→
  In “b”..nil
```

the most fundamental way is to use HOL4’s standard drivation rule (drule) `ISPEC` and primitive inference rule `MP`.

In SOS rule `PREFIX`, if we specialize the universally quantified variable E to $b.0$, we get a new theorem saying $\forall u.u.b.0 \xrightarrow{u} b.0$:

```
> ISPEC ‘‘prefix (label (name "b")) nil‘‘ PREFIX;
val it =
   |- !u. u..label (name "b")..nil --u-> label (name "b")..nil:
   thm
```

Doing this again on the rest universally quantified variable u, specialized to Action a, then we get again a new theorem $a.b.0 \xrightarrow{a} b.0$ (and saved into a variable `t1`):

```
> val t1 = ISPEC ‘‘label (name "a")‘‘
                 (ISPEC ‘‘prefix (label (name "b")) nil‘‘ PREFIX);
val t1 =
   |- label (name "a")..label (name "b")..nil
   --label (name "a")->
   label (name "b")..nil:
   thm
```

Now we want to use `SUM1` to reach the final theorem. There two ways to do this. The first way is to manually specialize all the four universally quantified variables in the theorem. To make this work easier, HOL4 has provided the drule `ISPECL`, which takes a list of terms and a theorem, internally it calls `ISPEC` repeatedly on each universally quantified variables in the theorem:

```
> val t2 = ISPECL [‘‘prefix (label (name "a"))
                          (prefix (label (name "b")) nil)‘‘,
                   ‘‘label (name "a")‘‘,
                   ‘‘prefix (label (name "b")) nil‘‘,
                   ‘‘prefix (label (name "b"))
                          (prefix (label (name "a")) nil)‘‘]
                  SUM1;
val t2 =
   |- label (name "a")..label (name "b")..nil
   --label (name "a")->
   label (name "b")..nil
  ==>
   label (name "a")..label (name "b")..nil +
   label (name "b")..label (name "a")..nil
   --label (name "a")->
   label (name "b")..nil:
   thm
```

Now if we see theorem `t1` as A, then `t2` looks like $A \Rightarrow B$. Now we're ready to use HOL4's primitive inference rule `MP` (Modus Ponens) to get B from A and $A \Rightarrow B$:

```
> MP t2 t1;
val it =
   |- label (name "a")..label (name "b")..nil +
   label (name "b")..label (name "a")..nil
   --label (name "a")->
   label (name "b")..nil:
   thm
```

This is exactly the target theorem we wanted to prove (or verify). Putting above code together, we can write down the following code piece in Standard ML as the procedure to construct out the theorem:

```
local
    val t1 = ISPEC ``label (name "a")``
                  (ISPEC ``prefix (label (name "b")) nil`` PREFIX)
    and t2 = SPECL [``prefix (label (name "a"))
                                  (prefix (label (name "b")) nil)``,
                    ``label (name "a")``,
                    ``prefix (label (name "b")) nil``,
                    ``prefix (label (name "b"))
                                  (prefix (label (name "a")) nil)``]
                   SUM1;
in
    val ex1 = save_thm ("ex1", MP t2 t1)
end;
```

In theory, any formalized theorem in HOL4 can be constructed in this primitive way manually. However it's quite inconvenient to supply large amount of specialized parameters to build theorems like above `t2`. A slightly smarter way is to use `MATCH_MP` directly on `SUM` and `t1`, HOL4's drule `MATCH_MP` will do Modus Ponens with automatic matching:

```
> val t2 = MATCH_MP SUM1 t1;
val t2 =
   |- !E'.
     label (name "a")..label (name "b")..nil + E'
     --label (name "a")->
     label (name "b")..nil:
   thm
```

This theorem looks almost the same as the target theorm, except for the universally quantified variable E', which takes any CCS term. Now the only thing left is to specialize it to $b.a.0$:

```
> ISPEC ``prefix (label (name "b"))
                 (prefix (label (name "a")) nil)`` t2;
val it =
   |- label (name "a")..label (name "b")..nil +
   label (name "b")..label (name "a")..nil
   --label (name "a")->
   label (name "b")..nil:
   thm
```

The third way to prove this simple theorem is to prove it from backward. In other words, we first put the final theorem as a *goal* to prove, then we apply possible theorems to reduce the goal to smaller (easier) sub-goals, until we arrive to basic logic facts.

Using HOL4's interactive proof management facility, we can just write down the target theorem and use command "g" to put it into the proof manager as the initial goal:

```
> g `TRANS (sum (prefix (label (name "a"))
                                 (prefix (label (name "b")) nil))
               (prefix (label (name "b"))
                        (prefix (label (name "a")) nil)))
          (label (name "a"))
          (prefix (label (name "b")) nil)`;
val it =
   Proof manager status: 1 proof.
1. Incomplete goalstack:
     Initial goal:

     label (name "a")..label (name "b")..nil +
     label (name "b")..label (name "a")..nil
     --label (name "a")->
     label (name "b")..nil
:
   proofs
```

To finish the proof, we need to apply the so-called *tacticals*, which translates current goal into new sub-goals. A tactical can be considered as the reverse object of its correspond derivation rules (or primitive inference rules). From previous forward proof of the same theorem, we have known that rules `SUM1` and `PREFIX` must be used. For backward proofs, the order of applying them must be reverted too.

To benefit from `SUM1` (or other rules generated from `Hol_reln`), the key tactical here is `MATCH_MP_TAC`. For any goal A, if we know there's a theorem with forms like $\forall x.B \Rightarrow A$, then `MATCH_MP_TAC` could translated this goal into single sub-goal B. Now we apply this tactical using command "`e`":

```
> e (MATCH_MP_TAC SUM1);
OK..
```

```
1 subgoal:
val it =

label (name "a")..label (name "b")..nil
--label (name "a")->
label (name "b")..nil
:
   proof
```

As we expected, now the new goal is simplier: the outside `sum` has been removed by rule `SUM1`. Now we can see this new goal looks just like our axiom `PREFIX`, with universally quantified variables specialized to certain terms. In another word, it's exactly the same as previous intermediate theorem `t1`.

To finish the proof, we can either benefit from `t1`, using tactical `ACCEPT_TAC` which simply take a theorem and compare it with current goal, and when they're the same, the proof is finished:

```
> e (ACCEPT_TAC t1);
OK..

Goal proved.
|- label (name "a")..label (name "b")..nil
   --label (name "a")->
   label (name "b")..nil
val it =
   Initial goal proved.
|- label (name "a")..label (name "b")..nil +
   label (name "b")..label (name "a")..nil
   --label (name "a")->
   label (name "b")..nil:
   proof
```

Of course, if there's no theorem `t1`, we can also define it freshly using previous method (call `ISPEC` or `ISPECL` on `PREFIX`) and then apply it with `ACCEPT_TAC`. But there's another better way: we can ask HOL4 to try to *rewrite* current goal with `PREFIX` referenced. In this way, HOL4 will rewrite the goal into `T`, the logical truth, and the proof is also finished: (before trying this new way, we can use command "`b()`" to go back to last step)

```
> b();
val it =

label (name "a")..label (name "b")..nil
--label (name "a")->
label (name "b")..nil

:
   proof
> e (REWRITE_TAC [PREFIX]);
```

```
OK..

Goal proved.
|- label (name "a")..label (name "b")..nil
   --label (name "a")->
   label (name "b")..nil
val it =
   Initial goal proved.
|- label (name "a")..label (name "b")..nil +
   label (name "b")..label (name "a")..nil
   --label (name "a")->
   label (name "b")..nil:
   proof
```

Rewriting is one of the most common used proof techniques, and the tactical `REWRITE_TAC` is the most common used rewriting tactical. It can actually take a list of reference theorems, and it will repeatedly rewrite current goal until the rewriting process converge.[3] Usually these theorems are equation theorems like $A = B$, and whenever there's a A in the goal, it becomes B. In our case, `PREFIX` can be seen as $\forall E\ u.\ \ u..E\ -u\!\rightarrow\! E\ \iff$ `T`, so the rewriting result is `T`, the logical truth.

In HOL4, a formal proof is just a normal Standard ML code file, the proof script is just a single function call on `store_thm`. Above proof, although is done by interactive commands, finally we should write it down into a piece of code, and give the theorem a name. Here it is:

```
(* (a.b.0 + b.a.0) --a-> (b.0) *)
val ex1'' = store_thm ("ex1''",
  ``TRANS (sum (prefix (label (name "a"))
                       (prefix (label (name "b")) nil))
               (prefix (label (name "b"))
                       (prefix (label (name "a")) nil)))
          (label (name "a"))
          (prefix (label (name "b")) nil)``,
    MATCH_MP_TAC SUM1
 >> REWRITE_TAC [PREFIX]);
```

Here, the symbol "> >" is an abbrevation of HOL's tactical `THEN`. Sometimes we also use double-backslashs at the end of a line, it's also an abbrevation of `THEN`. The so-called "tacticals" have the type `tactic -> tactic`, they're for connecting and combining multiple tactics into a single big one. Thus in above code, the whole proof step is nothing but a single parameter to ML function `store_thm`.

Goal-directed proofs, when written carefully and friendly, even with some extra code comments, can be human-readable. Actually from above code, we can see clearly the following information:

1. The theorem name;

[3] Thus it's possible the process diverges and goes into infinite loops. In HOL4, there're many other rewriting tacticals with slightly different features.

2. The theorem contents;

3. The proof script, including the two key theorems (SUM1 and PREFIX).

Given such a formal proof, the convinciblity of the correctness of theorem comes from the following facts:

1. the theorem proving tools (HOL4 here) is a reliable software with minimal, verified logical kernel.

2. our definition of CCS inference rules are clear, simple, and same as in the textbook.

3. there's no other axiom defined and involved in the proving process.

4. by replaying the proof process (or simply watching it), in theory we can construct a corresponding *informal* proof on pencil and paper. (Thus the only purpose of using software is to finally get this *informal* proof)

The formal proof of (a.b.0 + b.a.0) -b-> (a.0) is similar, the only difference is to use SUM2 instead of SUM1:

```
(* (a.b.0 + b.a.0) --b-> (a.0) *)
val ex2 = store_thm ("ex2",
  ``TRANS (sum (prefix (label (name "a"))
                       (prefix (label (name "b")) nil))
               (prefix (label (name "b"))
                       (prefix (label (name "a")) nil)))
          (label (name "b"))
          (prefix (label (name "a")) nil)``,
    MATCH_MP_TAC SUM2
 >> REWRITE_TAC [PREFIX]);
```

From now on, we'll only use backward proof techniques, and when necessary, we still have to construct intermediate theorems in forward way and use them directly as input of some tacticals.

2.3.2 Bigger examples

In last section we have shown that the tactical MATCH_MP_TAC is the main tool to benefit from the inference rules we defined for CCS. However just using MATCH_-MP_TAC is not enough to do all kinds of CCS/SOS derivations. Now we formalize a bigger derivation with more rules involved, and by showing the formal proof we introduce some other important tacticals.

$$\text{(Res)}\ \dfrac{\text{(Com)}\ \dfrac{\text{(Pref)}\ \dfrac{}{a.c.0 \xrightarrow{a} c.0} \qquad \text{(Sum}_1\text{)}\ \dfrac{\text{(Pref)}\ \dfrac{}{\bar{a}.0 \xrightarrow{\bar{a}} 0}}{\bar{a}.0 + c.0 \xrightarrow{\bar{a}} 0}}{a.c.0|\bar{a}.0 + c.0 \xrightarrow{\tau} c.0|0}}{(\nu c)(a.c.0|\bar{a}.0 + c.0) \xrightarrow{\tau} (\nu c)(c.0|0)}$$

Here is the formal proof of above derivations:

```
val ex3 = store_thm ("ex3",
  ``TRANS (restr { name "c" }
                (par (prefix (label (name "a"))
                             (prefix (label (name "c")) nil))
                     (sum (prefix (label (coname "a")) nil)
                          (prefix (label (name "c")) nil))))
          tau
          (restr { name "c" }
                (par (prefix (label (name "c")) nil) nil))``,
    MATCH_MP_TAC RESTR
 >> RW_TAC std_ss []
 >> MATCH_MP_TAC COM
 >> EXISTS_TAC ``name "a"``
 >> CONJ_TAC (* 2 sub-goals here *)
 >- REWRITE_TAC [PREFIX]
 >> MATCH_MP_TAC SUM1
 >> REWRITE_TAC [PREFIX, CCS_COMPL_def]);
```

When doing an informal proof for above theorem, it suffices to show that, inference rules `RESTR`, `COM`, and two `PREFIX`s must be used sequentially. However in the formal proof things are not that simple. Let's see what happened after the first tactical (`MATCH_MP_TAC RESTR`):

```
1 subgoal:
val it =

?l.
  label (name "a")..label (name "c")..nil ||
  (label (coname "a")..nil + label (name "c")..nil)
  --tau->
  label (name "c")..nil || nil
  /\
  ((tau = tau) \/
   (tau = label l) /\ ~(l IN {name "c"}) /\ ~(COMPL l IN {name "c"}))
:
   proof
```

Here, beside the CCS transition with outter `restr` removed, we also got an extra part starting with `(tau = tau)`. This is reasonable, because we can't freely remove the outter `restr`, unless the current transition action (τ here) wasn't restricted. There's also a bounded existential variable l, but we can safely ignore it in this case.

Before applying next rule `COM`, we must simplify the current goal and completely remove these extra parts. Fortunately the first part $\tau = \tau$ is always true, so the rest part in the term is not important any more. Here we actually need multiple basic logical theorems, but writing down the detailed proofs are not rewarding to people who is trying to understand the *essential* things in this proof. Thus we want HOL4 to simplify the current goal using all possible basic logic formulae, which we don't care about the details. The tactical `RW_TAC` with the so-called *simp set*, `std_ss` (standard simplification rule set) does exactly this trick. With this tactical we successfully simplified current goal into the desired one[4]:

```
> e (RW_TAC std_ss []);
OK..
1 subgoal:
val it =

label (name "a")..label (name "c")..nil ||
(label (coname "a")..nil + label (name "c")..nil)
--tau->
label (name "c")..nil || nil
:
   proof
```

Next tricky thing happens after applying the rule `COM`:

```
> e (MATCH_MP_TAC COM);
OK..
1 subgoal:
val it =

?l.
  label (name "a")..label (name "c")..nil
  --label l->
  label (name "c")..nil /\
  label (coname "a")..nil + label (name "c")..nil
  --label (COMPL l)->
  nil
:
   proof
```

Here we got an new goal with existential quatified variable l as the transition label. This is always the case when appling rule `COM`. If we take a closer look at rule `COM`

$$\begin{aligned}&\vdash\ E\ -\texttt{label}\ l\!\rightarrow\ E_1\ \land\ E'\ -\texttt{label (COMPL}\ l\texttt{)}\!\rightarrow\ E_2\ \Longrightarrow\\ &E\ \|\ E'\ -\tau\!\rightarrow\ E_1\ \|\ E_2\end{aligned}$$

we can see that, the two transition labels, l and $\bar{l}$, only appear in premiss of the rule. This means, if we went from the conclusion back to the premiss, we must make a guess about the transition labels. And for complex CCS terms, such a guess may required a deep look inside the remain terms. In the terminalogy of sequent calculus, the rule `COM` is actually a *cut*, which blocks the automatic inference process. When doing semi-automatic theorem proving, what we need is to make a choice and instantiate the existential variables. In current case, it's easy to see that, to make further simplification of current goals, we must choose the label l to be a. This is how the tactical `EXISTS_TAC` gets used:

[4] the 3rd empty list indicates no other extra special rewriting theorems.

```
> e (EXISTS_TAC ``name "a"``);
OK..
1 subgoal:
val it =

label (name "a")..label (name "c")..nil
--label (name "a")->
label (name "c")..nil
 /\
label (coname "a")..nil + label (name "c")..nil
--label (COMPL (name "a"))->
nil
:
   proof
```

Now we got a new sub-goal with forms like $A \wedge B$. But the part A we can easily prove with a simple rewriting using axiom `PREFIX`. To actually do this, we need to break the current goal into two sub-goals. the tactical `CONJ_TAC` does exactly this job:

```
> e (CONJ_TAC);
OK..
2 subgoals:
val it =

label (coname "a")..nil + label (name "c")..nil
--label (COMPL (name "a"))->
nil



label (name "a")..label (name "c")..nil
--label (name "a")->
label (name "c")..nil


2 subgoals
:
   proof
```

Noticed that, when multiple goals appears, the lowest goal on the screen is the current sub-goal. We already know how to prove this sub-goal:

```
> e (REWRITE_TAC [PREFIX]);
OK..

Goal proved.
|- label (name "a")..label (name "c")..nil
   --label (name "a")->
   label (name "c")..nil

Remaining subgoals:
val it =

label (coname "a")..nil + label (name "c")..nil
--label (COMPL (name "a"))->
nil

:
   proof
```

To prove the rest goal, first we need to use SUM1 to remove the right part of the sum:

```
> e (MATCH_MP_TAC SUM1);
OK..
1 subgoal:
val it =

label (coname "a")..nil --label (COMPL (name "a"))-> nil

:
   proof
```

Then we could do rewriting again using PREFIX, however there's one problem: the transition label is not the same as `label (coname "a")`, instead, it's `label (COMPL (name "a"))`. COMPL is a function for converting actions and their corresponding co-actions. It has the following definition:

```
(* Define the complement of a label, COMPL: Label -> Label. *)
val CCS_COMPL_def = Define `(COMPL (name s) = (coname s)) /\
                            (COMPL (coname s) = (name s))`;
```

Any definition is equtional theorem, therefore is accepted by HOL's rewriting system. Thus, to finish the whole proof, we need to rewrite the current goal using both PREFIX and CCS_COMPL_def:

```
> e (REWRITE_TAC [PREFIX, CCS_COMPL_def]);
OK..

Goal proved.
|- label (coname "a")..nil --label (COMPL (name "a"))-> nil

...

val it =
   Initial goal proved.
|- restr {name "c"}
     (label (name "a")..label (name "c")..nil ||
    (label (coname "a")..nil + label (name "c")..nil))
--tau->
restr {name "c"} (label (name "c")..nil || nil):
proof
```

With these new proving tecniques, we're ready to prove CCS transition theorems using any inference rules.

Chapter 3

Calculus of Communicating Systems

In this chapter we describe a formalization of Milner's *Calculus of Communicating Systems* in HOL theorem prover (HOL4). It covers basic definitions of CCS and its transition behaviors based on Structural Operational Semantics (SOS), the concepts and properties of strong bisimulation equivalence ($\sim$), weak bisimulation equivalence ($\approx$) and observational congruence ($\approx^c$, also called *rooted weak bisimulation equivalence*), together with their relationships. We have also formalized the Expansion Law, Hennessy Lemma, Deng Lemma and several versions of the "coarsest congruence contained in $\approx$" theorem.

The work was initially based on a porting of the old work by Monica Nesi using Hol88 theorem prover, then the author has made some modifications and improvements.

3.1 Labels and Actions

In most literature, there's no difference between *Actions* and *Labels*. A labeled transition systems (LTS for short) is a triple $TS = (Q, A, \rightarrow)$ in which A is the union of a countable set of input actions $\mathscr{L}$ and output actions (co-actions) $\overline{\mathscr{L}}$ plus a special invisible action $\tau \notin \mathscr{L} \cup \overline{\mathscr{L}}$.

In the formalization of CCS, however, it's better to have two distinct types: the type β `Label` (β is a type variable.) for visible actions, divided by input and output actions, and the type β `Action` is the union of all visible and invisible actions. This is a better approach because some constructions in CCS only accept visible actions as valid parameters, e.g. the restriction operator. And by having a dedicated type for just visible actions we can directly guarantee the exclusion of invisible actions (because otherwise it's a type mismatch).

In HOL4, the type β `Label` can be defined as a simple datatype (using HOL's datatype package) with a type variable β representing the underlying label types: (the first type variable α is reserved for other use)

```
Datatype ‘Label = name ’b | coname ’b‘;
```

Defined in this way, it's possible to precisely control the cardinality of labels available for the processes in question. In the earliest formalization by Monica Nesi, the type variable in above definition was forcedly instantiated with HOL's `string` type.[1] For practical purposes it's enough to use just strings as labels, but for large models generated automatically it's easier to use other label types, e.g. natural number (`num`).

Noticed that, in HOL every type must have at least one element. Thus no matter how "small" the type β is, there's at least two visible actions (one input and one output) available to the process. Some deep theorems (e.g. the full version of "coarsest congruence contained in $\approx$" must assume the existence of at least one visible action for the given processes, such assumptions can be removed when we're trying to formalize it. Our type β `Label` is inductive but not recursive. What's immediately available from above definitions, is some supporting theorems, for example: (We should keep in mind that, using a *datatype* is actually using its supporting theorems.)

```
Label_induction:
 ⊢ (∀b. P (name b)) ∧ (∀b. P (coname b)) ⟹ ∀L. P L
Label_nchotomy:
 ⊢ (∃b. LL = name b) ∨ ∃b. LL = coname b
Label_distinct:
 ⊢ name a ≠ coname a′
Label_11:
 ⊢ (∀a a′. (name a = name a′) ⟺ (a = a′)) ∧
   ∀a a′. (coname a = coname a′) ⟺ (a = a′)
```

The type β `Action` must contain all elements from the type β `Label`, plus the invisible action (τ). Previously this was defined as another simple datatype:

```
Datatype ‘Action = tau | label Label‘;
```

But recently the author has realized that, it's actually very natural to use HOL's optionTheory in the definition of Action, because the theory `option` defines a type operator `option` that 'lifts' its argument type, creating a type with all of the values of the argument and one other, specially distinguished value. For us, that 'specially distinguished value' is just τ. The new definition of Action type doesn't contain any new logical constants, it's nothing but a type abbreviation plus some overloading on existing constants:

```
type_abbrev ("Action", ‘‘:’b Label option‘‘);

overload_on ("tau",    ‘‘NONE :’b Action‘‘);
overload_on ("label",  ‘‘SOME :’b Label -> ’b Action‘‘);
```

By type instantiation of existing theorems provided by HOL's `optionTheory`, now we can easily get the following supporting theorems for the Action type:

[1] In her later developments for Value-Passing CCS ([11]) such a limitation was removed, and one more type variable γ was introduced for the type of indexing set.

Action	notation	HOL	HOL (alternative)
invisible action	τ	`tau`	τ
visible action	l	`label l`	`label` l
input action	a	`label (name "a")`	`In` "a"
output action	$\bar{a}$	`label (coname "a")`	`Out` "a"

Table 3.1: Actions used in CCS

```
Action_induction:
⊢ P τ ∧ (∀a. P (label a)) ⟹ ∀x. P x
Action_nchotomy:
⊢ (opt = τ) ∨ ∃x. opt = label x
Action_distinct:
⊢ τ ≠ label x
Action_11:
⊢ (label x = label y) ⟺ (x = y)
```

The main operation on the types β `Label` and β `Action` is `COMPL` for getting their complement actions: (for convenience we also define the complement of τ as itself)

```
⊢ (∀s. COMPL (name s) = coname s) ∧
  ∀s. COMPL (coname s) = name s
⊢ (∀l. COMPL (label l) = label (COMPL l)) ∧ (COMPL τ = τ)
```

As we know β `Label` and β `Action` are different types, the `COMPL` operator on them are actually overloaded operator of `COMPL_LAB` and `COMPL_ACT`, the complement operator for β `Label` and β `Action`.

The key theorem about β `Label` says that, doing complements twice for the same label gets the label itself:

```
COMPL_COMPL_LAB:
⊢ COMPL (COMPL l) = l
```

There's also a similar theorem for the double-complements of β `Action`.

Table 3.1 listed the notation of various actions:

3.1.1 Relabeling operator

In standard literature, Relabeling operator is usually defined as an unary substitution operator $\_[b/a]$, which takes a unary substitution b/a (hence, $a \neq b$), and a process p to construct a new process $p[b/a]$, whose semantics is that of p, where action $a(\bar{a})$ is turned into $b(\bar{b})$. And multi-label relabeling can be done by appending more unary substitution operators to the existing process after relabeling. The order of multiple relabelings is relevant, especially when new labels introduced in previous relabeling operation were further relabeled.

In our formalization, following the work of Monica Nesi, we support multi-label relabeling in single operation, and instead of using a list of substitutions, we have defined a new primitive type called "β `Relabeling`". β `Relabeling` is a is a bijection into a subset of functions β `Label` $\rightarrow\beta$ `Label`, which is called the *representation* of the type "β `Relabeling`". Not all functions of type "β `Label` $\rightarrow\beta$ `Label`" are valid representations of "β `Relabeling`", but only functions which satisfy the following property:

$$\vdash \texttt{Is\_Relabeling}\ f \iff \forall s.\ f\ (\texttt{coname}\ s) = \texttt{COMPL}\ (f\ (\texttt{name}\ s))$$

Noticed that, any identify function of type β `Label` $\rightarrow\beta$ `Label` also satisfies above property. Thus, beside specific substitutions we wanted, all relabeling functions are *total*: they must be able to handle all other labels too (just return the same label as input). (As we'll see later, such requirements could reduce the two rules for relabelling into just one).

But usually it's more convenient to represent relabeling functions as a list of substitutions of type (β `Label` $\times$ β `Label) list`. The operator `RELAB` can be used to define such a relabeling function. For instance, the term

`RELAB [(name "b",name "a"); (name "d",name "c")]`

can be used in place of a relabeling operator $[b/a, d/c]$, because its type is "β `Relabeling`". And it must be understood that, all relabeling functions are total functions: for all other labels except `a` and `c`, the substitution will be themselves (another way to express "No relabeling").

Finally, having the relabeling facility defined as a multi-label relabeling function and as part of CCS syntax, we can completely avoid the complexity of the Syntactic Substitution (c.f. p.171 of [4]) which has a complicated recursive definition[2] and heavily depends on some other recursive functions like $fn(\cdot)$ (free names) and $bn(\cdot)$ (bound names) for CCS processes (in our project, these functions are defined but not used).

3.2 The CCS Datatype

The core datatype "`(`α`,` β`) CCS`" in this formalization is defined as an inductive datatype in HOL, based on its Datatype package, as shown in Fig. 3.1.

Comparing with the original work, now the type of CCS processes has been extended with two type variables: α and β. α is the type of process constants, and β is the type of actions. In HOL, such a higher order type is represented as "`(`α`,` β`) CCS`". If both type variables were instantiated as `string`, the resulting type "`(string, string) CCS`" is equivalent with the `CCS` datatype in the old CCS formalization.

[2] However, syntactic relabeling is still considered as an "economic" way of doing relabeling, because having one native CCS operator will also introduce the corresponding SOS inference rules and equivalence laws.

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

Figure 3.1: CCS Datatype

As we have explained, there's no infinite sums: the sum operator is simply binary. Since the CCS datatype is inductively defined and has no infinite sum operator, it must be Finitary.

We have added some minimal grammar support using HOL's pretty printer, to represent CCS processes in more readable forms (this was not available in the old work). Table 3.2 has listed the notation of typical CCS processes and major operators supported by above definition:

Operator	**Notation**	**HOL**	**HOL (alternative)**
nil	$\mathbf{0}$	`nil`	`nil`
Rrefix	$a.b.0$	`prefix a (prefix b nil)`	a`..`b`..nil`
Sum	$p+q$	`sum p q`	p `+` q
Parallel	$p \mid q$	`par p q`	$p \parallel q$
Restriction	$(\nu L)p$	`restr L p`	$\nu\ L\ p$
Constant	$A = a.A$	`rec A (prefix a (var A))`	`rec` A `(`a`..var` A`)`

Table 3.2: Syntax of CCS operators

For Relabeling, as we described in the last section, to express "$p[b/a]$", it must be written as `relab` p `(RELAB [(name "b",name "a")])`, which is a little long literally.

For CCS processes defined by one or more constants, in our formalization in HOL4, all constants must be written into single term. (This is necessary for theorem proving, because otherwise there's no way to store all information into single variable in CCS-related theorems) The syntax for defining new constants is `rec` and the syntax to actually use a constant is `var`. To see how these operators are actually used, consider the following CCS process (the famous coffee machine model from [4]):

$$VM \overset{def}{=} coin.(\text{ask-esp}.VM_1 + \text{ask-am}.VM_2)$$

$$VM_1 \overset{def}{=} \overline{\text{esp-coffee}}.VM$$

$$VM_2 \overset{def}{=} \overline{\text{am-coffee}}.VM$$

In our formalization in HOL4, the above CCS process can be represented as the following single term:

```
‘‘rec "VM"
    (In "coin"
     ..
     (In "ask-esp" .. (rec "VM1" (Out "esp-coffee"..var "VM")) +
      In "ask-am" .. (rec "VM2" (Out "am-coffee"..var "VM"))))‘‘
```

That is, for the first time a new constant appears, use `rec` with the name of constants as string to “declare” it; when any constant appears again, use `var` to access it.

Finally, although not part of the formal definition, the **if-then-else** construct from value-passing CCS is automatically supported by HOL. This is because, for any boolean value b and two terms t_1 and t_2 of type α, the term **if** b **then** t_1 **else** t_2 has also the type α. Thus the conditional term can legally appears inside other CCS processes as a sub-process. We'll see in next section that it's necessary for handling transitions of CCS processes containing constants.

3.3 Transition Semantics

The transition semantics of CCS processes were defined by the following Structural Operational Semantics (SOS for short) rules:

$$(\text{Perf})\ \frac{}{\mu.p \xrightarrow{\mu} p} \qquad\qquad (\text{Par}_1)\ \frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q}$$

$$(\text{Rec})\ \frac{q[\texttt{rec}\ x.q\ /\ x] \xrightarrow{\mu} r}{\texttt{rec}\ x.q \xrightarrow{\mu} r} \qquad\qquad (\text{Par}_2)\ \frac{q \xrightarrow{\mu} q'}{p|q \xrightarrow{\mu} p|q'}$$

$$(\text{Sum}_1)\ \frac{p \xrightarrow{\mu} p'}{p+q \xrightarrow{\mu} p'} \qquad\qquad (\text{Par}_3)\ \frac{p \xrightarrow{\alpha} p' \qquad q \xrightarrow{\bar{\alpha}} q'}{p|q \xrightarrow{\tau} p'|q'}$$

$$(\text{Sum}_2)\ \frac{q \xrightarrow{\mu} q'}{p+q \xrightarrow{\mu} q'} \qquad\qquad (\text{Res})\ \frac{p \xrightarrow{\mu} p'}{(\nu a)p \xrightarrow{\mu} (\nu a)p'}\ \mu \neq a, \bar{a}$$

Besides, we have a rule for relabeling:

$$(\text{Rel})\ \frac{p \xrightarrow{\mu} p'}{p[f] \xrightarrow{f(\mu)} (p'[f]}$$

In some literatures [4], the rule Par$_3$ is called “Com” (communication), and the rule “Rec” (in a different form based on separated agent definitions) is also called “Cons” (constants). (Here we have preserved the rule names in the HOL88 work, because it's easier to locate for their names in the proof scripts.)

From the view of theorem prover (or just first-order logic), these inference rules are nothing but an *inductive definition* on 3-ary relation `TRANS` (with compact representation `-()->`) of type $(\alpha,\ \beta)$ `transition`, generated by HOL4's function `Hol_reln` [19]. Then we break them into separated theorems as primitive inference rules[3]:

`PREFIX:` $\vdash u\texttt{..}E\ -u\rightarrow\ E$

`REC:` $\vdash \texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X\ -u\rightarrow\ E_1 \Longrightarrow \texttt{rec}\ X\ E\ -u\rightarrow\ E_1$

`SUM1:` $\vdash E\ -u\rightarrow\ E_1 \Longrightarrow E\ \texttt{+}\ E'\ -u\rightarrow\ E_1$

`SUM2:` $\vdash E\ -u\rightarrow\ E_1 \Longrightarrow E'\ \texttt{+}\ E\ -u\rightarrow\ E_1$

`PAR1:` $\vdash E\ -u\rightarrow\ E_1 \Longrightarrow E \parallel E'\ -u\rightarrow\ E_1 \parallel E'$

`PAR2:` $\vdash E\ -u\rightarrow\ E_1 \Longrightarrow E' \parallel E\ -u\rightarrow\ E' \parallel E_1$

`PAR3:`

$$\begin{array}{l}\vdash E\ -\texttt{label}\ l\rightarrow\ E_1 \wedge E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\rightarrow\ E_2 \Longrightarrow \\ \quad E \parallel E'\ -\tau\rightarrow\ E_1 \parallel E_2\end{array}$$

`RESTR:`

$$\begin{array}{l}\vdash E\ -u\rightarrow\ E' \wedge \\ \quad ((u\ \texttt{=}\ \tau) \vee (u\ \texttt{=}\ \texttt{label}\ l) \wedge l \notin L \wedge \texttt{COMPL}\ l \notin L) \Longrightarrow \\ \quad \nu\ L\ E\ -u\rightarrow\ \nu\ L\ E'\end{array}$$

`RELABELING:`

$$\vdash E\ -u\rightarrow\ E' \Longrightarrow \texttt{relab}\ E\ rf\ -\texttt{relabel}\ rf\ u\rightarrow\ \texttt{relab}\ E'\ rf$$

Noticed that, in the rule `REC`, a recursive function `CCS_Subst` was used. It has the following definition which depends on the conditional clause (`if .. then .. else ..`):

$$\begin{array}{l}
\vdash (\forall E'\ X.\ \texttt{CCS\_Subst nil}\ E'\ X\ \texttt{=}\ \texttt{nil}) \wedge \\
\quad (\forall u\ E\ E'\ X.\ \texttt{CCS\_Subst}\ (u\texttt{..}E)\ E'\ X\ \texttt{=}\ u\texttt{..CCS\_Subst}\ E\ E'\ X) \wedge \\
\quad (\forall E_1\ E_2\ E'\ X. \\
\qquad \texttt{CCS\_Subst}\ (E_1\ \texttt{+}\ E_2)\ E'\ X\ \texttt{=} \\
\qquad \texttt{CCS\_Subst}\ E_1\ E'\ X\ \texttt{+}\ \texttt{CCS\_Subst}\ E_2\ E'\ X) \wedge \\
\quad (\forall E_1\ E_2\ E'\ X. \\
\qquad \texttt{CCS\_Subst}\ (E_1 \parallel E_2)\ E'\ X\ \texttt{=} \\
\qquad \texttt{CCS\_Subst}\ E_1\ E'\ X \parallel \texttt{CCS\_Subst}\ E_2\ E'\ X) \wedge \\
\quad (\forall L\ E\ E'\ X. \\
\qquad \texttt{CCS\_Subst}\ (\nu\ L\ E)\ E'\ X\ \texttt{=}\ \nu\ L\ (\texttt{CCS\_Subst}\ E\ E'\ X)) \wedge \\
\quad (\forall E\ f\ E'\ X. \\
\qquad \texttt{CCS\_Subst}\ (\texttt{relab}\ E\ f)\ E'\ X\ \texttt{=} \\
\qquad \texttt{relab}\ (\texttt{CCS\_Subst}\ E\ E'\ X)\ f) \wedge \\
\quad (\forall Y\ E'\ X. \\
\qquad \texttt{CCS\_Subst}\ (\texttt{var}\ Y)\ E'\ X\ \texttt{=}\ \textbf{if}\ Y\ \texttt{=}\ X\ \textbf{then}\ E'\ \textbf{else}\ \texttt{var}\ Y) \wedge
\end{array}$$

[3] They're considered as the axioms in our logic system, however they're not defined directly as axioms. HOL makes sure in such cases the logic system is still consistent.

$$\begin{array}{l}
\forall Y\ E\ E'\ X.\\
\quad \texttt{CCS\_Subst (rec}\ Y\ E\texttt{)}\ E'\ X\ \texttt{=}\\
\quad \textbf{if}\ Y\ \texttt{=}\ X\ \textbf{then}\ \texttt{rec}\ Y\ E\ \textbf{else}\ \texttt{rec}\ Y\ \texttt{(CCS\_Subst}\ E\ E'\ X\texttt{)}
\end{array}$$

The idea of `CCS_Subst` is very close to the variable substitution for λ-terms: it only replaced those *free variables* with the same name as the input variable.

In HOL4, any inductive relation defined by command `Hol_reln` will return with three (well, actually four) theorems: 1) the rules, 2) the induction (and strong induction) theorem and 3) the "cases" theorem. Only with all these theorems, the relation can be precisely defined. For example, to prove certain CCS transitions are impossible, the following long "cases" theorem (which asserts that the relation is a fixed point) must be used:

$$\begin{array}{l}
\vdash a_0\ {-a_1\rightarrow}\ a_2 \iff\\
\quad (a_0\ \texttt{=}\ a_1\texttt{..}a_2) \lor (\exists E\ E'.\ (a_0\ \texttt{=}\ E\ \texttt{+}\ E') \land E\ {-a_1\rightarrow}\ a_2) \lor\\
\quad (\exists E\ E'.\ (a_0\ \texttt{=}\ E'\ \texttt{+}\ E) \land E\ {-a_1\rightarrow}\ a_2) \lor\\
\quad (\exists E\ E_1\ E'.\ (a_0\ \texttt{=}\ E \parallel E') \land (a_2\ \texttt{=}\ E_1 \parallel E') \land E\ {-a_1\rightarrow}\ E_1) \lor\\
\quad (\exists E\ E_1\ E'.\ (a_0\ \texttt{=}\ E' \parallel E) \land (a_2\ \texttt{=}\ E' \parallel E_1) \land E\ {-a_1\rightarrow}\ E_1) \lor\\
\quad (\exists E\ l\ E_1\ E'\ E_2.\\
\qquad (a_0\ \texttt{=}\ E \parallel E') \land (a_1\ \texttt{=}\ \tau) \land (a_2\ \texttt{=}\ E_1 \parallel E_2) \land\\
\qquad E\ {-\texttt{label}\ l\rightarrow}\ E_1 \land E'\ {-\texttt{label (COMPL}\ l\texttt{)}\rightarrow}\ E_2) \lor\\
\quad (\exists E\ E'\ l\ L.\\
\qquad (a_0\ \texttt{=}\ \nu\ L\ E) \land (a_2\ \texttt{=}\ \nu\ L\ E') \land E\ {-a_1\rightarrow}\ E' \land\\
\qquad ((a_1\ \texttt{=}\ \tau) \lor (a_1\ \texttt{=}\ \texttt{label}\ l) \land l \notin L \land \texttt{COMPL}\ l \notin L)) \lor\\
\quad (\exists E\ u\ E'\ rf.\\
\qquad (a_0\ \texttt{=}\ \texttt{relab}\ E\ rf) \land (a_1\ \texttt{=}\ \texttt{relabel}\ rf\ u) \land\\
\qquad (a_2\ \texttt{=}\ \texttt{relab}\ E'\ rf) \land E\ {-u\rightarrow}\ E') \lor\\
\quad \exists E\ X.\ (a_0\ \texttt{=}\ \texttt{rec}\ X\ E) \land \texttt{CCS\_Subst}\ E\ \texttt{(rec}\ X\ E\texttt{)}\ X\ {-a_1\rightarrow}\ a_2
\end{array}$$

Here are some results proved using above "cases" theorem (i. e. they cannot be proved with only the SOS inference rules):

`NIL_NO_TRANS:` $\vdash \neg(\texttt{nil}\ {-u\rightarrow}\ E)$
`VAR_NO_TRANS:` $\vdash \neg(\texttt{var}\ X\ {-u\rightarrow}\ E)$

`TRANS_IMP_NO_NIL:`
$\vdash E\ {-u\rightarrow}\ E' \implies E \neq \texttt{nil}$

`TRANS_SUM_EQ:`
$\vdash E\ \texttt{+}\ E'\ {-u\rightarrow}\ E'' \iff E\ {-u\rightarrow}\ E'' \lor E'\ {-u\rightarrow}\ E''$

`TRANS_PAR_EQ:`

$$\begin{array}{l}
\vdash E \parallel E'\ {-u\rightarrow}\ E'' \iff\\
\quad (\exists E_1.\ (E''\ \texttt{=}\ E_1 \parallel E') \land E\ {-u\rightarrow}\ E_1) \lor\\
\quad (\exists E_1.\ (E''\ \texttt{=}\ E \parallel E_1) \land E'\ {-u\rightarrow}\ E_1) \lor\\
\quad \exists E_1\ E_2\ l.
\end{array}$$

$(u = \tau) \land (E'' = E_1 \parallel E_2) \land E$ −label l→ $E_1 \land$
E' −label (COMPL l)→ E_2

TRANS_RESTR_EQ:
$\vdash \nu\ L\ E\ -u\!\rightarrow\ E' \iff$
$\exists E''\ l.$
$(E' = \nu\ L\ E'') \land E\ -u\!\rightarrow\ E'' \land$
$((u = \tau) \lor (u =$ label $l) \land l \notin L \land$ COMPL $l \notin L)$

3.3.1 Decision procedure for CCS transitions

It's possible to use SOS inference rules and theorems derived from them for proving theorems about the transitions between any two CCS processes. However, what's more useful is the decision procedure which automatically decide all possible transitions and formally prove them.

For any CCS process, there is a decision procedure as a recursive function, which can completely decide all its possible (one-step) transitions. In HOL, this decision procedure can be implemented as a normal Standard ML function `CCS_TRANS_CONV` of type `term -> theorem`, the returned theorem fully characterize the possible transitions of the input CCS process.

For instance, we know that the process $(a.0|\bar{a}.0)$ have three possible transitions:

1. $(a.0|\bar{a}.0) \xrightarrow{a} (0|\bar{a}.0)$;
2. $(a.0|\bar{a}.0) \xrightarrow{\bar{a}} (a.0|0)$;
3. $(a.0|\bar{a}.0) \xrightarrow{\tau} (0|0)$.

To completely decide all possible transitions, if done manually, the following work should be done:

1. Prove there exists transitions from $(a.0|\bar{a}.0)$ (optionally);
2. Prove each of above three transitions using SOS inference rules;
3. Prove there's no other transitions, using the "cases" theorems generated from the `TRANS` relation.

Instead, if we use the function `CCS_TRANS_CONV` with the root process:

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

As the result, the following theorem is returned: From this theorem, we can see there're only three possible transitions and there's no others. Therefore it contains all information expressed by previous manually proved 5 theorems (in theory we can also try to manually prove this single theorem, but it's not easy since the steps required will be at least the sum of all previous proofs).

As a further example, if we put a restriction on label “a” and check the process $(\nu a)(a.0|\bar{a}.0)$ instead, there will be only one possible transition:

It’s possible to extract a list of possible transitions together with the actions, into a list. This work can be done automatically by the function `strip_trans`. Finally, if both the theorem and the list of transitions are needed, the function `CCS_TRANS` and its compact-form variant `CCS_TRANS’` can be used. For the previous example process $(a.0|\bar{a}.0)$, calling `CCS_TRANS’` on it in HOL’s interactive environment has the following results:

```
> CCS_TRANS ‘‘In "a"..nil || Out "a"..nil‘‘;
val it =
   (|- !u E.
     In "a"..nil || Out "a"..nil --u-> E <=>
     ((u = In "a") /\ (E = nil || Out "a"..nil) \/
      (u = Out "a") /\ (E = In "a"..nil || nil)) \/
     (u = tau) /\ (E = nil || nil),
    [(‘‘In "a"‘‘,
     ‘‘nil || Out "a"..nil‘‘),
     (‘‘Out "a"‘‘,
      ‘‘In "a"..nil || nil‘‘),
     (‘‘’t‘‘,
      ‘‘nil || nil‘‘)]):
   thm * (term * term) list
```

The main function `CCS_TRANS_CONV` is implemented in about 500 lines of Standard ML code, and it depends on many dedicated tactics written for CCS, and functions to access the internal structure of CCS-related theorem and terms. We have tried our best to make sure the correctness of this function, but certain bugs are still inevitable.[4] However, since it’s implemented in theorem prover, and the return value of this function is a theorem, what we can guarantee is the following things:

> Whenever the function terminates with a theorem returned, as long as the theorem has “correct” forms, the CCS transitions indicated in the returned theorem is indeed all possible transitions from the input process. No matter if there’re bugs in our program.

In another words, any remain bug in the program can only stop the whole function from returning a result, but as long as the result is returned, it cannot be wrong! This sounds like a different kind of trusted computing than the normal senses. In general, for any algorithm implemented in any normal programming languages, since the output is just a primitive value or data structure which can be arbitrary constructed or changed due to potential bugs in the software, the only way to trust these

[4] If the internal proof constructed in the function is wrong, then the function won’t return a theorem. But if the function successfully returns a theorem, the proof for this theorem must be correct, because there’s no other way to return a theorem except for correctly proving it in HOL theorem prover.

results, is to have the entire program carefully modeled and verified. But in our case, the Standard ML program code is not verified, but the result (once appears) can still be fully[5] trusted, simply because it's a theorem derived from HOL.

3.4 Strong equivalence

The concept of *bisimulation* and *bisimulation equivalence* (bisimilarity) and their variants have the central position in Concurrency Theory. One major approach in model checking is to check the bisimulation equivalence between the specification and implementation of the same system. Besides, it's well known that, strong equivalence as a relation, must be defined *co-inductively.* (And in fact, strong equivalence is one of the most well-studied co-inductive relation in computer science. [18]) In this section, we study the definition of strong and weak bisimulation and (bisimulation) equivalences, and their possible formalizations in HOL.

Recall the standard definition of strong bisimulation and strong equivalence (c.f. p.43 of [4]):

Definition 3.4.1. ((Strong) bisimulation and (strong) bisimulation equivalence) Let $TS = (Q, A, \rightarrow)$ be a transition system. A *bisimulation* is a relation $R \subset Q \times Q$ such that R and its inverse R^{-1} are both simulation relations. More explicitly, a bisimulation is a relation R such that if $(q_1, q_2) \in R$ then for all $\mu \in A$

- $\forall q_1'$ such that $q_1 \xrightarrow{\mu} q_1', \exists q_2'$ such that $q_2 \xrightarrow{\mu} q_2'$ and $(q_1', q_2') \in R$,
- $\forall q_2'$ such that $q_2 \xrightarrow{\mu} q_2', \exists q_1'$ such that $q_1 \xrightarrow{\mu} q_1'$ and $(q_1', q_2') \in R$.

Two states q and q' are *bisimular* (or *bisimulation equivalent*), denoted $q \sim q'$, if there exists a bisimulation R such that $(q, q') \in R$.

Noticed that, although above definition is expressed in LTS, it's also applicable to CCS in which each process has the semantic model as a rooted LTS. Given the fact that, all states involved in above definition are target states of direct or indirect transition of the initial pair of states, above definition can be directly used for CCS.

[5] Well, HOL theorem prover itself, like any other software, contains bugs for sure, but the chances for HOL to produce fake theorems are very very low, although such a chance is not zero in theory. If we went further in this direction, PolyML (which compiles HOL's ML code into binary executions) may also contain bugs, so even HOL's code is correct it could still produce fake theorems. The final solution should be using formally verified ML implementations like CakeML to build HOL and other ML-based theorem provers to completely eliminate such concerns, but so far it has't reached such a perfect level. Another way to convince the audience is that, HOL can also output the primitive reasoning steps (using those eight primitive rules) behind each theorem, through its *logging kernel* (and OpenTheory formats), in theory these steps can be verify by other HOL-family theorem provers or just by hands. My professor (Roberto Gorrieri) ever commented that I trusted HOL just like how he trusted Concurrency Workbench (CWB), I think this is not fair!

In HOL88, there's no direct way to define co-inductive relation. However, it's possible to follow above definition literally and define bisimulation first, then define the bisimulation equivalence on top of bisimulation. Here are the definitions translated from HOL88 to HOL4:

$$
\begin{array}{l}
\vdash \texttt{STRONG\_BISIM}\ \mathit{Bsm} \iff \\
\quad \forall E\ E'. \\
\quad\quad \mathit{Bsm}\ E\ E' \implies \\
\quad\quad \forall u. \\
\quad\quad\quad (\forall E_1.\ E\ {-u\rightarrow}\ E_1 \implies \exists E_2.\ E'\ {-u\rightarrow}\ E_2 \wedge \mathit{Bsm}\ E_1\ E_2) \wedge \\
\quad\quad\quad \forall E_2.\ E'\ {-u\rightarrow}\ E_2 \implies \exists E_1.\ E\ {-u\rightarrow}\ E_1 \wedge \mathit{Bsm}\ E_1\ E_2 \\
\vdash E \sim E' \iff \exists \mathit{Bsm}.\ \mathit{Bsm}\ E\ E' \wedge \texttt{STRONG\_BISIM}\ \mathit{Bsm}
\end{array}
$$

From the second definition, we can see that, $q \sim q'$ if there exists a bisimulation containing the pair (q, q'). This means that $\sim$ is the union of all bisimulations, i.e.,

$$\sim = \bigcup\{R \subset Q \times Q \colon R \text{ is a bisimulation}\}.$$

The other way to define strong equivalence is through the fixed point of the following function F: (c.f. p.72 of [4])

Definition 3.4.2. Given an LTS $(Q, A, \rightarrow)$, the function $F \colon \wp(Q \times Q) \rightarrow \wp(Q \times Q)$ (i.e., a transformer of binary relations over Q) is defined as follows. If $R \subset Q \times Q$, then $(q_1, q_2) \in F(R)$ if and only if for all $\mu \in A$

- $\forall q_1'$ such that $q_1 \xrightarrow{\mu} q_1'$, $\exists q_2'$ such that $q_2 \xrightarrow{\mu} q_2'$ and $(q_1', q_2') \in R$,
- $\forall q_2'$ such that $q_2 \xrightarrow{\mu} q_2'$, $\exists q_1'$ such that $q_1 \xrightarrow{\mu} q_1'$ and $(q_1', q_2') \in R$.

And we can see by comparing the definition of above function and the definition of bisimulation that (no formal proofs):

1. The function F is monotone, i.e. if $R_1 \subset R_2$ then $F(R_1) \subset F(R_2)$.
2. A relation $R \subset Q \times Q$ is a bisimulation if and only if $R \subset F(R)$.

Then according to Knaster-Tarski Fixed Point theorem, strong bisimilarity $\sim$ is the greatest fixed point of F. And this is also the definition of co-inductive relation defined by the same rules.

In HOL4, since the release Kananaskis-11, there's a new facility for defining co-inductive relation. The entry command is `Hol_coreln`, which has the same syntax as `Hol_reln` for definining inductive relations. Using `Hol_coreln`, it's possible to define the bisimulation equivalence *directly* in this way: (here we has chosen a new relation name `STRONG_EQ`)[6]

The first theorem is the original rules appearing in the definition. Roughly speaking, it's kind of rules for building a bisimulation relation in forward way, however

[6] Whenever ASCII-based HOL proof scripts were directly pasted, please understand the letter "`!`" as $\forall$, and "`?`" as $\exists$. They're part of HOL's term syntax. [20]

this is impossible because of the lack of base rules (which exists in most inductive relation). And it's not original in this case, since it can be derived from the last theorem `STRONG_EQ_cases` (RHS ⇒ LHS).

The second theorem is the co-induction principle. It says, for what ever relation which satisfies those rules, it must be contained in strong equivalence. In another word, it make sure the target relation is the maximal relation containing all others.

The purpose of the last theorem (also called "cases" theorem), is to make sure the target relation is indeed a fixed point of the function F built by the given rules. However, it doesn't give any information about the size of such a fixed point. In general, if the greatest fixed point and least fixed point doesn't coincide, without the restriction by co-induction theorem, the rest two theorems will not give a precise definition for that relation. For strong equivalence, we already know that, the least fixed point of F is empty relation $\emptyset$, and the great fixed point is the strong equivalence $\sim$. And in fact, the "cases" theorem has "defined" a relation which lies in the middle of the greatest and least fixed point. To see why this argument is true, we found this theorem as an equation could be used as a possible definition of strong equivalence: (c.f. p. 49 of [4])

Definition 3.4.3. Define *recursively* a new behavioral relation $\sim' \in Q \times Q$ as follows: $q_1 \sim' q_2$ *if and only if* for all $\mu \in A$

- $\forall q_1'$ such that $q_1 \xrightarrow{\mu} q_1', \exists q_2'$ such that $q_2 \xrightarrow{\mu} q_2'$ and $q_1' \sim' q_2'$,
- $\forall q_2'$ such that $q_2 \xrightarrow{\mu} q_2', \exists q_1'$ such that $q_1 \xrightarrow{\mu} q_1'$ and $q_1' \sim' q_2'$.

This is exactly the same as above "cases" theorem if the theorem were used as a definition of strong equivalence. Robin Milner calls this theorem the " property (*)" of strong equivalence. (c.f. p. 88 of [2]) But as Prof. Gorrieri's book [4] already told with examples: "this does not identify a unique relation, as many different relations satisfy this recursive definition.", and the fact that any mathematical (or logic) definitions must precisely specify the targeting object (unless the possible covered range itself is a targeting object).

But why the recursive definition failed to define a largest bisimulation (i.e. strong equivalence)? The textbooks didn't give a clear answer, but in the view of theorem proving, now it's quite clear: such a recursive definition can only restrict the target relation into the range of all fixed points, while it's the co-induction theorem who finally restricts the target relation to the greatest solution. Without any of them, the solution will not be unique (thus not a valid mathematical definition).

Based on the definition of `STRONG_EQUIV` and SOS inference rules for the `TRANS` relation, we have proved a large set of theorems concerning the strong equivalence of CCS processes. Below is a list of fundamental congruence theorems for strong equivalence:

`STRONG_EQUIV_SUBST_PREFIX:`
$\vdash E \sim E' \Longrightarrow \forall u.\ u..E \sim u..E'$

`STRONG_EQUIV_PRESD_BY_SUM:`
$\vdash E_1 \sim E_1' \land E_2 \sim E_2' \Longrightarrow E_1 \texttt{+} E_2 \sim E_1' \texttt{+} E_2'$

`STRONG_EQUIV_PRESD_BY_PAR:`
$\vdash E_1 \sim E_1' \land E_2 \sim E_2' \Longrightarrow E_1 \parallel E_2 \sim E_1' \parallel E_2'$

`STRONG_EQUIV_SUBST_RESTR:`
$\vdash E \sim E' \Longrightarrow \forall L.\ \nu\ L\ E \sim \nu\ L\ E'$

`STRONG_EQUIV_SUBST_RELAB:`
$\vdash E \sim E' \Longrightarrow \forall rf.\ \texttt{relab}\ E\ rf \sim \texttt{relab}\ E'\ rf$

Noticed that, the strong bisimulation equivalence is co-inductively defined, and two processes are strong equivalent if there's a bisimulation containing them. Thus, to prove two processes are strong equivalent, it's enough to find a bisimulation containing them. To prove the they're not strong equivalent, it's enough to try to construct a bisimulation starting from them and the proof is finished whenever the attempt fails. In any case, there's no need to do induction on the data type of involved CCS processes.

3.4.1 Algebraic Laws for strong equivalence

Here are the strong laws proved for the sum operator: (noticed that, the lack of some parentheses is because we have defined the sum and parallel operators as left-associative)

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

Not all above theorems are primitive (in the sense of providing a minimal axiomatization set for proving all other strong algebraic laws). The first several theorems must be proved by constructing bisimulation relations and then verifying the definitions of strong bisimulation and strong equivalence, and their formal proofs were written in goal-directed ways. Instead, the last three ones were all constructed in forward way by applications of previous proven algebraic laws, without directly using any SOS inference rules and the definition of strong equivalence. Such constructions were based on two useful ML functions `S_SYM` and `S_TRANS` which builds new strong laws from the symmetry and transitivity of strong equivalence:

```
(* Define S_SYM such that, when given a theorem A
   |- STRONG_EQUIV t1 t2,
   returns the theorem A |- STRONG_EQUIV t2 t1. *)
fun S_SYM thm = MATCH_MP STRONG_EQUIV_SYM thm;

(* Define S_TRANS such that, when given the theorems thm1 and
   thm2, applies
   STRONG_EQUIV_TRANS on them, if possible. *)
fun S_TRANS thm1 thm2 =
    if rhs_tm thm1 = lhs_tm thm2 then
       MATCH_MP STRONG_EQUIV_TRANS (CONJ thm1 thm2)
    else
       failwith
        "transitivity␣of␣strong␣equivalence␣not␣applicable";
```

For instance, to construct the proof of STRONG_SUM_MID_IDEMP, the following code was written:

```
(* STRONG_SUM_MID_IDEMP:
   |- !E E’. STRONG_EQUIV (sum (sum E E’) E) (sum E’ E)
 *)
val STRONG_SUM_MID_IDEMP = save_thm (
   "STRONG_SUM_MID_IDEMP",
    GEN ‘‘E: CCS‘‘
     (GEN ‘‘E’: CCS‘‘
       (S_TRANS
        (SPEC ‘‘E: CCS‘‘
         (MATCH_MP STRONG_EQUIV_SUBST_SUM_R
          (SPECL [‘‘E: CCS‘‘, ‘‘E’: CCS‘‘] STRONG_SUM_COMM)))
        (S_TRANS
         (SPECL [‘‘E’: CCS‘‘, ‘‘E: CCS‘‘, ‘‘E: CCS‘‘]
                         STRONG_SUM_ASSOC_R)
         (SPEC ‘‘E’: CCS‘‘
          (MATCH_MP STRONG_EQUIV_SUBST_SUM_L
           (SPEC ‘‘E: CCS‘‘ STRONG_SUM_IDEMP)))))));
```

Here are the strong laws we have proved for the par operator:

STRONG_PAR_IDENT_R:	$\vdash E \parallel \texttt{nil} \sim E$
STRONG_PAR_COMM:	$\vdash E \parallel E' \sim E' \parallel E$
STRONG_PAR_IDENT_L:	$\vdash \texttt{nil} \parallel E \sim E$
STRONG_PAR_ASSOC:	$\vdash E \parallel E' \parallel E'' \sim E \parallel (E' \parallel E'')$

STRONG_PAR_PREF_TAU:
$\vdash u..E \parallel \tau..E' \sim u..(E \parallel \tau..E') + \tau..(u..E \parallel E')$

STRONG_PAR_TAU_PREF:
$\vdash \tau..E \parallel u..E' \sim \tau..(E \parallel u..E') + u..(\tau..E \parallel E')$

STRONG_PAR_TAU_TAU:

```
⊢ τ..E ∥ τ..E′ ∼ τ..(E ∥ τ..E′) + τ..(τ..E ∥ E′)

STRONG_PAR_PREF_NO_SYNCR:
 ⊢ l ≠ COMPL l′ ⟹
   ∀E E′.
     label l..E ∥ label l′..E′ ∼
     label l..(E ∥ label l′..E′) + label l′..(label l..E ∥ E′)

STRONG_PAR_PREF_SYNCR:
 ⊢ (l = COMPL l′) ⟹
   ∀E E′.
     label l..E ∥ label l′..E′ ∼
     label l..(E ∥ label l′..E′) +
     label l′..(label l..E ∥ E′) + τ..(E ∥ E′)
```

And the strong laws for the restriction operator:

```
STRONG_RESTR_NIL:           ⊢ ν L nil ∼ nil
STRONG_RESTR_SUM:           ⊢ ν L (E + E′) ∼ ν L E + ν L E′
STRONG_RESTR_PREFIX_TAU:    ⊢ ν L (τ..E) ∼ τ..ν L E

STRONG_RESTR_PR_LAB_NIL:
 ⊢ l ∈ L ∨ COMPL l ∈ L ⟹ ∀E. ν L (label l..E) ∼ nil

STRONG_RESTR_PREFIX_LABEL:
 ⊢ l ∉ L ∧ COMPL l ∉ L ⟹
   ∀E. ν L (label l..E) ∼ label l..ν L E
```

The strong laws for the relabeling operator:

```
STRONG_RELAB_NIL:
 ⊢ relab nil rf ∼ nil

STRONG_RELAB_SUM:
 ⊢ relab (E + E′) rf ∼ relab E rf + relab E′ rf

STRONG_RELAB_PREFIX:
 ⊢ relab (u..E) (RELAB labl) ∼
   relabel (RELAB labl) u..relab E (RELAB labl)
```

The strong laws for the recursion operator (for constants):

```
STRONG_UNFOLDING:
 ⊢ rec X E ∼ CCS_Subst E (rec X E) X

STRONG_PREF_REC_EQUIV:
 ⊢ u..rec s (v..u..var s) ∼ rec s (u..v..var s)
```

```
STRONG_REC_ACT2:
 ⊢ rec s (u..u..var s) ∼ rec s (u..var s)
```

All above three theorems for recursion operator were fundamental (in the sense that, they cannot be proved by just using other strong laws).

Finally, all above strong laws could be used either manually or as part of the decision procedure for automatically deciding strong equivalences between two CCS process. However such a decision procedure is not done in the current project.

3.4.2 The Strong Expansion Law

Another big piece of proof work in this project is the representation and proof of the following *expansion law* (sometimes also called the *interleaving law*:

Proposition 3.4.1. *(Expansion Law) Let* $p = \sum_{i=1}^{n} \mu_i.p_i$ *and* $q = \sum_{j=1}^{m} \mu'_j.q_j$*. Then*

$$p|q \sim \sum_{i=1}^{n} \mu_i.(p_i|q) + \sum_{j=1}^{m} \mu'_j.(p|q_j) + \sum_{i,j:\overline{\mu_i}=\mu'_j} \tau.(p_i|q_j) \tag{3.1}$$

Some characteristics made the formal proof very special and different from all other theorems that we have proved so far. First of all, arithmetic numbers (of type `num`) were involved for the first time, and now our CCS theory depends on elementary mathematical theories provided by HOL, namely the `prim_recTheory` and `arithmeticTheory`. Although arithmetic operations like $+, -, \cdot, /$ were not involved (yet), but we do need to compare number values and use some related theorems.

Also two CCS accessors were defined and used to access the internal structure of CCS processes, namely `PREF_ACT` for getting the initial action and `PREF_PROC` for getting the rest of process without the first action. Together there's predicate `Is_Prefix` for testing if a CCS is a prefixed process:

```
 ⊢ PREF_ACT (u..E) = u
 ⊢ PREF_PROC (u..E) = E
 ⊢ Is_Prefix E ⟺ ∃u E'. E = u..E'
```

They are needed because we're going to represent $\mu_i.p_i$ as the value of a function: $f(i)$ in which f has the type `num →(α, β) CCS`. And in this way, to get μ_i and p_i we have to use accessors: "`PREF_ACT (f i)`" and "`PREF_PROC (f i)`".

The next job is to represent a finite sum of CCS processes. This is done by the following recursive function `SIGMA`:

```
SIGMA f 0 = f 0
SIGMA f (SUC n) = SIGMA f n + f (SUC n)
```

Thus if there's a function f of type num $\rightarrow$(α, β) CCS, we should be able to represent $\sum_{i=1}^{n} f(i)$ by HOL term "SIGMA f n".

Now if we took a deeper look at the last summation of the right side of the expansion law, i.e. $\sum_{i,j:\overline{\mu_i}=\mu'_j} \tau.(p_i|q_j)$, we found that such a "sum" cannot be represented directly, because there're two index i, j and their possible value pairs used in the sum depends on the synchronization of corresponding actions from each p_i and q_j. What we actually need is a recursively defined function taking all the p_i and q_j and return the synchronized process in forms like $\sum \tau.(p_i|q_j)$.

But this is still too complicated, instead we first define functions to synchronize just one process with another group of processes. This work is achieved by the function SYNC of type β Action $\rightarrow$(α, β) CCS $\rightarrow$(num $\rightarrow$(α, β) CCS) $\rightarrow$num $\rightarrow$(α, β) CCS:

```
⊢ (∀u P f.
      SYNC u P f 0 =
      if (u = τ) ∨ (PREF_ACT (f 0) = τ) then nil
      else if LABEL u = COMPL (LABEL (PREF_ACT (f 0))) then
        τ..(P ‖ PREF_PROC (f 0))
      else nil) ∧
   ∀u P f n.
     SYNC u P f (SUC n) =
     if (u = τ) ∨ (PREF_ACT (f (SUC n)) = τ) then
       SYNC u P f n
     else if
       LABEL u = COMPL (LABEL (PREF_ACT (f (SUC n))))
     then
       τ..(P ‖ PREF_PROC (f (SUC n))) + SYNC u P f n
     else SYNC u P f n
```

Then the synchronization of two group of processes can be further defined by another recursive function ALL_SYNC of type (num $\rightarrow$(α, β) CCS) $\rightarrow$num $\rightarrow$(num $\rightarrow$(α, β) CCS) $\rightarrow$nun

```
⊢ (∀f f′ m.
      ALL_SYNC f 0 f′ m =
      SYNC (PREF_ACT (f 0)) (PREF_PROC (f 0)) f′ m) ∧
   ∀f n f′ m.
     ALL_SYNC f (SUC n) f′ m =
     ALL_SYNC f n f′ m +
     SYNC (PREF_ACT (f (SUC n))) (PREF_PROC (f (SUC n))) f′ m
```

Some lemmas about SIGMA and the two synchronization functions were proved first:

```
SIGMA_TRANS_THM_EQ:
⊢ SIGMA f n −u→ E ⟺ ∃k. k ≤ n ∧ f k −u→ E
```

```
SYNC_TRANS_THM_EQ:
```

```
⊢ SYNC u P f m −v→ Q ⟺
   ∃j l.
     j ≤ m ∧ (u = label l) ∧
     (PREF_ACT (f j) = label (COMPL l)) ∧ (v = τ) ∧
     (Q = P ∥ PREF_PROC (f j))
```

```
ALL_SYNC_TRANS_THM_EQ:
⊢ ALL_SYNC f n f′ m −u→ E ⟺
   ∃k k′ l.
     k ≤ n ∧ k′ ≤ m ∧ (PREF_ACT (f k) = label l) ∧
     (PREF_ACT (f′ k′) = label (COMPL l)) ∧ (u = τ) ∧
     (E = PREF_PROC (f k) ∥ PREF_PROC (f′ k′))
```

Finally, we have proved the Expansion Law in the following form:

```
STRONG_EXPANSION_LAW:
⊢ (∀i. i ≤ n ⟹ Is_Prefix (f i)) ∧
   (∀j. j ≤ m ⟹ Is_Prefix (f′ j)) ⟹
   SIGMA f n ∥ SIGMA f′ m ∼
   SIGMA (λi. PREF_ACT (f i)..(PREF_PROC (f i) ∥ SIGMA f′ m))
     n +
   SIGMA (λj. PREF_ACT (f′ j)..(SIGMA f n ∥ PREF_PROC (f′ j)))
     m + ALL_SYNC f n f′ m
```

3.5 Weak transitions and the EPS relation

In this part, the main purpose is to define the weak equivalence co-inductively *first* and then prove the traditional definition (like STRONG_EQUIV) as a theorem. Using HOL's coinductive relation module (Hol_coreln), it's much easier to get the same set of theorems like those for strong equivalence. These works are not part of the old CCS formalization in Hol88.

There're multiple ways to define the concept of weak transitions used in the definition of weak bisimulation. In early approach like Milner's book, the first step is to define a EPS relation, which indicates that between two processes there's nothing but zero or more τ transitions. In HOL, this can be defined through a non-recursive inductive relation and the RTC (reflexive transitive closure) on top of it:

Definition 3.5.1. (EPS) For any two CCS processes $E, E' \in Q$, define relation $EPS \subseteq Q \times Q$ as the reflexive transitive closure (RTC) of single-τ transition between E and E' ($E \xrightarrow{\tau} E'$):[7]

```
⊢ EPS = (λE E′. E −τ→ E′)*
```

[7] In HOL4's relationTheory, the relation types is curried: instead of having the same type "α reln" as the math definition, it has the type "$\alpha \to \alpha \to$bool". And the star(*) notation is for defining RTCs.

Intuitively speaking, $E \overset{\epsilon}{\Rightarrow} E'$ (Math notion: $E \overset{\epsilon}{\Longrightarrow} E'$) means there're zero or more *tau*-transitions from p to q.

Sometimes it's necessary to consider different transition cases when $p \overset{\epsilon}{\Rightarrow} q$ holds, or induct on the number of *tau* transitions between p and q. With such a definition, beside the obvious reflexive and transitive properties, a large amount of "cases" and induction theorem already proved in HOL's `relationTheory` are immediately available to us:

Proposition 3.5.1. *(The "cases" theorem of the* `EPS` *relation)*

1. *Taking one τ-transition at left:* *[EPS_cases1]*

$$\vdash x \overset{\epsilon}{\Rightarrow} y \iff (x = y) \lor \exists u.\ x -\tau\rightarrow u \land u \overset{\epsilon}{\Rightarrow} y$$

2. *Taking one τ-transition at right:* *[EPS_cases2]*

$$\vdash x \overset{\epsilon}{\Rightarrow} y \iff (x = y) \lor \exists u.\ x \overset{\epsilon}{\Rightarrow} u \land u -\tau\rightarrow y$$

3. *Three cases of EPS transition:* *[EPS_cases]*

$$\vdash E \overset{\epsilon}{\Rightarrow} E' \iff E -\tau\rightarrow E' \lor (E = E') \lor \exists E_1.\ E \overset{\epsilon}{\Rightarrow} E_1 \land E_1 \overset{\epsilon}{\Rightarrow} E'$$

Proposition 3.5.2. *(The induction and strong induction principles of the* `EPS` *relation)*

1. *Induction from left:* *[EPS_ind]*

$$\vdash (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ x -\tau\rightarrow y \land P\ y\ z \Longrightarrow P\ x\ z) \Longrightarrow \\ \forall x\ y.\ x \overset{\epsilon}{\Rightarrow} y \Longrightarrow P\ x\ y$$

2. *Induction from right:* *[EPS_ind_right]*

$$\vdash (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ P\ x\ y \land y -\tau\rightarrow z \Longrightarrow P\ x\ z) \Longrightarrow \\ \forall x\ y.\ x \overset{\epsilon}{\Rightarrow} y \Longrightarrow P\ x\ y$$

3. *Strong induction from left:* *[EPS_strongind]*

$$\vdash (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ x -\tau\rightarrow y \land y \overset{\epsilon}{\Rightarrow} z \land P\ y\ z \Longrightarrow P\ x\ z) \Longrightarrow \\ \forall x\ y.\ x \overset{\epsilon}{\Rightarrow} y \Longrightarrow P\ x\ y$$

4. *Strong induction from right:* *[EPS_strongind_right]*

$$\vdash (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ P\ x\ y \land x \overset{\epsilon}{\Rightarrow} y \land y -\tau\rightarrow z \Longrightarrow P\ x\ z) \Longrightarrow \\ \forall x\ y.\ x \overset{\epsilon}{\Rightarrow} y \Longrightarrow P\ x\ y$$

5. *Induction from the middle:* *[EPS_INDUCT]*

$$\begin{aligned}&\vdash (\forall E\ E'.\ E\ -\tau\rightarrow\ E' \implies P\ E\ E') \land (\forall E.\ P\ E\ E) \land\\ &\quad (\forall E\ E_1\ E'.\ P\ E\ E_1 \land P\ E_1\ E' \implies P\ E\ E') \implies\\ &\quad \forall x\ y.\ x \overset{\epsilon}{\Rightarrow} y \implies P\ x\ y\end{aligned}$$

Then we define the weak transition between two CCS processes upon the `EPS` relation:

Definition 3.5.2. For any two CCS processes $E, E' \in Q$, define "weak transition" relation $\Longrightarrow \subseteq Q \times A \times Q$, where A can be τ or a visible action: $E \overset{a}{\Longrightarrow} E'$ if and only if there exists two processes E_1 and E_2 such that $E \overset{\epsilon}{\Longrightarrow} E_1 \overset{a}{\longrightarrow} E_2 \overset{\epsilon}{\Longrightarrow} E'$:

`WEAK_TRANS:`
$\vdash E\ =u\Rightarrow\ E' \iff \exists E_1\ E_2.\ E \overset{\epsilon}{\Rightarrow} E_1 \land E_1\ -u\rightarrow\ E_2 \land E_2 \overset{\epsilon}{\Rightarrow} E'$

Using above two definitions and the "cases" and induction theorems, a large amount of properties about `EPS` and `WEAK_TRANS` were proved:

Proposition 3.5.3. *(Properties of* `EPS` *and* `WEAK_TRANS`*)*

1. *Any transition also implies a weak transition:*

   $\vdash E\ -u\rightarrow\ E' \implies E\ =u\Rightarrow\ E'$ [TRANS_IMP_WEAK_TRANS]

2. *Weak τ-transition implies* `EPS` *relation:*

   $\vdash E\ =\tau\Rightarrow\ E_1 \iff \exists E'.\ E\ -\tau\rightarrow\ E' \land E' \overset{\epsilon}{\Rightarrow} E_1$ [WEAK_TRANS_TAU]

3. *τ-transition implies* `EPS` *relation:*

   $\vdash E\ -\tau\rightarrow\ E' \implies E \overset{\epsilon}{\Rightarrow} E'$ [TRANS_TAU_IMP_EPS]

4. *Weak τ-transition implies an τ transition followed by EPS transition:*

   `WEAK_TRANS_TAU_IMP_TRANS_TAU:`
   $\vdash E\ =\tau\Rightarrow\ E' \implies \exists E_1.\ E\ -\tau\rightarrow\ E_1 \land E_1 \overset{\epsilon}{\Rightarrow} E'$

5. `EPS` *implies τ-prefixed* `EPS`*:*

   $\vdash E \overset{\epsilon}{\Rightarrow} E' \implies \tau..E \overset{\epsilon}{\Rightarrow} E'$ [TAU_PREFIX_EPS]

6. *Weak τ-transition implies τ-prefixed weak: τ-transition:*

   $\vdash E\ =u\Rightarrow\ E' \implies \tau..E\ =u\Rightarrow\ E'$ [TAU_PREFIX_WEAK_TRANS]

7. *A weak transition wrapped by EPS transitions is still a weak transition:*

EPS_WEAK_EPS:
$\vdash E \overset{\epsilon}{\Rightarrow} E_1 \land E_1 =u\Rightarrow E_2 \land E_2 \overset{\epsilon}{\Rightarrow} E' \implies E =u\Rightarrow E'$

8. *A weak transition after a τ-transition is still a weak transition:*

   $\vdash E -\tau\rightarrow E_1 \land E_1 =u\Rightarrow E' \implies E =u\Rightarrow E'$ [TRANS_TAU_AND_WEAK]

9. *Any transition followed by an EPS transition becomes a weak transition:*

   $\vdash E -u\rightarrow E_1 \land E_1 \overset{\epsilon}{\Rightarrow} E' \implies E =u\Rightarrow E'$ [TRANS_AND_EPS]

10. *An EPS transition implies either no transition or a weak τ-transition:*

    $\vdash E \overset{\epsilon}{\Rightarrow} E' \implies (E = E') \lor E =\tau\Rightarrow E'$ [EPS_IMP_WEAK_TRANS]

11. *Two possible cases for the first step of a weak transition:*

    WEAK_TRANS_cases1:
    $\vdash E =u\Rightarrow E_1 \implies$
    $(\exists E'.\ E -\tau\rightarrow E' \land E' =u\Rightarrow E_1) \lor \exists E'.\ E -u\rightarrow E' \land E' \overset{\epsilon}{\Rightarrow} E_1$

12. *The weak transition version of SOS inference rule (Sum_1) and (Sum_2):*

    $\vdash E =u\Rightarrow E_1 \implies E + E' =u\Rightarrow E_1$ [WEAK_SUM1]
    $\vdash E =u\Rightarrow E_1 \implies E' + E =u\Rightarrow E_1$ [WEAK_SUM2]

3.6 Weak bisimulation equivalence

The concepts of weak bisimulation and weak bisimulation equivalence (a.k.a. observational equivalence) and their properties were used all over this project. Several basic results (Deng lemma, Hennessy lemma, "Coarsest congruence contained in weak equivalence") were formally proved in this project, they all talk about the relationship between weak bisimulation equivalence and rooted weak bisimulation equivalence (a.k.a. observational congruence, we'll use this shorted names in the rest of the paper). Also, since observational congruence is not recursively defined but relies on the definition of weak equivalence, the properties of weak equivalence were heavily used in the proof of properties of observational congruence.

On the other side, it's quite easy to derive almost all the algebraic laws for weak equivalence (and observational congruence) from strong algebraic laws, because strong equivalence implies weak equivalence (and also observational congruence). This also reflects the fact that, although strong equivalence and its algebraic laws were relative less useful in real-world model checking, they do have contributions for deriving more useful algebraic laws. And from the view of theorem proving it totally make sense: if we try to prove any algebraic law for weak equivalence *directly*, the proof will be quite long and difficult, and the handling of *tau*-transitions will be a common part in all these proofs. But if we use the strong algebraic laws as lemmas, the proofs were actually divided into two logical parts: one for handling the algebraic law itself, the other for handling the τ-transitions.

The definition of weak bisimulation is the same as in [4], except for the use of EPS in case of τ-transitions:

Definition 3.6.1. (Weak bisimulation)

$$\begin{array}{l}
\vdash \texttt{WEAK\_BISIM}\ \mathit{Wbsm} \iff \\
\quad \forall E\ E'. \\
\quad\quad \mathit{Wbsm}\ E\ E' \implies \\
\quad\quad (\forall l. \\
\quad\quad\quad (\forall E_1. \\
\quad\quad\quad\quad E\ \texttt{-label}\ l\texttt{->}\ E_1 \implies \\
\quad\quad\quad\quad \exists E_2.\ E'\ \texttt{=label}\ l\texttt{=>}\ E_2 \land \mathit{Wbsm}\ E_1\ E_2) \land \\
\quad\quad\quad \forall E_2. \\
\quad\quad\quad\quad E'\ \texttt{-label}\ l\texttt{->}\ E_2 \implies \exists E_1.\ E\ \texttt{=label}\ l\texttt{=>}\ E_1 \land \mathit{Wbsm}\ E_1\ E_2) \land \\
\quad\quad (\forall E_1.\ E\ \texttt{-}\tau\texttt{->}\ E_1 \implies \exists E_2.\ E' \overset{\epsilon}{\Rightarrow} E_2 \land \mathit{Wbsm}\ E_1\ E_2) \land \\
\quad\quad \forall E_2.\ E'\ \texttt{-}\tau\texttt{->}\ E_2 \implies \exists E_1.\ E \overset{\epsilon}{\Rightarrow} E_1 \land \mathit{Wbsm}\ E_1\ E_2
\end{array}$$

Weak bisimulation has some common properties:

Proposition 3.6.1. *Properties of weak bisimulation*

1. *The identity relation is a weak bisimulation:*

   $$\vdash \texttt{WEAK\_BISIM}\ (\lambda x\ y.\ x = y) \qquad \texttt{[IDENTITY\_WEAK\_BISIM]}$$

2. *The converse of a weak bisimulation is still a weak bisimulation:*

   $$\vdash \texttt{WEAK\_BISIM}\ \mathit{Wbsm} \implies \texttt{WEAK\_BISIM}\ \mathit{Wbsm}^T \quad \texttt{[IDENTITY\_WEAK\_BISIM]}$$

3. *The composition of two weak bisimulations is a weak bisimulation:*

   $$\begin{array}{l}\vdash \texttt{WEAK\_BISIM}\ \mathit{Wbsm}_1 \land \texttt{WEAK\_BISIM}\ \mathit{Wbsm}_2 \implies \\ \quad \texttt{WEAK\_BISIM}\ (\mathit{Wbsm}_2 \circ_r \mathit{Wbsm}_1) \qquad \texttt{[COMP\_WEAK\_BISIM]}\end{array}$$

4. *The union of two weak bisimulations is a weak bisimulation:*

   $$\begin{array}{l}\vdash \texttt{WEAK\_BISIM}\ \mathit{Wbsm}_1 \land \texttt{WEAK\_BISIM}\ \mathit{Wbsm}_2 \implies \\ \quad \texttt{WEAK\_BISIM}\ (\mathit{Wbsm}_1 \cup_r \mathit{Wbsm}_2) \qquad \texttt{[UNION\_WEAK\_BISIM]}\end{array}$$

There're two ways to define weak bisimulation equivalence in HOL4, one is to define it as the union of all weak bisimulations:

Definition 3.6.2. (Alternative definition of weak equivalence) For any two CCS processes E and E', they're *weak bisimulation equivalent* (or weak bisimilar) if and only if there's a weak bisimulation relation between E and E':

`WEAK_EQUIV`:
$\vdash\ E\ \approx\ E' \iff \exists\, Wbsm.\ Wbsm\ E\ E' \land \texttt{WEAK\_BISIM}\ Wbsm$

This was the definition used by Monica Nesi in Hol88 in which there was no direct support for defining co-inductive relations. The new method we have used in this project, is to use HOL4's new co-inductive relation defining facility `Hol_coreln` to define weak bisimulation equivalence:

```
val (WEAK_EQUIV_rules, WEAK_EQUIV_coind, WEAK_EQUIV_cases)
 = Hol_coreln `
    (!(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')`;
```

The disadvantage of this new method is that, the rules used in above definition actually duplicated the definition of weak bisimulation, while the advantage is that, HOL4 automatically proved an important theorem and returned it as the third return value of above definition. This theorem is also called "the property (*)" (in Milner's book [2]:

Proposition 3.6.2. *(The property (*) for weak bisimulation equivalence)*

`WEAK_PROPERTY_STAR`:

$$
\begin{aligned}
&\vdash a_0 \approx a_1 \iff \\
&\quad (\forall\, l. \\
&\qquad (\forall\, E_1.\ a_0 \ -\texttt{label}\ l\!\rightarrow\ E_1 \implies \exists\, E_2.\ a_1 \ =\!\texttt{label}\ l\!\Rightarrow\ E_2 \land E_1 \approx E_2) \land \\
&\qquad \forall\, E_2.\ a_1 \ -\texttt{label}\ l\!\rightarrow\ E_2 \implies \exists\, E_1.\ a_0 \ =\!\texttt{label}\ l\!\Rightarrow\ E_1 \land E_1 \approx E_2) \land \\
&\quad (\forall\, E_1.\ a_0 \ -\tau\!\rightarrow\ E_1 \implies \exists\, E_2.\ a_1 \overset{\epsilon}{\Rightarrow} E_2 \land E_1 \approx E_2) \land \\
&\quad \forall\, E_2.\ a_1 \ -\tau\!\rightarrow\ E_2 \implies \exists\, E_1.\ a_0 \overset{\epsilon}{\Rightarrow} E_1 \land E_1 \approx E_2
\end{aligned}
$$

It's known that, above property cannot be used as an alternative definition of weak equivalence, because it doesn't capture all possible weak equivalences. But it turns out that, for the proof of most theorems about weak bisimilarities this property is enough to be used as a rewrite rule in their proofs. And, if we had used the old method to define weak equivalence, it's quite difficult to prove above property (*). In previous project, the property (*) for strong equivalence was proved based on the old method, then in this project we have completely removed these code and now both strong and weak bisimulation equivalences were based on the new method. On the other side, the fact that Monica Nesi can define co-inductive relation without using `Hol_coreln` has shown that, the core HOL Logic doesn't need to be extended to support co-inductive relation, and all what `Hol_coreln` does internally is to use the existing HOL theorems to construct the related proofs.

Using the alternative definition of weak equivalence, it's quite simple to prove that, the weak equivalence is an equivalence relation:

Proposition 3.6.3. *(Weak equivalence is an equivalence relation)*

$\vdash$ `equivalence WEAK_EQUIV` [WEAK_EQUIV_equivalence]

or

$\vdash E \approx E$ [WEAK_EQUIV_REFL]
$\vdash E \approx E' \Longrightarrow E' \approx E$ [WEAK_EQUIV_SYM]
$\vdash E \approx E' \land E' \approx E'' \Longrightarrow E \approx E''$ [WEAK_EQUIV_TRANS]

The substitutability of weak equivalence under various CCS process operators were then proved based on above definition and property (*). However, as we know weak equivalence is not a congruence, in some of these substitutability theorems we must added extra assumptions on the processes involved, i.e. the stability of CCS processes:

Definition 3.6.3. (Stable processes (or agents)) A process (or agent) is said to be *stable* if there's no τ-transition coming from it's root:

$\vdash \texttt{STABLE}\ E \iff \forall u\ E'.\ E\ {-u\to}\ E' \Longrightarrow u \neq \tau$

Notice that, the stability of a CCS process doesn't imply the τ-free of all its sub-processes. Instead the definition only concerns on the first transition leading from the process (root).

Among other small lemmas, we have proved the following properties of weak bisimulation equivalence:

Proposition 3.6.4. *Properties of weak bisimulation equivalence)*

1. *Weak equivalence is substitutive under prefix operator:*

   $\vdash E \approx E' \Longrightarrow \forall u.\ u..E \approx u..E'$ [WEAK_EQUIV_SUBST_PREFIX]

2. *Weak equivalence of stable agents is preserved by binary summation:*

   $\vdash E_1 \approx E_1' \land \texttt{STABLE}\ E_1 \land \texttt{STABLE}\ E_1' \land E_2 \approx E_2' \land \texttt{STABLE}\ E_2 \land$
   $\texttt{STABLE}\ E_2' \Longrightarrow$
   $E_1 + E_2 \approx E_1' + E_2'$ [WEAK_EQUIV_PRESD_BY_SUM]

3. *Weak equivalence is preserved by guarded binary summation:*

   `WEAK_EQUIV_PRESD_BY_GUARDED_SUM:`
   $\vdash E_1 \approx E_1' \land E_2 \approx E_2' \Longrightarrow a_1\,.\,.\,E_1 \;\texttt{+}\; a_2\,.\,.\,E_2 \approx a_1\,.\,.\,E_1' \;\texttt{+}\; a_2\,.\,.\,E_2'$

4. *Weak equivalence of stable agents is substitutive under binary summation on the right:*

   `WEAK_EQUIV_SUBST_SUM_R:`
   $\vdash E \approx E' \land \texttt{STABLE}\ E \land \texttt{STABLE}\ E' \Longrightarrow \forall E''.\ E \;\texttt{+}\; E'' \approx E' \;\texttt{+}\; E''$

5. *Weak equivalence of stable agents is substitutive under binary summation on the left:*

   `WEAK_EQUIV_SUBST_SUM_L:`
   $\vdash E \approx E' \land \texttt{STABLE}\ E \land \texttt{STABLE}\ E' \Longrightarrow \forall E''.\ E'' \;\texttt{+}\; E \approx E'' \;\texttt{+}\; E'$

6. *Weak equivalence is preserved by parallel operator:*

   `WEAK_EQUIV_PRESD_BY_PAR:`
   $\vdash E_1 \approx E_1' \land E_2 \approx E_2' \Longrightarrow E_1 \parallel E_2 \approx E_1' \parallel E_2'$

7. *Weak equivalence is substitutive under restriction operator:*

   `WEAK_EQUIV_SUBST_RESTR:`
   $\vdash E \approx E' \Longrightarrow \forall L.\ \nu\ L\ E \approx \nu\ L\ E'$

8. *Weak equivalence is substitutive under relabelling operator:*

   `WEAK_EQUIV_SUBST_RELAB:`
   $\vdash E \approx E' \Longrightarrow \forall rf.\ \texttt{relab}\ E\ rf \approx \texttt{relab}\ E'\ rf$

Finally, we have proved that, strong equivalence implies weak equivalence:

Theorem 3.6.1. *(Strong equivalence implies weak equivalence)*

$\vdash E \sim E' \Longrightarrow E \approx E'$ `[STRONG_IMP_WEAK_EQUIV]`

Here we omit all the algebraic laws for weak equivalence, because they were all easily derived from the corresponding algebraic laws for strong equivalence, except for the following τ-law:

Theorem 3.6.2. *The τ-law for weak equivalence)*

$\vdash \tau\,.\,.\,E \approx E$ `[TAU_WEAK]`

3.7 Observational Congruence

The concept of rooted weak bisimulation equivalence (also namsed *observation congruence*) is an “obvious fix” to convert weak bisimulation equivalence into a congruence. Its definition is not recursive but based on the definition of weak equivalence:

Definition 3.7.1. (Observation congruence) Two CCS processes are observation congruence if and only if for any transition from one of them, there's a responding weak transition from the other, and the resulting two sub-processes are weak equivalence:

$$\begin{array}{l}\vdash\ E\ \approx^c\ E'\ \iff\\ \quad \forall\, u\,.\\ \qquad (\forall\, E_1\,.\ E\ -u\rightarrow\ E_1\ \Longrightarrow\ \exists\, E_2\,.\ E'\ =u\Rightarrow\ E_2\ \land\ E_1\ \approx\ E_2)\ \land\\ \qquad \forall\, E_2\,.\ E'\ -u\rightarrow\ E_2\ \Longrightarrow\ \exists\, E_1\,.\ E\ =u\Rightarrow\ E_1\ \land\ E_1\ \approx\ E_2 \qquad \texttt{[OBS\_CONGR]}\end{array}$$

By observing the differences between the definition of observation equivalence (weak equivalence) and congruence, we can see that, observation equivalence requires a little more: for each τ-transition from one process, the other process must response with at least one τ-transition. Thus what's immediately proven is the following two theorems:

Theorem 3.7.1. *(Observation congruence implies observation equivalence)*

$$\vdash\ E\ \approx^c\ E'\ \Longrightarrow\ E\ \approx\ E' \qquad \texttt{[OBS\_CONGR\_IMP\_WEAK\_EQUIV]}$$

Theorem 3.7.2. *(Observation equivalence on stable agents implies observation congruence)*

WEAK_EQUIV_STABLE_IMP_CONGR:
$\vdash\ E\ \approx\ E'\ \land\ \texttt{STABLE}\ E\ \land\ \texttt{STABLE}\ E'\ \Longrightarrow\ E\ \approx^c\ E'$

Surprisingly, it's not trivial to prove that, the observation equivalence is indeed an equivalence relation. The reflexivity and symmetry are trivial:

Proposition 3.7.1. *(The reflexivity and symmetry of observation congruence)*

$$\vdash\ E\ \approx^c\ E \qquad \texttt{[OBS\_CONGR\_REFL]}$$
$$\vdash\ E\ \approx^c\ E'\ \Longrightarrow\ E'\ \approx^c\ E \qquad \texttt{[OBS\_CONGR\_SYM]}$$

But the transitivity is hard to prove.[8] Our proof here is based on the following lemmas:

Lemma 3.7.1. *If two processes E and E' are observation congruence, then for any EPS transition coming from E, there's a corresponding EPS transition from E', and the resulting two subprocesses are weakly equivalent:*

[8] Actually it's not proven in the old work, the formal proofs that we did in this project is completely new.

`OBS_CONGR_EPS`:

$\vdash\ E \approx^c E' \Longrightarrow \forall E_1 .\ E \overset{\epsilon}{\Rightarrow} E_1 \Longrightarrow \exists E_2 .\ E' \overset{\epsilon}{\Rightarrow} E_2 \land E_1 \approx E_2$

Proof. By (right) induction[9] on the number of τ in the EPS transition of E. In the base case, there's no τ at all, the E transits to itself. And in this case E' can respond with itself, which is also an EPS transition:

$$\begin{tikzcd}
E \arrow[r, dotted, no head, "\approx^c"] \arrow[d, no head, "="] & E' \arrow[d, no head, "="] \\
E \arrow[r, dotted, no head, "\approx"] & E'
\end{tikzcd}$$

For the induction case, suppose the proposition is true for zero or more τ transitions except for the last step, that's, $\forall E, \exists E1, E2$, such that $E \overset{\epsilon}{\Rightarrow} E_1$, $E' \overset{\epsilon}{\Rightarrow} E_2$ and $E_1 \approx E_2$. Now by definition of weak equivalence, if $E_1 \ {-\tau\rightarrow} E_1'$ then there exists $E2'$ such that $E_2 \overset{\epsilon}{\Rightarrow} E_2'$ and $E_1' \approx E_2'$. Then by transitivity of EPS, we have $E' \overset{\epsilon}{\Rightarrow} E_2 \land E_2 \overset{\epsilon}{\Rightarrow} E_2' \Longrightarrow E' \overset{\epsilon}{\Rightarrow} E_2'$, thus E_2' is a valid response required by observation congruence:

$$\begin{tikzcd}
E \arrow[r, dotted, no head, "\approx^c"] \arrow[d, Rightarrow, "\epsilon"] & E' \arrow[d, Rightarrow, "\epsilon"] \arrow[dd, bend left, "\epsilon"] \\
\forall E_1 \arrow[r, dotted, no head, "\approx"] \arrow[d, "\tau"] & \forall E_2 \arrow[d, Rightarrow, "\epsilon"] \\
\forall E_1' \arrow[r, dotted, no head, "\approx"] & \exists E_2'
\end{tikzcd}$$

□ □

Lemma 3.7.2. *If two processes E and E' are observation congruence, then for any weak transition coming from E, there's a corresponding weak transition from E', and the resulting two subprocesses are weakly equivalent:*

$\vdash\ E \approx^c E' \Longrightarrow \forall u\ E_1 .\ E \ {=u\Rightarrow}\ E_1 \Longrightarrow \exists E_2 .\ E' \ {=u\Rightarrow}\ E_2 \land E_1 \approx E_2$

Proof. (sketch Consider the two cases when the action is τ or not τ. For all weak τ-transitions coming from E, the observation congruence requires that there's at least one τ following E' and the resulting two sub-processes, say E_1' and E_2 are weak equivalence. Then the desired responses can be found by using a similar existence lemma for weak equivalence:

$$\begin{tikzcd}
E \arrow[r, dotted, no head, "\approx^c"] \arrow[d, "\tau"] \arrow[dd, bend right, "\tau"'] & E' \arrow[d, Rightarrow, "\tau"] \arrow[dd, bend left, "\tau"] \\
\exists E_1' \arrow[r, dotted, no head, "\approx"] \arrow[d, Rightarrow, "\epsilon"] & \exists E_2 \arrow[d, Rightarrow, "\epsilon"] \\
\forall E_1 \arrow[r, dotted, no head, "\approx"] & \exists E_2'
\end{tikzcd}$$

[9] The induction theorem used here is `EPS_ind_right`.

For all the non-τ weak transitions from E, the proof follows from previous lemma and a similar existence lemma for weak equivalence. The following figure is a sketch for the proof of this case:

$$\begin{tikzcd}
E \arrow[r, dash, dotted, "\approx^c"] \arrow[d, Rightarrow, "\epsilon"] \arrow[ddd, bend right, "\forall L"'] & E' \arrow[d, Rightarrow, "\epsilon"] \arrow[ddd, bend left, "L"] \\
\exists E'_1 \arrow[r, dash, dotted, "\approx"] \arrow[d, "L"] & \exists E'_2 \arrow[d, Rightarrow, "L"] \\
\exists E_2 \arrow[r, dash, dotted, "\approx"] \arrow[d, Rightarrow, "\epsilon"] & \exists E''_2 \arrow[d, Rightarrow, "\epsilon"] \\
\forall E_1 \arrow[r, dash, dotted, "\approx"] & \exists E2'''
\end{tikzcd}$$

In the previous figure, the existence of E'_2 follows by previous lemma, the existence of E''_2 follows by the definition of weak equivalence, and the existence of E'''_2 follows by the next existence lemma of weak equivalence. □

The existence lemma for weak equivalences that we mentioned in previous proof is the following one:

Lemma 3.7.3. `EPS_TRANS_AUX`:

$\vdash E \overset{\epsilon}{\Rightarrow} E_1 \Longrightarrow$
$\quad \forall\, Wbsm\ E'.$
$\qquad \texttt{WEAK\_BISIM}\ Wbsm \wedge Wbsm\ E\ E' \Longrightarrow \exists\, E_2.\ E' \overset{\epsilon}{\Rightarrow} E_2 \wedge Wbsm\ E_1\ E_2$

Now we prove the transitivity of Observation Congruence ($\approx^c$):

Theorem 3.7.3. *(Transitivity of Observation Congruence)*

$\vdash E \approx^c E' \wedge E' \approx^c E'' \Longrightarrow E \approx^c E''$ `[OBS_CONGR_TRANS]`

Proof. Suppose $E \approx^c E'$ and $E' \approx^c E''$, we're going to prove $E \approx^c E''$ by checking directly the definition of observation congruence.

For any u and E_1 which satisfy $E\ {-u}{\rightarrow} E_1$, by definition of observation congruence, there exists E_2 such that $E'\ {=u}{\Rightarrow} E_2$ with $E_1 \approx E_2$. By above Lemma 2, there exists another E_3 such that $E''\ {=u}{\Rightarrow} E_3$ with $E_2 \approx E_3$. By the already proven transitivity of weak equivalence, $E_1 \approx E_3$, thus E_3 is the required process which satisfies the definition of observation congruence. This proves the first part. The other part is completely symmetric.

$$\begin{tikzcd}
\forall E1 \arrow[r, dash, dotted, "\approx"] \arrow[rr, bend left, "\approx(goal)"] & \exists E_2 \arrow[r, dash, dotted, "\approx"] & \exists E_3 \\
\forall E \arrow[r, dash, dotted, "\approx^c"] \arrow[u, "\forall u"] & E' \arrow[r, dash, dotted, "\approx^c"] \arrow[u, Rightarrow, "u"] & E'' \arrow[u, Rightarrow, "u"]
\end{tikzcd}$$

□

Then we have proved the congruence (substitutivity) of observational congruence under various CCS process operators:

Proposition 3.7.2. *(The substitutivity of Observational Congruence)*

1. *Observation congruence is substitutive under the prefix operator:*

   `OBS_CONGR_SUBST_PREFIX:`
   $\vdash E \approx^c E' \Longrightarrow \forall u.\ u\texttt{..}E \approx^c u\texttt{..}E'$

2. *Observation congruence is substitutive under binary summation:*

   `OBS_CONGR_PRESD_BY_SUM`
   $\vdash E_1 \approx^c E_1' \land E_2 \approx^c E_2' \Longrightarrow E_1 \texttt{+} E_2 \approx^c E_1' \texttt{+} E_2'$

3. *Observation congruence is preserved by parallel composition:*

   `OBS_CONGR_PRESD_BY_PAR:`
   $\vdash E_1 \approx^c E_1' \land E_2 \approx^c E_2' \Longrightarrow E_1 \parallel E_2 \approx^c E_1' \parallel E_2'$

4. *Observation congruence is substitutive under the restriction operator:*

   `OBS_CONGR_SUBST_RESTR:`
   $\vdash E \approx^c E' \Longrightarrow \forall L.\ \nu\ L\ E \approx^c \nu\ L\ E'$

5. *Observation congruence is substitutive under the relabeling operator:*

   `OBS_CONGR_SUBST_RELAB:`
   $\vdash E \approx^c E' \Longrightarrow \forall rf.\ \texttt{relab}\ E\ rf \approx^c \texttt{relab}\ E'\ rf$

Finally, like the case for weak equivalence, we can easily prove the relationship between strong equivalence and observation congruence:

Theorem 3.7.4. *(Strong equivalence implies observation congruence)*

$\vdash E \sim E' \Longrightarrow E \approx^c E'$ `[STRONG_IMP_OBS_CONGR]`

With this result, all algebraic laws for observation congruence can be derived from the corresponding algebraic laws of strong equivalence. Here we omit these theorems, except for the following four τ-laws:

Theorem 3.7.5. *(The τ-laws for observational congruence)*

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

3.8 Relationship between Weak Equivalence and Observational Congruence

The relationship between weak equivalence and observation congruence was an interesting research topic, and there're many deep lemmas related. In this project, we have proved two such deep lemmas. The first one is the following Deng Lemma (for weak bisimularity[10]):

Theorem 3.8.1. *(Deng lemma for weak bisimilarity) If* $p \approx q$*, then one of the following three cases holds:*

1. $\exists p'$ *such that* $p \ -\tau\!\rightarrow p'$ *and* $p' \approx q$*, or*
2. $\exists q'$ *such that* $q \ -\tau\!\rightarrow q'$ *and* $p \approx q'$*, or*
3. $p \approx^c q$*.*

Formally:

$$\begin{array}{l} \vdash p \approx q \implies \\ \quad (\exists\, p'.\ p \ -\tau\!\rightarrow\ p' \land p' \approx q) \lor (\exists\, q'.\ q \ -\tau\!\rightarrow\ q' \land p \approx q') \lor \\ \quad p \approx^c q \end{array} \qquad \texttt{[DENG\_LEMMA]}$$

Proof. Actually there's no need to consider thee difference cases. Using the logical tautology $(\neg P \land \neg Q \implies R) \implies P \lor Q \lor R$, the theorem can be reduced to the following goal:

> Prove $p \approx^c q$, with the following three assumptions:
>
> 1. $p \approx q$
> 2. $\neg\exists\, p'.\quad p \ -\tau\!\rightarrow p' \land p' \approx q$
> 3. $\neg\exists\, q'.\quad q \ -\tau\!\rightarrow q' \land p \approx q'$

Now we check the definition of observation congruence: for any transition from p, say $p \ -u\!\rightarrow E_1$, consider the cases when $u = \tau$ and $u \neq \tau$:

1. If $u = \tau$, then by $p \approx q$ and the definition of weak equivalence, there exists E_2 such that $q \overset{\epsilon}{\Rightarrow} E_2$ and $E_1 \approx E_2$. But by assumption we know $q \neq E2$, thus $q \overset{\epsilon}{\Rightarrow} E_2$ contains at least one τ-transition, thus is actually $q \ =\!\tau\!\Rightarrow E_2$, which is required by the definition of observation congruence for $p \approx q$.

$$\begin{tikzcd} p \arrow[r, dotted, no head, "\approx^c"] \arrow[d, "\tau"] & q \arrow[d, Rightarrow, "\epsilon(\tau)"] \arrow[dl, dotted, no head, "\not\approx"'] \\ \forall E_1 \arrow[r, dotted, no head, "\approx"] & \exists E_2 \end{tikzcd}$$

[10] The original Deng lemma is for another kind of equivalence relation called *rooted branching bisimularity*, which is not touched in this project.

2. If $u = L$, then the requirement of observation congruence is directly satisfied.

The other direction is completely symmetric. □

Now we prove the famous Hennessy Lemma:

Theorem 3.8.2. *(Hennessy Lemma) For any processes p and q, $p \approx q$ if and only if ($p \approx^c q$ or $p \approx^c \tau..q$ or $\tau..p \approx^c q$):*

$\vdash p \approx q \iff p \approx^c q \lor p \approx^c \tau..q \lor \tau..p \approx^c q$ [HENNESSY_LEMMA]

Proof. The "if" part (from right to left) can be easily derived by applying:

- OBS_CONGR_IMP_WEAK_EQUIV,
- TAU_WEAK,
- WEAK_EQUIV_SYM, and
- WEAK_EQUIV_TRANS

We'll focus on the hard "only if" part (from left to right). The proof represent here is slightly simplier than the one in [4], but the idea is the same. The proof is based on creative case analysis:

1. If there exists an E such that $p \ -\tau\rightarrow E \land E \approx q$, we can prove $p \approx^c \tau..q$ by expanding $p \approx q$ by WEAK_PROPERTY_STAR. The other needed theorems are the definition of weak transition, EPS_REFL, SOS rule PREFIX and TRANS_PREFIX, TAU_PREFIX_WEAK_TRANS and TRANS_IMP_WEAK_TRANS.

2. If there's no E such that $p \ -\tau\rightarrow E \land E \approx q$, we can further check if there exist an E such that $q \ -\tau\rightarrow E \land p \approx E$, and in this case we can prove $\tau..p \approx^c q$ in the same way as the above case.

3. Otherwise we got exactly the same condition as in Deng Lemma (after the initial goal reduced in the previous proof), and in this case we can directly prove that $p \approx^c q$.

□

This formal proof has basically shown that, for most informal proofs in Concurrency Theory which doesn't depend on external mathematics theories, the author has got the ability to express it in HOL theorem prover.

3.9 The theory of (pre)congruence

One of the highlight of this project is the formal proofs for various versions of the "coarsest congruence contained in weak equivalence",

Proposition 3.9.1. *(Coarsest congruence contained in $\approx$) For any processes p and q, we have $p \approx^c q$ if and only if $\forall r.\ p + r \approx q + r$.*

At first glance, the name of above theorem doesn't make much sense. To see the nature of above theorem more clearly, here we represent a rather complete theory about the congruence of CCS. It's based on contents from [21].

To formalize the concept of congruence, we need to define "semantic context" first. There're multiple solutions, here we have chosen a simple solution based on λ-calculus:

Definition 3.9.1. (Semantic context of CCS) The semantic context (or one-hole context) of CCS is a function $C[\cdot]$ of type "$(\alpha,\ \beta)$ context" recursively defined by following rules:

$$\begin{array}{l}
\texttt{CONTEXT}\ (\lambda t.\ t) \\
\texttt{CONTEXT}\ (\lambda t.\ p) \\
\texttt{CONTEXT}\ e \implies \texttt{CONTEXT}\ (\lambda t.\ a..e\ t) \\
\texttt{CONTEXT}\ e_1 \land \texttt{CONTEXT}\ e_2 \implies \texttt{CONTEXT}\ (\lambda t.\ e_1\ t + e_2\ t) \\
\texttt{CONTEXT}\ e_1 \land \texttt{CONTEXT}\ e_2 \implies \texttt{CONTEXT}\ (\lambda t.\ e_1\ t \parallel e_2\ t) \\
\texttt{CONTEXT}\ e \implies \texttt{CONTEXT}\ (\lambda t.\ \nu\ L\ (e\ t)) \\
\texttt{CONTEXT}\ e \implies \texttt{CONTEXT}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf) \qquad \texttt{[CONTEXT\_rules]}
\end{array}$$

By repeatedly applying above rules, one can imagine that, any CCS term with zero or more "holes" at any depth, becomes a λ-function, and by calling the function with another CCS term, the holes were filled by that term.

The notable property of semantic context is that, the functional combination of two contexts is still a context:

Proposition 3.9.2. *(The combination of one-hole contexts) If both c_1 and c_2 are two semantic contexts, then $c_1 \circ c_2$[11] is still a one-hole context:*

CONTEXT_combin:
$\vdash$ CONTEXT c_1 $\land$ CONTEXT c_2 $\implies$ CONTEXT $(c_1 \circ c_2)$

Proof. By induction on the first context c_1. □

Now we're ready to define the concept of congruence (for CCS):

Definition 3.9.2. (Congruence of CCS) An equivalence relation $\approx$[12] on a specific space of CCS processes is a *congruence* iff for every n-ary operator f, one has $g_1 \approx h_1 \land \cdots g_n \approx h_n \Rightarrow f(g_1, \ldots, g_n) \approx f(h_1, \ldots, h_n$. This is the case iff for every semantic context $C[\cdot]$ on has $g \approx h \Rightarrow C[g] \approx C[h]$:

[11] $(c_1 \circ c_2)t := c_1(c_2 t)$.

[12] The symbol $\approx$ here shouldn't be understood as weak equivalence.

```
congruence_def:
congruence R ⟺ equivalence R ∧ precongruence R
```

If we remove the requirement that the relation must be equivalence, we got a similar definition of pre-congruence:

```
precongruence_def:
precongruence R ⟺
∀x y ctx. CONTEXT ctx ⟹ R x y ⟹ R (ctx x) (ctx y)
```

We can easily prove that, strong equivalence and observation congruence is indeed a congruence following above definition, using the substitutability and preserving properties of these relations:

Theorem 3.9.1. *Strong Equivalence ($\sim$) and Observation Congruence ($\approx^C$) are both congruence according to the above definition:*

```
STRONG_EQUIV_congruence:
 ⊢ congruence STRONG_EQUIV

OBS_CONGR_congruence:
 ⊢ congruence OBS_CONGR
```

For relations which is not congruence, it's possible to "convert" them into congruence:

Definition 3.9.3. (Constructing congruences from equivalence relation) Given an equivalence relation $\sim$[13], define $\sim^c$ by:

```
CC R = (λg h. ∀c. CONTEXT c ⟹ R (c g) (c h))                [CC_def]
```

This new operator on relations has the following three properties:

Proposition 3.9.3. *For all equivalence relation R, R^c is a congruence:*

```
 ⊢ equivalence R ⟹ congruence (CC R)                  [CC_congruence]
```

Proof. By construction, $\sim^c$ is a congruence. For if $g \sim^c h$ and $D[\cdot]$ is a semantic context, then for every semantic context $C[\cdot]$ also $C[D[\cdot]]$ is a semantic context, so $\forall C[\cdot].\ (C[D[g]] \sim C[D[h]])$ and hence $D[g] \sim^c D[h]$. □

Proposition 3.9.4. *For all R, R^c is finer than R:*

```
 ⊢ CC R ⊆ᵣ R                                              [CC_is_finer]
```

Proof. The trivial context guarantees that $g \sim^c h \Rightarrow g \sim h$, so $\sim^c$ is finer than $\sim$. □

[13] The Symbol $\sim$ here shouldn't be understood as strong equivalence.

Proposition 3.9.5. *For all R, R^c is the coarsest congruence finer than R, that is, for any other congruence finer than R, it's finer than R^c:*

$\vdash$ `congruence` $R' \wedge R' \subseteq_r R \Longrightarrow R' \subseteq_r$ `CC` R `[CC_is_coarsest]`

Proof. If $\approx$ is any congruence finer than $\sim$, then

$$g \approx h \Rightarrow \forall C[\cdot].\ (C[g] \approx C[h]) \Rightarrow \forall C[\cdot].\ (C[g] \sim C[h]) \Rightarrow g \sim^c h. \qquad (3.2)$$

Thus $\approx$ is finer than $\sim^c$. (i.e. $\sim^c$ is coarser than $\approx$, then the arbitrariness of $\approx$ implies that $\sim^c$ is coarsest.) □

As we know weak equivalence is not a congruence, and one way to "fix" it, is to use observation congruence which is based on weak equivalence but have special treatments on the first transitions. The other way is to build a congruence from existing weak equivalence relation, using above approach based on one-hole contexts. Such a congruence has a new name:

Definition 3.9.4. (Weak bisimulation congruence) The coarsest congruence that is finer than weak bisimulation equivalence is called *weak bisimulation congruence* (notation: $\sim^c_w$):

$\vdash$ `WEAK_CONGR = CC WEAK_EQUIV` `[WEAK_CONGR]`

or

`WEAK_CONGR_THM:`
$\vdash$ `WEAK_CONGR` `=` $(\lambda g\ h.\ \forall c.$ `CONTEXT` $c \Longrightarrow c\ g \approx c\ h)$

So far, the weak bisimulation congruence $\sim^c_w$ defined above is irrelevant with rooted weak bisimulation (a.k.a. observation congruence) $\approx^c$, which has the following standard definition also based on weak equivalence:

`OBS_CONGR:`
$\vdash E \approx^c E' \iff$
$\quad \forall u.$
$\quad\quad (\forall E_1.\ E\ {-u\rightarrow}\ E_1 \Longrightarrow \exists E_2.\ E'\ {=u\Rightarrow}\ E_2 \wedge E_1 \approx E_2) \wedge$
$\quad\quad \forall E_2.\ E'\ {-u\rightarrow}\ E_2 \Longrightarrow \exists E_1.\ E\ {=u\Rightarrow}\ E_1 \wedge E_1 \approx E_2$

But since observational congruence is congruence, it must be finer than weak bisimulation congruence:

Lemma 3.9.1. *Observational congruence implies weak bisimulation congruence:*

`OBS_CONGR_IMP_WEAK_CONGR:`
$\vdash p \approx^c q \Longrightarrow$ `WEAK_CONGR` $p\ q$

On the other side, by consider the trivial context and sum contexts in the definition of weak bisimulation congruence, we can easily prove the following result:

Lemma 3.9.2. *Weak bisimulation congruence implies sum equivalence:*

`WEAK_CONGR_IMP_SUM_EQUIV`:
$\vdash$ `WEAK_CONGR` $p\ q \implies$ `SUM_EQUIV` $p\ q$

Noticed that, in above theorem, the sum operator can be replaced by any other operator in CCS, but we know sum is special because it's the only operator in which the weak equivalence is not preserved after substitutions.

From above two lemmas, we can easily see that, weak equivalence is between the observation congruence and an unnamed relation $\{(p, q) \colon \forall\, r. p + r \approx q + r\}$ (we can temporarily call it "sum equivalence", because we don't if it's a congruence, or even if it's contained in weak equivalence). If we could further prove that "sum equivalence" is finer than observation congruence, then all three congruences (observation congruence, weak equivalence and the "sum equivalence" must all coincide, as illustrated in the following figure:

$$\begin{tikzcd}
\text{Weak equivalence}(\approx) & & \text{Sum equivalence} \arrow[ddl, bend left, "\subseteq?"] \\
& \text{Weak bisim. congruence}(\sim^c_w) \arrow[ul, "\subseteq"] \arrow[ur, "\subseteq"'] & \\
& \text{Observational congruence}(\approx^c) \arrow[u, "\subseteq"] &
\end{tikzcd}$$

This is why the proposition at the beginning of this section is called "coarsest congruence contained in weak equivalence", it's actually trying to prove that the "sum equivalence" is finer than "observation congruence" therefore makes "weak bisimulation congruence" ($\sim^c_w$) coincide with "observation congruence" ($\approx^c$).

3.10 Coarsest congruence contained in $\approx$

The easy part (left $\implies$ right) is already proven in previous section by combining

- `OBS_CONGR_IMP_WEAK_CONGR`, and
- `WEAK_CONGR_IMP_SUM_EQUIV`,

or it can be proved directly, using:

- `OBS_CONGR_IMP_WEAK_EQUIV`, and
- `OBS_CONGR_SUBST_SUM_R`

Theorem 3.10.1. *(The easy part "Coarsest congruence contained in $\approx$")*

$\vdash p \approx^c q \implies \forall\, r.\ p + r \approx q + r$ `[COARSEST_CONGR_LR]`

Thus we only focus on the hard part (right $\implies$ left) in the rest of this section.

3.10.1 With classical cardinality assumptions

A classic restriction is to assume cardinality limitations on the two processes, so that didn't use up all possible labels. Sometimes this assumption is automatically satisfied, for example: the CCS is finitrary and the set of all actions is infinite. But in our setting, the CCS datatype contains twi type variables, and if the set of all possible labels has only finite cardinalities, this assumtion may not be satisfied.

In [2] (Proposition 3 in Chapter 7, p. 153), Robin Milner simply calls this important theorem as "Proposition 3":

Proposition 3.10.1. *(Proposition 3 of observation congruence) Assume that $\mathcal{L}(P) \cup \mathcal{L}(Q) \neq \mathcal{L}$. Then $P \approx^c Q$ iff, for all R, $P + R \approx Q + R$.*

And in [4] (Theorem 4.5 in Chapter 4, p. 185), Prof. Roberto Gorrieri calls it "Coarsest congruence contained in $\approx$" (so did us in this paper):

Theorem 3.10.2. *(Coarsest congruence contained in $\approx$) Assume that $\mathrm{fn}(p) \cup \mathrm{fn}(q) \neq \mathscr{L}$. Then $p \approx^c q$ if and only if $p + r \approx q + r$ for all $r \in \mathscr{P}$.*

Both $\mathcal{L}(\cdot)$ and $\mathrm{fn}(\cdot)$ used in above theorems mean the set of "non-τ actions" (i.e. labels) used in a given process.

We analyzed the proof of above theorem and have found that, the assumption that the two processes didn't use up all available labels. Instead, it can be weakened to the following stronger version, which assumes the following properties instead:

Definition 3.10.1. (Processes having free actions) A CCS process is said to have *free actions* if there exists an non-τ action such that it doesn't appear in any transition or weak transition directly leading from the root of the process:

```
free_action_def:
⊢ free_action p ⟺ ∃ a. ∀ p′. ¬(p =label a⇒ p′)
```

Theorem 3.10.3. *(Stronger version of "Coarsest congruence contained in $\approx$", only the hard part) Assuming for two processes p and q have free actions, then $p \approx^c q$ if $p + r \approx q + r$ for all $r \in \mathscr{P}$:*

```
COARSEST_CONGR_RL:
⊢ free_action p ∧ free_action q ⟹ (∀ r. p + r ≈ q + r) ⟹ p ≈ᶜ q
```

This new assumption is weaker because, even p and q may have used all possible actions in their transition graphs, as long as there's one such free action for their first-step weak transitions, therefore the theorem still holds. Also noticed that, the two processes do not have to share the same free actions, this property focuses on single process.

Proof. (Proof of the stronger version of "Coarsest congruence contained in $\approx$") The kernel idea in this proof is to use that free action, say a, and have $p + a.0 \approx q + a.0$ as the working basis. Then for any transition from $p + a.0$, say $p + a.0 \overset{u}{\Longrightarrow} E_1$, there must be a weak transition of the same action u (or EPS when $u = \tau$) coming from

$q + a.0$ as the response. We're going to use the free-action assumptions to conclude that, when $u = \tau$, that EPS must contain at least one τ (thus satisfied the definition of observation congruence):

$$\begin{tikzcd}
p + a.0 \arrow[r, dotted, no head, "\approx"] \arrow[d, "u=\tau"] & q + a.0 \arrow[d, Rightarrow, "\epsilon"] \\
E_1 \arrow[r, dotted, no head, "\approx"] & E_2
\end{tikzcd}$$

Indeed, if the EPS leading from $q + a.0$ actually contains no τ-transition, that is, $q + a.0 = E_2$, then E_1 and E_2 cannot be weak equivalence: for any a-transition from $q + a.0$, $E1$ must response with a weak a-transition as $E_1 \overset{a}{\Longrightarrow} E_1'$, but this means $p \overset{a}{\Longrightarrow} E_1'$, which is impossible by free-action assumption on p:

$$\begin{tikzcd}
p \arrow[dr, "\tau"] \arrow[ddr, "a"'] & p + a.0 \arrow[r, dotted, no head, "\approx"] \arrow[d, "\tau"] & q + a.0 = E_2 \arrow[d, "a"] \\
& E_1 \arrow[ur, dotted, no head, "\approx"] \arrow[d, Rightarrow, "a"] & 0 \\
& E_1' \arrow[ur, dotted, no head, "\approx"] &
\end{tikzcd}$$

Once we have $q + a.0 \overset{\tau}{\Longrightarrow} E2$, the first τ-transition must comes from q, then it's obvious to see that E_2 is a valid response required by observation congruence of p and q in this case.

When $p \xrightarrow{L} E_1$, we have $p + a.0 \xrightarrow{L} E_1$, then there's an E_2 such that $q + a.0 \overset{L}{\Longrightarrow} E_2$. We can further conclude that $q \overset{L}{\Longrightarrow} E_2$ because by free-action assumption $L \neq a$. This finishes the first half of the proof, the second half (for all transition coming from q) is completely symmetric. □

Combining the easy and hard parts, the following theorem is proved:

Theorem 3.10.4. *(Coarsest congruence contained in ≈)*

COARSEST_CONGR_THM:
$\vdash$ free_action $p \land$ free_action $q \Longrightarrow (p \approx^c q \iff \forall r.\ p + r \approx q + r)$

3.10.2 Without cardinality assumptions

In 2005, Rob J. van Glabbeek published a paper [21] showing that "the weak bisimulation congruence can be characterized as rooted weak bisimulation equivalence, even without making assumptions on the cardinality of the sets of states or actions of the process under consideration". That is to say, above "Coarsest congruence contained in ≈" theorem holds even for two arbitrary processes! The idea is actually from Jan Willem Klop back to the 80s, but it's not published until that 2005 paper. This proof is not known to Robin Milner in [2]. The author carefully investigated this paper and formalized the proof in it.

The main result is the following version of the hard part of "Coarsest congruence contained in $\approx$" theorem under new assumptions:

Theorem 3.10.5. *(Coarsest congruence contained in $\approx$, new assumptions) For any two CCS processes p and q, if there exists another stable (i.e. first-step transitions are never τ-transition) process k which is not weak bisimlar with any sub-process follows from p and q by* one-step *weak transitions, then $p \approx^c q$ if $p + r \approx q + r$ for all $r \in \mathscr{P}$.*

```
⊢ (∃k.
      STABLE k ∧ (∀p′ u. p =u⇒ p′ ⟹ ¬(p′ ≈ k)) ∧
      ∀q′ u. q =u⇒ q′ ⟹ ¬(q′ ≈ k)) ⟹
   (∀r. p + r ≈ q + r) ⟹
   p ≈ᶜ q
```

Proof. Assuming the existence of that special process k, and take an arbitrary non-τ action, say a (this is always possible in our setting, because in higher order logic any valid type must contain at least one value), we'll use the fact that $p+a.k \approx q+a.k$ as our working basis. For all transitions from p, say $p \xrightarrow{u} E_1$, we're going to prove that, there must be a corresponding weak transition such that $q \overset{u}{\Longrightarrow} E_2$, and $E_1 \approx E_2$ (thus $p \approx^c q$. There're three cases to consider:

1. τ-transitions: $p \xrightarrow{\tau} E_1$. By SOS rule (Sum$_1$), we have $p + a.k \xrightarrow{\tau} E_1$, now by $p + a.k \approx q + a.k$ and the property (*) of weak equivalence, there exists an E_2 such that $q + a.k \overset{\epsilon}{\Longrightarrow} E_2$. We can use the property of k to assert that, such an EPS transition must contains at least one τ-transition. Because if it's not, then $q + a.k = E_2$, and since $E_1 \approx E_2$, for transition $q + a.k \xrightarrow{a} k$, E_1 must make a response by $E_1 \overset{a}{\Longrightarrow} E_1'$, and as the result we have $p \overset{a}{\Longrightarrow} E_1'$ and $E_1' \approx k$, which is impossible by the special choice of k:

$$\begin{tikzcd}
p \arrow[dr, "\tau"] \arrow[ddr, Rightarrow, "a"'] & p + a.k \arrow[d, "\tau"] \arrow[r, dotted, dash, "\approx"] & q + a.k = E_2 \arrow[d, "a"] \\
& E_1 \arrow[ur, dotted, dash, "\approx"] \arrow[d, Rightarrow, "a"] & k \\
& E_1' \arrow[ur, dotted, dash, "\not\approx"] &
\end{tikzcd}$$

2. If there's a a-transition coming from p (means that the arbitrary chosen action a is normally used by processes p and q), that is, $p \xrightarrow{a} E_1$, also $p+a.k \xrightarrow{a} E_1$, by property (*) of weak equivalence, there exists E_2 such that $q + a.k \overset{a}{\Longrightarrow} E_2$:

$$\begin{tikzcd}
p \arrow[dr, "a"] & p + a.k \arrow[d, "a"] \arrow[r, dotted, dash, "\approx"] & q + a.k \arrow[d, Rightarrow, "a"] \\
& \forall E_1 \arrow[r, dotted, dash, "\approx"] & \exists E_2
\end{tikzcd}$$

   We must further divide this weak transition into two cases based on its first step:

(a) If the first step is a τ-transition, then for sure this entire weak transition must come from q (otherwise the first step would be an a-transition from $a.k$). And in this case we can easily conclude $q \overset{a}{\Longrightarrow} E_2$ without using the property of k:

$$\begin{tikzcd}
p \arrow[rrr, bend left, "\approx^c"] \arrow[dr, "a"'] & p + a.k \arrow[d, "a"] \arrow[r, dotted, no head, "\approx"] & q + a.k \arrow[d, "\tau"] & q \arrow[dl, "\tau"'] \arrow[ddl, Rightarrow, "a"] \\
 & \forall E_1 \arrow[dr, dotted, no head, "\approx"] & \exists E' \arrow[d, Rightarrow, "a"] & \\
 & & \exists E_2 &
\end{tikzcd}$$

(b) If the first step is an a-transition, we can prove that, this a-transition must come from h (then the proof finishes for the entire a-transition case). Because if it's from the $a.k$, since k is stable, then there's no other coice but $E_2 = k$ and $E_1 \approx E_2$. This is again impossible for the special choice of k:

$$\begin{tikzcd}
p \arrow[dr, "a"'] & p + a.k \arrow[d, "a"] \arrow[r, dotted, no head, "\approx"] & q + a.k \arrow[d, "a"] \\
 & \forall E_1 \arrow[r, dotted, no head, "\not\approx"] & E2 = k
\end{tikzcd}$$

3. For other L-transitions coming from p, where $L \neq a$ and $L \neq \tau$. As a response to $p + a.k \xrightarrow{L} E_1$, we have $q + a.k \overset{L}{\Longrightarrow} E_2$ and $E_1 \approx E_2$. It's obvious that $q \overset{L}{\Longrightarrow} E_2$ in this case, no matter what the first step is (it can only be τ and L) and this satisfies the requirement of observation congruence natually:

$$\begin{tikzcd}
p \arrow[rrr, bend left, "\approx^c"] \arrow[dr, "\forall L"'] & p + a.k \arrow[d, "L"] \arrow[r, dotted, no head, "\approx"] & q + a.k \arrow[d, Rightarrow, "L"] & q \arrow[dl, Rightarrow, "L"] \\
 & \forall E_1 \arrow[r, dotted, no head, "\approx"] & \exists E2 &
\end{tikzcd}$$

The other direction (for all transitions coming from q) is completely symmetric. Combining all the cases, we have $p \approx^c q$. □

Now it remains to prove the existence of the special process mentioned in the assumption of above theorem.

3.10.3 Arbitrary many non-bisimilar processes

Strong equivalence, weak equivalence, observation congruence, they're all equivalence relations on CCS process space. General speaking, each equivalence relation must have *partitioned* all processes into several disjoint equivalence classes: processes in the same equivalence class are equivalent, and processes in different equivalence class are not equivalent.

The assumption in previous Theorem 3.10.5 requires the existence of a special CCS process, which is not weak equivalence to any sub-process leading from the two root processes by weak transitions. On worst cases, there may be infinite such sub-processes[14] Thus there's no essential differences to consider all states in the process group instead.

Then it's natural to ask if there are infinite equivalence classes of CCS processes. If so, then it should be possible to choose one which is not equivalent with all the (finite) states in the graphs of the two given processes. It turns out that, after Jan Willem Klop, it's possible to construct such processes, in which each of them forms a new equivalence class, we call them "Klop processes" in this paper:

Definition 3.10.2. (Klop processes) For each ordinal λ, and an arbitrary chosen non-τ action a, define a CCS process k_λ as follows:

1. $k_0 = 0$,
2. $k_{\lambda+1} = k_\lambda + a.k_\lambda$ and
3. 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.

Unfortunately, it's impossible to express infinite sums in our CCS datatype settings[15] without introducing new axioms. Therefore we have followed a two-step approach in this project: first we consider only the finite-state CCS (no need for axioms), then we turn to the general case.

3.10.4 Finite-state CCS

If both processes p and q are finite-state CCS processes, that is, the number of reachable states from p and q are both finite. And in this case, the following limited version of Klop processes can be defined as a recursive function (on natural numbers) in HOL4:

Definition 3.10.3. (Klop processes as recursive function on natural numbers)

```
KLOP a 0 = nil
KLOP a (SUC n) = KLOP a n + label a..KLOP a n                [KLOP_def]
```

By induction on the definition of Klop processes and SOS inference rules (Sum_1) and (Sum_2), we can easily prove the following properties of Klop functions:

[14] Even the CCS is finite branching, that's because after a weak transition, the end process may have an infinite τ-chain, and with each τ-transition added into the weak transition, the new end process is still a valid weak transition, thus lead to infinite number of weak transitions.

[15] And such infinite sums seems to go beyond the ability of the HOL's Datatype package

Proposition 3.10.2. *(Properties of Klop functions and processes)*

1. *All Klop processes are stable:*

   $\vdash$ `STABLE (KLOP` a n`)` [KLOP_PROP0]

2. *All transitions of a Klop process must lead to another smaller Klop process, and any smaller Klop process must be a possible transition of a larger Klop process:*

   [KLOP_PROP1]
   $\vdash$ `KLOP` a n `−label` a`→` E $\iff$ $\exists m.\ m < n \land (E$ `=` `KLOP` a $m)$

3. *The weak transition version of above property:*

   [KLOP_PROP1']
   $\vdash$ `KLOP` a n `=label` a`⇒` E $\iff$ $\exists m.\ m < n \land (E$ `=` `KLOP` a $m)$

4. *All Klop processes are distinct according to strong equivalence:*

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

5. *All Klop processes are distinct according to weak equivalence:*

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

6. *Klop functions are one-one:*

   $\vdash$ `ONE_ONE (KLOP` a`)` KLOP_ONE_ONE

Once we have a recursive function defined on all natural numbers $0, 1, \ldots$, we can map them into a set containing all these Klop processes, and the set is countable infinite. On the other side, the number of all states coming from two finite-state CCS processes p and q is finite. Choosing from an infinite set for an element distinct with any subprocess leading from p and q, is always possible. This result is purely mathematical, completely falling into basic set theory:

Lemma 3.10.1. *Given an equivalence relation R defined on a type, and two sets A, B of elements in this type, A is finite, B is infinite, and all elements in B are not equivalent, then there exists an element k in B which is not equivalent with any element in A:*

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

Proof. We built an explicit mapping f from A to B[16], for all $x \in A$, $y = f(x)$ if $y \in B$ and y is equivalent with x. But it's possible that no element in B is equivalent with x, and in this case we just choose an arbitrary element as $f(x)$. Such a mapping is to make sure the range of f always fall into B.

Now we can map A to a subset of B, say B_0, and the cardinality of B_0 must be equal or smaller than the cardinality of A, thus finite. Now we choose an element k from the rest part of B, this element is the desire one, because for any element $x \in A$, if it's equivalent with k, consider two cases for $y = f(x) \in B_0$:

1. y is equivalent with x. In this case by transitivity of R, we have two distinct elements y and k, one in B_0, the other in $B \setminus B_0$, they're equivalent. This violates the assumption that all elements in B are distinct.

2. y is arbitrary chosen because there's no equivalent element for x in B. But we already know one: k.

Thus there's no element x (in A) which is equivalent with k. □

To reason about finite-state CCS, we also need to define the concept of "finite-state":

Definition 3.10.4. (Definitions related to finite-state CCS)

1. Define *reachable* as the RTC of a relation, which indicates the existence of a transition between two processes:

   $\texttt{Reach} = (\lambda E\ E'.\ \exists u.\ E\ {-u\rightarrow}\ E')^*$ [Reach_def]

2. The "nodes" of a process is the set of all processes reachable from it:

   $\vdash \texttt{NODES}\ p = \{q \mid \texttt{Reach}\ p\ q\}$ [NODES_def]

3. A process is finite-state if the set of nodes is finite:

   $\vdash \texttt{finite\_state}\ p \iff \texttt{FINITE}\ (\texttt{NODES}\ p)$ [finite_state_def]

Among many properties of above definitions, we mainly rely on the following "obvious" property on weak transitions:

Proposition 3.10.3. *If p weakly transit to q, then q must be in the node set of p:*

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

[16] There're multiple ways to prove this lemma, a simpler proof is to make a reverse mapping from B to the power set of A (or further use the Axiom of Choice (AC) to make a mapping from B to A), then the non-injectivity of this mapping will contradict the fact that all elements in the infinite set are distinct. Our proof doesn't need AC, and it relies on very simple truths about sets.

Using all above results, now we can easily prove the following finite version of “Klop lemma”:

Lemma 3.10.2. *(Klop lemma, the finite version) For any two finite-state CCS p and q, there exists another process k, which is not weak equivalent with any subprocess weakly transited from p and q:*

```
KLOP_LEMMA_FINITE:
⊢ ∀p q.
    finite_state p ∧ finite_state q ⟹
    ∃k.
      STABLE k ∧ (∀p′ u. p =u⇒ p′ ⟹ ¬(p′ ≈ k)) ∧
      ∀q′ u. q =u⇒ q′ ⟹ ¬(q′ ≈ k)
```

Combining above lemma with Theorem 3.10.5 and Theorem 3.10.1, we can easily prove the following theorem for finite-state CCS:

Theorem 3.10.6. *(Coarsest congruence contained in ≈ for finite-state CCS)*

```
⊢ finite_state p ∧ finite_state q ⟹
  (p ≈ᶜ q ⟺ ∀r. p + r ≈ q + r)            [COARSEST_CONGR_FINITE]
```

3.10.5 Finitary CCS and general cases

For Finitrary CCS with potential infinite states, the proof of “Coarsest congruence contained in ≈” is currently not known. Currently we tend to believe the proof doesn’t exist, i.e. Observational Congruence may not be the coarsest congruence contained in weak equivalence if Finitary CCS is considered.

For more general cases in which CCS’s sum operator is allowed to take infinite (not only countable but also arbitrary largee) number of processes. The proof of “Coarsest congruence contained in ≈”, according to Rob J. van Glabbeek’s paper [21], indeed exists. However, this proof involves arbitrary large ordinals which is not supported in Higher Order Logic (not the software, but the logic itself).

The limitation also happens in HOL’s datatype package: infinite sums are not directly supported. However, if we’re allowed to add one axiom to enable infinite sums of CCS processes without touching the existing CCS datatype definition, we can actually precisely formalize the proof with the same steps as in [21]. For such a work, please refer to the author’s “internship” project report [22] with proof scripts available at [17]. This work is not included into this thesis simply because an axiom is introduced, which may potentially break the consistency of HOL.

[17] https://github.com/binghe/informatica-public/tree/master/CCS2

Chapter 4

A Formalization of “bisimulation up to”

“Bisimulation up to” is a powerful proof technique for proving many difficult results in process algebra. Generally speaking, it’s a technique for reducing the size of the relation needed to define a bisimulation. By definition, two processes are bisimilar if there exists a bisimulation relation containing them as a pair. However, in practice this definition is hardly ever followed plainly; instead, to reduce the size of the relations exhibited one prefers to define relations which are bisimulations only when closed up under some specific and priviledged relation, so to relieve the proof work needed. We call this an *“up-to” technique.* It is a pretty general device which allows a great variety of prssibilities.

According to [23], the variety of the up-to techniques is useful because it allows us each time to make the most convenient choice, depending upon the equilibrium we want between the size of the relation to exhibit and the fineness of the closure relation(s).

In this thesis project, we have basically followed the path of Robin Milner, by using “Bisimulaition up to strong equivalence” to prove the “Unique solutions of equations” theorem (for the strong equivalence case). But all the rest versions of “Unique solutions of equations” theorems didn’t use any “Bisimulation up to” techniques. This is mostly because the errors in the original version of Milner’s 1989 book [2], which has lead to the failures when applying “Weak bisimulation up to” techniques in the proof of unique solutions theorem. Instead, in next chapter we’ll present new proofs, which is slightly longer than the version in Milner’s book, but has no dependencies on any “Bisimulation up to” techniques, therefore the overall proof size is smaller. In this chapter, we have still formalized all versions of “Bisimulation up to” mentioned in Milner’s book, its errata, and [23], plus one more variants for observational congruence (but it’s too restrictive for proving the corresponding unique solutions theorem).

4.1 Bisimulation up to $\sim$

Following [2], the concept of "Bisimulation up to $\sim$" starts with a generalization of the notion of strong bisimulation, which is often more useful in applications. The following definition and proposition put the idea on a firm basis. Henceforward we shall oftern write $P\mathcal{R}Q$ to mean $(P, Q) \in \mathcal{R}$, for any binary relation $\mathcal{R}$. Note also that $\sim \mathcal{S} \sim$ is a composition of binary relations, so that $P \sim \mathcal{S} \sim Q$ means that for some P' and Q' we have $P \sim P'$, $P'\mathcal{S}Q'$ and $Q' \sim Q$.

Definition 4.1.1. (Bisimulation up to $\sim$) $\mathcal{S}$ is a "*bisimulation up to* $\sim$" if $P\mathcal{S}Q$ implies, for all $\alpha \in Act$,

1. Whenever $P \overset{\alpha}{\to} P'$ then, for some Q', $Q \overset{\alpha}{\to} Q'$ and $P' \sim \mathcal{S} \sim Q'$,
2. Whenever $Q \overset{\alpha}{\to} Q'$ then, for some P', $P \overset{\alpha}{\to} P'$ and $P' \sim \mathcal{S} \sim Q'$.

Or formally,

```
STRONG_BISIM_UPTO:
 ⊢ STRONG_BISIM_UPTO Bsm ⟺
   ∀E E′.
     Bsm E E′ ⟹
     ∀u.
       (∀E₁.
          E −u→ E₁ ⟹
          ∃E₂.
            E′ −u→ E₂ ∧
            (STRONG_EQUIV ∘ᵣ Bsm ∘ᵣ STRONG_EQUIV) E₁ E₂) ∧
       ∀E₂.
         E′ −u→ E₂ ⟹
         ∃E₁.
           E −u→ E₁ ∧ (STRONG_EQUIV ∘ᵣ Bsm ∘ᵣ STRONG_EQUIV) E₁ E₂
```

Pictorially, clause (1) says that if $P\mathcal{S}Q$ and $P\overset{\alpha\,\prime}{P}$ then we can fill in the following diagram:

$$\begin{tikzcd}
 & P \arrow[dl, "\alpha"] & \mathcal{S} & Q \arrow[dr, "\alpha"] & \\
P' \arrow[r, dotted, dash, "\sim"] & P'' & \mathcal{S} & Q'' \arrow[r, dotted, dash, "\sim"] & Q'
\end{tikzcd}$$

"Bisimulation up to $\sim$" has the following basic properties:

Proposition 4.1.1. *Properties of "strong bisimulation up to $\sim$"*

1. *Identity relation is a "strong bisimulation up to $\sim$":*

```
⊢ STRONG_BISIM_UPTO (=)              [IDENTITY_STRONG_BISIM_UPTO]
```

2. *The converse of a "strong bisimulation up to $\sim$" is still "strong bisimulation up to $\sim$":*

   CONVERSE_STRONG_BISIM_UPTO:
   $\vdash$ STRONG_BISIM_UPTO $Wbsm \implies$ STRONG_BISIM_UPTO $Wbsm^T$

And we have proved the following lemma, which establishes the relationship between "strong bisimulation up to $\sim$" and "strong bisimulation":

Lemma 4.1.1. *If $\mathcal{S}$ is a "bisimulation up to $\sim$", then $\sim \mathcal{S} \sim$ is a strong bisimulation:*

STRONG_BISIM_UPTO_LEMMA:
$\vdash$ STRONG_BISIM_UPTO $Bsm \implies$
STRONG_BISIM (STRONG_EQUIV $\circ_r$ Bsm $\circ_r$ STRONG_EQUIV)

Proof. The idea is to fix two process E and E', which satisfies $E \sim \circ Bsm \circ E'$, then check it for the definition of strong bisimulation: for all E_1 such that $E \ -u\rightarrow E_1$, there exists E_2'' such that $E' \ -u\rightarrow E_2''$ (the other side is totally symmetric), as shown in the following graph:

$$\begin{tikzcd}
E \arrow[rr, dash, "\sim"] \arrow[d, "\forall u"] & & \exists y' \arrow[r, dash, "Bsm"] \arrow[dl, "u"] & \exists y \arrow[rr, dash, "\sim"] \arrow[dr, "u"] & & E' \arrow[d, "u"] \\
\forall E_1 \arrow[r, dash, "\sim"] & \exists E_2 \arrow[r, dash, "\sim"] & \exists y''' \arrow[r, dash, "Bsm"] & \exists y'' \arrow[r, dash, "\sim"] & \exists E_2' \arrow[r, dash, "\sim"] & \exists E_2''
\end{tikzcd}$$

During the proof, needed lemmas are the definition of "bisimulation up to $\sim$" (for expanding "y' Bsm y" into "$E_2 \sim y'''$ Bsm $y'' \sim E_2'$"), plus the property (*) and transitivity of strong equivalence. □

Based on above lemma, we then easily proved the following proposition:

Theorem 4.1.1. *If $\mathcal{S}$ is a "bisimulation up to $\sim$", then $\mathcal{S} \subseteq \sim$:*

STRONG_BISIM_UPTO_THM:
$\vdash$ STRONG_BISIM_UPTO $Bsm \implies Bsm \subseteq_r$ STRONG_EQUIV

Hence, to prove $P \sim Q$, we only have to find a strong bisimulation up to $\sim$ which contains (P, Q).

4.2 Bisimulation up to $\approx$

The concept of bisimulation up to $\approx$ is a modified version of Milner's original definition presented in modern textbooks and originally in [23].

Definition 4.2.1. (Bisimulation up to $\approx$) $\mathcal{S}$ is a "*bisimulation up to* $\approx$" if $P\mathcal{S}Q$ implies, for all $\alpha \in Act$,

1. Whenever $P \overset{\alpha}{\rightarrow} P'$ then, for some Q', $Q \overset{\hat{\alpha}}{\Rightarrow} Q'$ and $P' \sim \mathcal{S} \approx Q'$,

2. Whenever $Q \overset{\alpha}{\rightarrow} Q'$ then, for some P', $P \overset{\hat{\alpha}}{\Rightarrow} P'$ and $P' \approx \mathcal{S} \sim Q'$.

Or formally,

```
WEAK_BISIM_UPTO:
⊢ WEAK_BISIM_UPTO Wbsm ⟺
   ∀E E'.
     Wbsm E E' ⟹
     (∀l.
        (∀E1.
           E −label l→ E1 ⟹
           ∃E2.
             E' =label l⇒ E2 ∧
             (WEAK_EQUIV ∘r Wbsm ∘r STRONG_EQUIV) E1 E2) ∧
        ∀E2.
          E' −label l→ E2 ⟹
          ∃E1.
            E =label l⇒ E1 ∧
            (STRONG_EQUIV ∘r Wbsm ∘r WEAK_EQUIV) E1 E2) ∧
     (∀E1.
        E −τ→ E1 ⟹
        ∃E2.
          E' =ε⇒ E2 ∧ (WEAK_EQUIV ∘r Wbsm ∘r STRONG_EQUIV) E1 E2) ∧
     ∀E2.
       E' −τ→ E2 ⟹
       ∃E1. E =ε⇒ E1 ∧ (STRONG_EQUIV ∘r Wbsm ∘r WEAK_EQUIV) E1 E2
```

A few things must be noticed:

1. In HOL4, the big "O" notion as relation composition has different orders with usual Math notion: *the right-most relation takes the input argument first*, which is actually the case for function composition: $(f \circ g)(x) = f(g(x))$. Thus in all HOL-generated terms like

   ```
   (WEAK_EQUIV ∘r Wbsm ∘r STRONG_EQUIV) E1 E2
   ```

   in this paper, it should be understood like "$E_1 \sim y$ Wbsm $y' \approx E_2$". (There was no such issues for the strong bisimulation cases, because we had $\sim$ on both side)

2. The original definition in Milner's book [2], in which he used $\approx \mathcal{S} \approx$ in all places in above definition, has been found (by his student, now Prof. Davide Sangiogi) as problematic. The reason has been explained in Gorrieri's book [4] (page 65), that the resulting relation may not be a subset of $\approx$! Thus we have used the definition from Gorrieri's book, with the definition in Sangiorgi's book [18] (page 115) doubly confirmed.

3. Some authors (e.g. Prof. Davide Sangiorgi) uses the notions like $P \overset{\hat{\mu}}{\Rightarrow} P'$ to represent special case that, $P \overset{\epsilon}{\Rightarrow} P'$ when $\mu = \tau$ (i.e. it's possible that $P = Q$). Such notions are concise, but inconvenient to use in formalization work, because the EPS transition, in many cases, has very different characteristics. Thus we have above long definition but easier to use when proving all needed results.

Two basic properties to help understanding "bisimulation up to $\approx$":

Lemma 4.2.1. *(Properties of bisimulation up to $\approx$)*

1. *The identity relation is "bisimulation up to $\approx$":*

   $\vdash$ `WEAK_BISIM_UPTO (=)` `[IDENTITY_WEAK_BISIM_UPTO]`

2. *The converse of a "bisimulation up to $\approx$" is still "bisimulation up to $\approx$":*

   `CONVERSE_WEAK_BISIM_UPTO:`
   $\vdash$ `WEAK_BISIM_UPTO` $\mathit{Wbsm} \implies$ `WEAK_BISIM_UPTO` Wbsm^T

Now we want to prove the following main lemma:

Lemma 4.2.2. *If $\mathcal{S}$ is a "bisimulation up to $\approx$", then $\approx \mathcal{S} \approx$ is a bisimulation.*

`WEAK_BISIM_UPTO_LEMMA:`
$\vdash$ `WEAK_BISIM_UPTO` $\mathit{Wbsm} \implies$
`WEAK_BISIM (WEAK_EQUIV` $\circ_r$ Wbsm $\circ_r$ `WEAK_EQUIV)`

Proof. Milner's books simply said that the proof is "analogous" to the same lemma for strong bisimulation. This is basically true, from left to right (for visible transitions):

$$\begin{tikzcd}
E \arrow[rr, dash, "\approx"] \arrow[d, "\forall l"'] & & \exists y' \arrow[r, dash, "Wbsm"] \arrow[dl, Rightarrow, "l"] & \exists y \arrow[rr, dash, "\approx"] \arrow[dr, Rightarrow, "l"] & & E' \arrow[d, Rightarrow, "l"] \\
\forall E_1 \arrow[r, dash, "\approx"] & \exists E_2 \arrow[r, dash, "\sim"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[r, dash, "\approx"] & \exists E_2' \arrow[r, dash, "\approx"] & \exists E_2''
\end{tikzcd}$$

There's a little difficulty, however. Given $y \approx E'$ and $y \overset{l}{\Rightarrow} E_2'$, the existence of E_2'' doesn't follow directly from the definition or property (*) of weak equivalence. Instead, we have to prove a lemma (to be presented below) to finish this last step.

More difficulties appear from right to left:

$$\begin{tikzcd}
E \arrow[rr, dash, "\approx"] \arrow[d, Rightarrow, "l"] & & \exists y' \arrow[r, dash, "Wbsm"] \arrow[dl, Rightarrow, "l"] & \exists y \arrow[rr, dash, "\approx"] \arrow[dr, Rightarrow, "l"] & & E' \arrow[d, "\forall l"] \\
\exists E_1'' \arrow[r, dash, "\approx"] & \exists E_1' \arrow[r, dash, "\approx"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[r, dash, "\sim"] & \exists E_1 \arrow[r, dash, "\approx"] & \forall E_2
\end{tikzcd}$$

The problem is, given y' Bsm y and $y \overset{l}{\Rightarrow} E_1$, the existence of E_1' doesn't follow directly from the definition of "bisimulation up to $\approx$", instead this result must be proved (to be presented below) and the proof is non-trivial.

The other two cases concerning τ-transitions:

$$\begin{tikzcd}
E \arrow[rr, dash, "\approx"] \arrow[d, "\tau"] & & \exists y' \arrow[rr, dash, "Wbsm"] \arrow[dl, Rightarrow, "\epsilon"] & & \exists y \arrow[rr, dash, "\approx"] \arrow[dr, Rightarrow, "\epsilon"] & & E' \arrow[d, Rightarrow, "\epsilon"] \\
\forall E_1 \arrow[r, dash, "\approx"] & \exists E_2 \arrow[r, dash, "\sim"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[rr, dash, "\approx"] & & \exists E_2' \arrow[r, dash, "\approx"] & \exists E_2''
\end{tikzcd}$$

in which the EPS transition bypass of weak equivalence (from E_2' to E_2'') must be proved, and

$$\begin{tikzcd}
E \arrow[rr, dash, "\approx"] \arrow[d, Rightarrow, "\epsilon"] & & \exists y' \arrow[rr, dash, "Wbsm"] \arrow[dl, Rightarrow, "\epsilon"] & & \exists y \arrow[rr, dash, "\approx"] \arrow[dr, Rightarrow, "\epsilon"] & & E' \arrow[d, "\tau"] \\
\exists E_1'' \arrow[r, dash, "\approx"] & \exists E_1' \arrow[r, dash, "\approx"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[rr, dash, "\sim"] & & \exists E_1 \arrow[r, dash, "\approx"] & \forall E_2
\end{tikzcd}$$

in which the EPS transition bypass for "bisimulation up to $\approx$" (from E_1 to E_1') must be proved as a lemma. □

As a summary of all "difficulties", here is a list of lemmas we have used to prove the previous lemma. Each lemma has also their "companion lemma" concerning the other directions (here we omit them):

Lemma 4.2.3. *(Useful lemmas concerning the first weak transitions from $\sim$, $\approx$ and "bisimulation up to $\approx$")*

1. `STRONG_EQUIV_EPS`:
   $\vdash E \sim E' \Longrightarrow \forall E_1.\ E \overset{\epsilon}{\Rightarrow} E_1 \Longrightarrow \exists E_2.\ E' \overset{\epsilon}{\Rightarrow} E_2 \wedge E_1 \sim E_2$

2. `WEAK_EQUIV_EPS`:
   $\vdash E \approx E' \Longrightarrow \forall E_1.\ E \overset{\epsilon}{\Rightarrow} E_1 \Longrightarrow \exists E_2.\ E' \overset{\epsilon}{\Rightarrow} E_2 \wedge E_1 \approx E_2$

3. `WEAK_EQUIV_WEAK_TRANS_label`:
   $\vdash E \approx E' \Longrightarrow$
   $\quad \forall l\ E_1.\ E\ \texttt{=label}\ l\!\Rightarrow E_1 \Longrightarrow \exists E_2.\ E'\ \texttt{=label}\ l\!\Rightarrow E_2 \wedge E_1 \approx E_2$

4. `WEAK_EQUIV_WEAK_TRANS_tau`:
   $\vdash E \approx E' \Longrightarrow \forall E_1.\ E =\!\tau\!\Rightarrow E_1 \Longrightarrow \exists E_2.\ E' \overset{\epsilon}{\Rightarrow} E_2 \wedge E_1 \approx E_2$

5. `WEAK_BISIM_UPTO_EPS`:
   $\vdash \texttt{WEAK\_BISIM\_UPTO}\ \mathit{Wbsm} \Longrightarrow$
   $\quad \forall E\ E'.$
   $\quad\quad \mathit{Wbsm}\ E\ E' \Longrightarrow$
   $\quad\quad \forall E_1.$
   $\quad\quad\quad E \overset{\epsilon}{\Rightarrow} E_1 \Longrightarrow$
   $\quad\quad\quad \exists E_2.\ E' \overset{\epsilon}{\Rightarrow} E_2 \wedge (\texttt{WEAK\_EQUIV} \circ_r \mathit{Wbsm} \circ_r \texttt{STRONG\_EQUIV})\ E_1\ E_2$

6. `WEAK_BISIM_UPTO_WEAK_TRANS_label`:

$$\begin{array}{l}
\vdash \texttt{WEAK\_BISIM\_UPTO}\ \mathit{Wbsm} \implies \\
\quad \forall E\ E'. \\
\quad\quad \mathit{Wbsm}\ E\ E' \implies \\
\quad\quad \forall l\ E_1. \\
\quad\quad\quad E\ \texttt{=label}\ l\!\Rightarrow\ E_1 \implies \\
\quad\quad\quad \exists E_2. \\
\quad\quad\quad\quad E'\ \texttt{=label}\ l\!\Rightarrow\ E_2\ \wedge \\
\quad\quad\quad\quad (\texttt{WEAK\_EQUIV}\ \circ_r\ \mathit{Wbsm}\ \circ_r\ \texttt{STRONG\_EQUIV})\ E_1\ E_2
\end{array}$$

Proof. (Proof sketch of above lemmas) The proof of `STORNG_EQUIV_EPS` and `WEAK_EQUIV_EPS` depends on the following "right-side induction" [1] of the EPS transition:

$$\begin{array}{ll}
\vdash (\forall x.\ P\ x\ x)\ \wedge\ (\forall x\ y\ z.\ P\ x\ y\ \wedge\ y\ {-\tau\!\rightarrow}\ z \implies P\ x\ z) \implies & \\
\quad \forall x\ y.\ x \overset{\epsilon}{\Rightarrow} y \implies P\ x\ y & [\texttt{EPS\_ind\_right}]
\end{array}$$

Basically, we need to prove that, if the lemma already holds for $n-1$ τ-transitions, it also holds for n τ-transitions.

The proof of `WEAK_BISIM_UPTO_EPS` is also based on above induction theorem. The induction case of this proof can be sketched using the following graph:

$$\begin{tikzcd}
 & E \arrow[rrr, dash, "Wbsm"] \arrow[d, dotted, dash] & & & E' \arrow[d, Rightarrow, "\epsilon"] & \\
 & E_1 \arrow[r, dash, "\sim"] \arrow[dl, "\tau"'] & \exists y' \arrow[r, dash, "Wbsm"] \arrow[dl, "\tau"'] & \exists y \arrow[r, dash, "\approx"] \arrow[dr, Rightarrow, "\epsilon"] & E_2 \arrow[dr, Rightarrow, "\epsilon"] & \\
\forall E_1' \arrow[r, dash, "\approx"] & \exists E_2' \arrow[r, dash, "\sim"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[r, dash, "\approx"] & \exists E_2'' \arrow[r, dash, "\approx"] & \exists E_2'''
\end{tikzcd}$$

The goal is to find E_2''' which satisfy $E' \overset{\epsilon}{\Rightarrow} E2'''$. As we can see from the graph, using the induction, now given y' Wbsm y and $y' \overset{\tau}{\rightarrow} E_2'$, we can easily crossover the "bisimulation up to $\approx$" and assert the existence of E_2'' to finish the proof.

The proof of `WEAK_BISIM_UPTO_WEAK_TRANS_label` is based on `WEAK_BISIM_UPTO_EPS`. It is much more difficult, because an even bigger graph must be step-by-

[1] Such induction theorems are part of HOL's theorem `relationTheory` for RTCs (reflexive transitive closure). The proof of transitivity of observation congruence also heavily depends on this induction theorem, but in the work of Monica Nesi where the EPS relation is manually defined inductively, such an induction theorem is not available (and it's not easy to prove it), as a result Monica Nesi couldn't finish the proof for transitivity of observation congruence, which is incredible hard to prove without proving lemmas like `WEAK_EQUIV_EPS` first.

step constructed:

$$\begin{tikzcd}
 & E \arrow[rrr, dash, "Wbsm"] \arrow[d, Rightarrow, "\epsilon"] & & & E' \arrow[d, Rightarrow, "\epsilon"] & \\
 & \exists E_1' \arrow[r, dash, "\sim"] \arrow[dl, "l"] & \exists y' \arrow[r, dash, "Wbsm"] \arrow[dl, "l"] & \exists y \arrow[r, dash, "\approx"] \arrow[dr, Rightarrow, "l"] & E_2' \arrow[dr, Rightarrow, "l"] & \\
\exists E_2 \arrow[r, dash, "\sim"] \arrow[dr, dotted, dash] & \exists E_2'' \arrow[rrr, dash, "\sim \text{Wbsm} \approx"] \arrow[dr, dotted, dash] & & & \exists E_2''' \arrow[r, dash, "\approx"] \arrow[dl, dotted, dash] & \exists E_2^{(4)} \arrow[dl, dotted, dash] \\
 & \exists E_2 \arrow[r, dash, "\sim"] \arrow[dl, Rightarrow, "\epsilon"] & \exists y''' \arrow[r, dash, "Wbsm"] \arrow[dl, Rightarrow, "\epsilon"] & \exists y'' \arrow[r, dash, "\approx"] \arrow[dr, Rightarrow, "\epsilon"] & \exists E_2^{(4)} \arrow[dr, Rightarrow, "\epsilon"] & \\
\forall E_1 \arrow[r, dash, "\sim"] & \exists E_2^{(5)} \arrow[rrr, dash, "\sim \text{Wbsm} \approx"] & & & \exists E_2^{(6)} \arrow[r, dash, "\approx"] & \exists E_2^{(7)}
\end{tikzcd}$$

That is, for all E_1 such that $E \overset{l}{\Rightarrow} E_1$ (which by definition of weak transitions exists E_1' and E_2 such that $E \overset{\epsilon}{\Rightarrow} E_1'$, $E_1' \overset{l}{\rightarrow} E_2$ and $E_2 \overset{\epsilon}{\Rightarrow} E_1$), we would like to finally find an $E_2^{(7)}$ such that $E' \overset{l}{\Rightarrow} E_2^{(7)}$. This process is long and painful, and we have to use `WEAK_BISIM_UPTO_EPS` twice. The formal proof tries to build above graph by asserting the existences of each process step-by-step, until it finally reached to $E_2^{(}7)$. This proof is so-far the largest formal proof (in single branch) that the author ever met, before closing it has 26 assumptions which represents above graph:

```
?E2. E' ==label l=>> E2 /\ (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E2
------------------------------------
  0.  WEAK_BISIM_UPTO Wbsm
  1.  Wbsm E E'
  2.  E ==label l=>> E1
  3.  EPS E E1'
  4.  E1' --label l-> E2
  5.  EPS E2 E1
  6.  EPS E' E2'
  7.  STRONG_EQUIV E1' y'
  8.  Wbsm y' y
  9.  WEAK_EQUIV y E2'
  10.  y' --label l-> E2''
  11.  STRONG_EQUIV E2 E2''
  12.  y ==label l=>> E2'''
  13.  (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E2'' E2'''
  14.  E2' ==label l=>> E2''''
  15.  WEAK_EQUIV E2''' E2''''
  16.  STRONG_EQUIV E2 y'''
  17.  Wbsm y''' y''
  18.  WEAK_EQUIV y'' E2''''
  19.  EPS y''' E2'''''
```

```
20.  STRONG_EQUIV E1 E2'''''
21.  EPS y'' E2''''''
22.  (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E2''''' E2''''''
23.  EPS E2'''' E2'''''''
24.  WEAK_EQUIV E2'''''' E2'''''''
25.  (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E2'''''''
```

□

All above lemmas concern the cases from left and right (for all P, exists Q such that ...) To prove the other side (for all Q there exist P such that ...), there's no need to go over the painful proving process again, instead we can easily derive the other side by using `CONVERSE_WEAK_BISIM_UPTO`. For example, once above lemma `WEAK_BISIM_UPTO_WEAK_TRANS_label` is proved, it's trivial to get the following companion lemma:

Lemma 4.2.4. *(The companion lemma of* `WEAK_BISIM_UPTO_WEAK_TRANS_label`*)*

```
WEAK_BISIM_UPTO_WEAK_TRANS_label':
 ⊢ WEAK_BISIM_UPTO Wbsm ⟹
   ∀ E E'.
     Wbsm E E' ⟹
     ∀ l E₂.
       E' =label l⇒ E₂ ⟹
       ∃ E₁.
         E =label l⇒ E₁ ∧
         (STRONG_EQUIV ∘ᵣ Wbsm ∘ᵣ WEAK_EQUIV) E₁ E₂
```

Finally, once the main lemma `WEAK_BISIM_UPTO_LEMMA`, the following final result can be easily proved, following the same idea in the proof of strong bisimulation cases:

Theorem 4.2.1. *If $\mathcal{S}$ is a bisimulation up to $\approx$, then $\mathcal{S} \subseteq\approx$: WEAK_BISIM_UPTO_THM:*

```
 ⊢ WEAK_BISIM_UPTO Wbsm ⟹ Wbsm ⊆ᵣ WEAK_EQUIV
```

As we have told at the beginning of this chapter, above result cannot be used for proving the "unique solution of equations" theorem for the weak equivalence (and observational congruence) cases. So above final theorem is not used anywhere in this thesis. It's simply an interesting result on its own.

4.3 Another version of "Bisimulation upto $\approx$"

Milner's 1989 book contains an error: the definition of "bisimulation upto $\approx$" cannot be used to prove the "unique solutions of equations" theorem for weak equivalence. This issue was originally found by Prof. Davide Sangiorgi when he was a student of Robin Milner. The solution was published as a new paper [24], co-authored by them. The solution, however, is not to fix the original definition but to invent an *alternative* version of "bisimulation up to ≈" which has not relationship with the old one. It's been found that, both versions were useful. In our project, this alternative version is also formalized, to support the proof of "unique solutions of equations" theorem for weak equivalence. Here is the definition:

Definition 4.3.1. (Another version of bisimulation up to ≈) $\mathcal{S}$ is a "*bisimulation up to* ≈" if $P\mathcal{S}Q$ implies, for all $\alpha \in Act$,

1. Whenever $P \overset{\alpha}{\Rightarrow} P'$ then, for some Q', $Q \overset{\hat{\alpha}}{\rightarrow} Q'$ and $P' \approx \mathcal{S} \approx Q'$,
2. Whenever $Q \overset{\alpha}{\Rightarrow} Q'$ then, for some P', $P \overset{\hat{\alpha}}{\rightarrow} P'$ and $P' \approx \mathcal{S} \approx Q'$.

Or formally,

```
WEAK_BISIM_UPTO_ALT:
 ⊢ WEAK_BISIM_UPTO_ALT $Wbsm$ $\iff$
    $\forall E\ E'$.
      $Wbsm\ E\ E'$ $\Longrightarrow$
      ($\forall l$.
         ($\forall E_1$.
            $E$ =label $l$⇒ $E_1$ $\Longrightarrow$
            $\exists E_2$.
              $E'$ =label $l$⇒ $E_2$ $\wedge$
              (WEAK_EQUIV $\circ_r$ $Wbsm$ $\circ_r$ WEAK_EQUIV) $E_1$ $E_2$) $\wedge$
         $\forall E_2$.
           $E'$ =label $l$⇒ $E_2$ $\Longrightarrow$
           $\exists E_1$.
             $E$ =label $l$⇒ $E_1$ $\wedge$
             (WEAK_EQUIV $\circ_r$ $Wbsm$ $\circ_r$ WEAK_EQUIV) $E_1$ $E_2$) $\wedge$
      ($\forall E_1$.
         $E$ =$\tau$⇒ $E_1$ $\Longrightarrow$
         $\exists E_2$.
           $E'$ $\overset{\epsilon}{\Rightarrow}$ $E_2$ $\wedge$ (WEAK_EQUIV $\circ_r$ $Wbsm$ $\circ_r$ WEAK_EQUIV) $E_1$ $E_2$) $\wedge$
      $\forall E_2$.
        $E'$ =$\tau$⇒ $E_2$ $\Longrightarrow$
        $\exists E_1$. $E$ $\overset{\epsilon}{\Rightarrow}$ $E_1$ $\wedge$ (WEAK_EQUIV $\circ_r$ $Wbsm$ $\circ_r$ WEAK_EQUIV) $E_1$ $E_2$
```

Noticed that, now there're two weak equivalences which surround the central relation. (So instead of calling it "alternative version", maybe "double-weak version" is better.)

It turns out that, the proof for properties of this alternative relation is slightly easier. We have proved exactly the same lemmas and theorems for it, of which the last two results are the following ones:

Lemma 4.3.1. *If $\mathcal{S}$ is a "bisimulation up to ≈" (alternative version), then ≈ $\mathcal{S}$ ≈ is a weak bisimulation:*

```
WEAK_BISIM_UPTO_ALT_LEMMA:
 ⊢ WEAK_BISIM_UPTO_ALT Wbsm ⟹
   WEAK_BISIM (WEAK_EQUIV ∘r Wbsm ∘r WEAK_EQUIV)
```

Theorem 4.3.1. *If $\mathcal{S}$ is a "bisimulation up to $\approx$" (alternative version), then $\mathcal{S} \subseteq \approx$:*

```
WEAK_BISIM_UPTO_ALT_THM:
 ⊢ WEAK_BISIM_UPTO_ALT Wbsm ⟹ Wbsm ⊆r WEAK_EQUIV
```

In next chapter, we'll use the theorem WEAK_BISIM_UPTO_ALT_THM to prove Milner's "unique solutions of equations" theorems for weak equivalence, however without it the proof can still be finished (by constructing a more complex bisimulation), just a little longer.

4.4 Observational bisimulation up to $\approx$

This is a new creation by the author during the early proof attempts for Milner's "unique solutions of equations" theorems for observational congruence. It turns out that, this new relation is too restrictive, thus finally it's not used. Here we briefly mention it as an independent (useless) findings. The definition is based on the original "bisimulation up to $\approx$" with EPS transitions replaced with normal weak τ transtions:

Definition 4.4.1. (Observational bisimulation up to $\approx$) $\mathcal{S}$ is a "*observational bisimulation up to $\approx$*" if $P\mathcal{S}Q$ implies, for all $\alpha \in Act$,

1. Whenever $P \overset{\alpha}{\rightarrow} P'$ then, for some Q', $Q \overset{\alpha}{\Rightarrow} Q'$ and $P' \sim \mathcal{S} \sim Q'$,
2. Whenever $Q \overset{\alpha}{\rightarrow} Q'$ then, for some P', $P \overset{\alpha}{\Rightarrow} P'$ and $P' \sim \mathcal{S} \sim Q'$.

Or formally,

```
OBS_BISIM_UPTO:
 ⊢ OBS_BISIM_UPTO Obsm ⟺
   ∀E E′.
     Obsm E E′ ⟹
     ∀u.
       (∀E₁.
          E −u→ E₁ ⟹
          ∃E₂.
            E′ =u⇒ E₂ ∧
            (WEAK_EQUIV ∘r Obsm ∘r STRONG_EQUIV) E₁ E₂) ∧
       ∀E₂.
         E′ −u→ E₂ ⟹
         ∃E₁.
           E =u⇒ E₁ ∧ (STRONG_EQUIV ∘r Obsm ∘r WEAK_EQUIV) E₁ E₂
```

Notice the removal of “hats” in above definition in comparasion with previous versions of “bisimulation up to $\approx$”, thus it’s the so far most concise definition among others.

Properties are similar. Then we proved a final theorem (there’s no lemma) for proving two processes are observational congruence: instead of proving it directly from definition, we only have to find an “observational bisimulation up to” which contains the pair of processes:

Theorem 4.4.1. *If $\mathcal{S}$ is a “observational bisimulation up to $\approx$”, then $\mathcal{S} \subseteq \approx^c$:*

$\mathtt{OBS\_BISIM\_UPTO\_THM}$:
$\vdash \mathtt{OBS\_BISIM\_UPTO}\ \mathit{Obsm} \implies \mathit{Obsm} \subseteq_r \mathtt{OBS\_CONGR}$

It was hard to prove above theorem, but it remains to find potential applications for this beautiful result.

Chapter 5

Unique Solutions of Equations

To prove two processes are (strong or weak) bisimilar, currently we have explored two kind of bisimulation proof methods: one is based on the definition: finding a bisimulation relation containing the given two processes, which is usually hard, while the problem may become easier using bisimilation up-to techniques; the other is to use those algebraic laws to derive the resulting conclusion step by step, although there's no decision procedure (algorithms) to construct such proofs automatically.

In Milner's 1989 book [2], he carefully explains that the bisimulation proof method is not supposed to be the only method for reasoning about bisimilarity. Indeed, various interesting examples in the book are handled using other techniques, notably *unique solution of equations*, whereby two tuples of processes are component-wise bisimilar if they are solutions of the same system of equations. The main conclusion is that, under a certain condition on the expression E, there is an unique P (up to $\sim$) such that

$$P \sim E\{P/X\}$$

That solution is, naturally, the agent A defined by $A \overset{\text{def}}{=} E\{A/X\}$.

Clearly this cannot be true for all E; in the case where E is just X, for example, *every* agant P satisfies the equation, because the equation is just $P \sim P$. But we shall see that the conclusion holds provided that X is weakly guarded in E, in which the concept of *weak guardness* will be explained soon in the following section.

For simplicity purposes we only consider single-variable equation in this project. There's no much additional work towards to multi-variable cases from the view of informal proofs in related books (see also [4], page 181–183). But to formally represent CCS equations with multiple variables, there may involves modified basic datatypes and many small theorems for recursively replacing those variables in CCS expressions. At the end of this thesis, the author has explored some approaches towards multi-variable equations, however, even the proofs of single-variable cases reflect the central ideas of this kind of results, given that even many textbooks and papers only prove these theorems for single-variable cases.

5.1 Guarded Expressions

If we consider only single-variable equations, it's straightforward to treat expressions like $E(X)$ as a λ-function taking any single CCS process returning another. In this way, no new datatypes were introduced, and actually that single variable X never appears alone in any formal proof, thus no need to represent it as a dedicated object.

In our previous work, to formalize the theory of congruence for CCS, we have defined the concept of "context" based on λ-calculus. There're actually two types of contexts: one-hole and multi-hole (this includes no-hole case):

Definition 5.1.1. (one-hole context of CCS) The (one-hole) semantic context of CCS is a function $C[\cdot]$ of type "`(α, β) context`" recursively defined by following rules:

```
OH_CONTEXT (λt. t)
OH_CONTEXT c ⟹ OH_CONTEXT (λt. a..c t)
OH_CONTEXT c ⟹ OH_CONTEXT (λt. c t + x)
OH_CONTEXT c ⟹ OH_CONTEXT (λt. x + c t)
OH_CONTEXT c ⟹ OH_CONTEXT (λt. c t ∥ x)
OH_CONTEXT c ⟹ OH_CONTEXT (λt. x ∥ c t)
OH_CONTEXT c ⟹ OH_CONTEXT (λt. ν L (c t))
OH_CONTEXT c ⟹ OH_CONTEXT (λt. relab (c t) rf)    [OH_CONTEXT_rules]
```

Definition 5.1.2. (multi-hole context of CCS) The semantic context of CCS is a function $C[\cdot]$ of type "`(α, β) context`" recursively defined by following rules:

```
CONTEXT (λt. t)
CONTEXT (λt. p)
CONTEXT e ⟹ CONTEXT (λt. a..e t)
CONTEXT e₁ ∧ CONTEXT e₂ ⟹ CONTEXT (λt. e₁ t + e₂ t)
CONTEXT e₁ ∧ CONTEXT e₂ ⟹ CONTEXT (λt. e₁ t ∥ e₂ t)
CONTEXT e ⟹ CONTEXT (λt. ν L (e t))
CONTEXT e ⟹ CONTEXT (λt. relab (e t) rf)              [CONTEXT_rules]
```

By repeatedly applying these inductive rules, one can imagine that, the "holes" in any CCS expressions at any depth, can be filled by the same process, when the λ function is called with that process. For CCS equations containing only one variable, we can simply treat a context as the core part of an equation. Thus, if C is a context, P is the solution of the equation (in case of strong equivalence), this actually means $P \sim C\ P$. Thus the variable itself never need to be formalized, and it never appears in any related proof.

Notice the difference with one-hole context in the branches of sum and parallel operators. And also the possibility that, the expression may finally contains no variable at all (this is necessary, otherwise we can't finish the proof).

One major drawback of above techniques is, we cannot further define the weakly guardedness (or sequential property) as a predicate of CCS expressions, simply because there's no way to recursively check the internal structure of λ-functions, as such functions were basically black-boxes once defined. A workaround solution is to define it independently and recursively:

Definition 5.1.3. (Weakly guarded expressions) X is *weakly guarded* in E if each occurrence of X is within some subexpression $\alpha.F$ of E:

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

Notice the only difference between weakly guarded expressions and normal expressions is at their first branch. In this way, a weakly guarded expression won't expose the variable without a prefixed action (could be τ).

To make a connection between above two kind of expressions (and the one-hole context), we have proved their relationships by induction on their structures:

Proposition 5.1.1. *(Relationship between one-hole/multi-hold contexts and weakly guarded expressions)*

1. *One-hole context is also context:*

   OH_CONTEXT_IS_CONTEXT:
   $\vdash \texttt{OH\_CONTEXT}\ c \Longrightarrow \texttt{CONTEXT}\ c$

2. *Weakly guarded expressions is also context:*

   WG_IS_CONTEXT:
   $\vdash \texttt{WG}\ e \Longrightarrow \texttt{CONTEXT}\ e$

Noticed that, the first result (and one-hole contexts) is never needed in the rest of this thesis, while the second one will be heavily used.

One limitation in our definitions is the lacking of CCS constants (i.e. `var` and `rec` operators defined as part of our CCS datatypes) in all above recursive definitions. This doesn't means the expressions cannot contains constants, just these constants must be irrelevant with the variable, that is, *variable substitutions never happen inside the body of any CCS constant!*. This restriction can be removed when we can prove the congruence of equivalences under `rec` operators, however this needs to discuss the "free variables" in CCS process, currently it's beyond the scope of this thesis.

5.2 Milner's three "unique solution of equations" theorems

Here we formalized three versions of the "unique solution of equations" theorem in Milner's book.

5.2.1 For strong equivalence

Based on results on bisimulation up to $\sim$, we have first proved the following non-trivial lemma. It states in effect that if X is weakly guarded in E, then the "first move" of E is independent of the agent substituted for X:

Lemma 5.2.1. *(Lemma 3.13 of [2]) If the variable X are weakly guarded in E, and $E\{P/X\} \overset{\alpha}{\to} P'$, then P' takes the form $E'\{P/X\}$ (for some expression E'), and moreover, for any Q, $E\{Q/X\} \overset{\alpha}{\to} E'\{Q/X\}$:*

```
STRONG_UNIQUE_SOLUTIONS_LEMMA:
⊢ WG E ⟹
   ∀P a P'.
     E P −a→ P' ⟹
     ∃E'. CONTEXT E' ∧ (P' = E' P) ∧ ∀Q. E Q −a→ E' Q
```

We're now ready to prove the following, the main proposition above the "unique solution of equations":

Theorem 5.2.1. *(Proposition 3.14 of [2]) Let the expression E contains at most the variable X, and let X be weakly guarded in E, then*

$$\textit{If } P \sim E\{P/X\} \textit{ and } Q \sim E\{Q/X\} \textit{ then } P \sim Q. \tag{5.1}$$

```
STRONG_UNIQUE_SOLUTIONS:
⊢ WG E ⟹ ∀P Q. P ∼ E P ∧ Q ∼ E Q ⟹ P ∼ Q
```

In above proof, we have identified 14 major sub-goals, dividing into 7 groups, in which each pairs are symmetric (thus having similar proof steps). The proof of this last theorem consists of 500 lines (each line usually have 2 or 3 HOL tactics, to make the proof not too long in lines).

5.2.2 For weak equivalence

Actually Milner's book contains only two "unique solutions of equations" theorems, one for strong equivalence, the other for observational congruence (with a wrong proof). But there's indeed a version for weak equivalence which shares a large portion of proof steps with the case for "observation congruence". The problem is, since weak equivalence is not congruence, to make correct statement of this theorem, we must slightly modify the concept of sequential expresses with further restrictions: no direct sums (or only guarded sums):

Definition 5.2.1. (Sequential expressions restricted with guarded sum)

```
GSEQ (λt. t)
GSEQ (λt. p)
GSEQ e ⟹ GSEQ (λt. a..e t)
GSEQ e₁ ∧ GSEQ e₂ ⟹ GSEQ (λt. a₁..e₁ t + a₂..e₂ t)        [GSEQ_rules]
```

The "unique solution of equations" theorem for weak equivalence requires more restrictions on the expression, namely, strong guardness:

Definition 5.2.2. ((Strongly) guarded expressions) X is (strongly) guarded in E if each occurrence of X is within some subexpression of E of the form $l.F$ (l is a visible action):

```
SG (λt. p)
CONTEXT e ⟹ SG (λt. label l..e t)
SG e ⟹ SG (λt. a..e t)
SG e₁ ∧ SG e₂ ⟹ SG (λt. e₁ t + e₂ t)
SG e₁ ∧ SG e₂ ⟹ SG (λt. e₁ t ∥ e₂ t)
SG e ⟹ SG (λt. ν L (e t))
SG e ⟹ SG (λt. relab (e t) rf)                          [SG_rules]
```

Now we're ready to state and prove the following lemma:

Lemma 5.2.2. *If the variable X are (strongly) guarded and sequential 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, $G\{Q/X\} \xrightarrow{\alpha} H\{Q/X\}$. Moreover H is sequential, and if $\alpha = \tau$ then H is also guarded.*

```
WEAK_UNIQUE_SOLUTIONS_LEMMA:
 ⊢ SG G ∧ GSEQ G ⟹
   ∀P a P′.
     G P −a→ P′ ⟹
     ∃H.
       GSEQ H ∧ ((a = τ) ⟹ SG H) ∧ (P′ = H P) ∧
       ∀Q. G Q −a→ H Q
```

An important technique in the proof of above lemma is the ability to do inductions on the structure of G which is both guarded and sequential. But if we apply the induction theorem generated by SG and GSEQ separately, the total numer of proof sub-goals is *huge*. Instead, we have proved the following combined induction theorems for SG+GSEQ expressions:

Proposition 5.2.1. *(Combined induction principle for guarded and sequential expressions)*

```
SG_GSEQ_strong_induction:
 ⊢ (∀p. R (λt. p)) ∧ (∀l e. GSEQ e ⟹ R (λt. label l..e t)) ∧
   (∀a e. SG e ∧ GSEQ e ∧ R e ⟹ R (λt. a..e t)) ∧
   (∀e₁ e₂.
      SG e₁ ∧ GSEQ e₁ ∧ R e₁ ∧ SG e₂ ∧ GSEQ e₂ ∧ R e₂ ⟹
      R (λt. τ..e₁ t + τ..e₂ t)) ∧
   (∀l₂ e₁ e₂.
      SG e₁ ∧ GSEQ e₁ ∧ R e₁ ∧ GSEQ e₂ ⟹
```

$$\begin{array}{l}
\quad\quad R\ (\lambda t.\ \tau\texttt{..}e_1\ t\ \texttt{+}\ \texttt{label}\ l_2\texttt{..}e_2\ t))\ \wedge \\
\quad (\forall l_1\ e_1\ e_2. \\
\quad\quad \texttt{GSEQ}\ e_1 \wedge \texttt{SG}\ e_2 \wedge \texttt{GSEQ}\ e_2 \wedge R\ e_2 \Longrightarrow \\
\quad\quad R\ (\lambda t.\ \texttt{label}\ l_1\texttt{..}e_1\ t\ \texttt{+}\ \tau\texttt{..}e_2\ t))\ \wedge \\
\quad (\forall l_1\ l_2\ e_1\ e_2. \\
\quad\quad \texttt{GSEQ}\ e_1 \wedge \texttt{GSEQ}\ e_2 \Longrightarrow \\
\quad\quad R\ (\lambda t.\ \texttt{label}\ l_1\texttt{..}e_1\ t\ \texttt{+}\ \texttt{label}\ l_2\texttt{..}e_2\ t)) \Longrightarrow \\
\quad \forall e.\ \texttt{SG}\ e \wedge \texttt{GSEQ}\ e \Longrightarrow R\ e
\end{array}$$

Once above lemma is proved, it's just one small step toward the target theorem:

Theorem 5.2.2. *(Unique solution of equations for weak equivalence) Let E be guarded and sequential expressions, and let $P \approx E\{P/X\}$, $Q \approx E\{Q/X\}$. Then $P \approx Q$.*

`WEAK_UNIQUE_SOLUTIONS`:

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

5.2.3 For observational congruence

For "unique solutions of equations" theorem of observational congruence, we have used a whole new technique, i.e. the following lemma:

Lemma 5.2.3. *To prove two processes E and E' are observational congruence, it's enough to construct a bisimulation with additional observational transition properties:*

`OBS_CONGR_BY_WEAK_BISIM`:

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

This lemma can be easily proved by definition of observational congruence and weak equivalence, however it's never mentioned in Milner's book. Actually we think it's impossible to prove the corresponding "unique solutions" theorem without this result.

What we have proved is the following one. Notice that, we didn't use any "bisimulation up-to" techniques, because they're not applicable once above theorem `OBS_CONGR_BY_WEAK_BISIM` is used. Instead we have directly constructed a bisimulation to finish the proof.

Lemma 5.2.4. *If the variable X are (strongly) guarded and sequential in G, and $G\{P/X\} \overset{\alpha}{\rightarrow} P'$, then P' takes the form $H\{P/X\}$ (for some expression H), and for any Q, $G\{Q/X\} \overset{\alpha}{\rightarrow} H\{Q/X\}$. Moreover H is sequential, and if $\alpha = \tau$ then H is also guarded.*

OBS_UNIQUE_SOLUTIONS_LEMMA:

$$\begin{array}{l} \vdash \texttt{SG}\ G \land \texttt{SEQ}\ G \Longrightarrow \\ \quad \forall P\ a\ P'. \\ \quad\quad G\ P\ {-a}\!\rightarrow\ P' \Longrightarrow \\ \quad\quad \exists H. \\ \quad\quad\quad \texttt{SEQ}\ H \land ((a = \tau) \Longrightarrow \texttt{SG}\ H) \land (P' = H\ P) \land \forall Q.\ G\ Q\ {-a}\!\rightarrow\ H\ Q \end{array}$$

Theorem 5.2.3. *(Unique solution of equations for observational congruence) Let E be guarded and sequential expressions, and let $P \approx^c E\{P/X\}$, $Q \approx^c E\{Q/X\}$. Then $P \approx^c Q$.*

OBS_UNIQUE_SOLUTIONS:

$$\vdash \texttt{SG}\ E \land \texttt{SEQ}\ E \Longrightarrow \forall P\ Q.\ P \approx^c E\ P \land Q \approx^c E\ Q \Longrightarrow P \approx^c Q$$

Chapter 6

Equations, Contractions and Unique Solutions

Here we have used the title of Prof. Sangiorgi's paper [3] as the title of this chapter, as the purpose of all the work mentioned in this chapter is to prove the key "unique solutions of contractions" theorem in that paper. Although the theorem looks similar with Milner's classical results, the underlying proof idea and techniques are completely different. And we have to formalize new fundamental transition concepts (Trace) in order to finish the proof.

In this chapter, we start with the formalization of another relation called "expansion", which can be seen as the origin of the "contraction" relation. We're going to prove that, the expansion relation between two processes (called 'expands') is a pre-congruence and pre-order.

Then we introduce the "contraction" relation and prove that, contractions between two processes (called 'contracts') have the same properties as expansion, with something more reasonable.

During the proof attempts of the "unique solution of contractions/expansions" theorem, we found that, sometimes we need to track the precise number of steps inside certain weak transitions, and such number will be passed into next proof steps during the challenging of contractions, expansions and weak equivalences. As a result, we have to further formalize the (strong) trace transitions between two CCS processes, and this involves HOL's `listTheory` into our formalization project. Many intermediate results using lists were proved during this work, while the final statement of neither the unique solutions theorems nor their lemmas need to explicitly mention lists (or traces). This is very interesting and unexpected, because, after having formalized so many results, plus Milner's original three "unique solutions of equations" theorems, we still need to invent new devices to finish the proof of Sangiorgi's results. Thus this last piece (and central piece) of thesis project has really made something new and goes beyond all previous achievements.

6.1 Expansion

Expansion is a non-symmetric relation for CCS processes. "P expands Q" means "P is at least as fast as Q", or more generally "Q uses at least as much resources as P". Expansion is studied by Arun-Kumar and Hennessy [25] under a different terminology: they show that expansion is a mathematically tractable preorder and has a complete proof system for finite terms based on a modification of the standard τ laws for CCS. In CCS, strong and weak bisimilarity are congruence relations (for weak bisimilarity, guarded sums are required), and expansion is a precongruence.

Its definition and properties are quite like weak equivalence. To define expansion, we need to follow a two-step processes, first define a predicate of relations, called "expansion", then define a 2-ary relation called 'contracts' as the maximal relation containing all expansions.

Definition 6.1.1. (expansion) A processes relation $\mathcal{R}$ is an *expansion* if, whenever $P \,\mathcal{R}\, Q$,

1. $P \xrightarrow{\mu} P'$ implies that there is Q' with $Q \xrightarrow{\hat{\mu}} Q'$ and $P' \,\mathcal{R}\, Q'$;
2. $Q \xrightarrow{\mu} Q'$ implies that there is P' with $P \overset{\mu}{\Rightarrow} P'$ and $P' \,\mathcal{R}\, Q'$.

or formally:

[EXPANSION]

$$\begin{array}{l}
\vdash \texttt{EXPANSION}\ \mathit{Exp} \iff \\
\quad \forall E\ E'\texttt{.} \\
\quad\quad \mathit{Exp}\ E\ E' \implies \\
\quad\quad (\forall l\texttt{.} \\
\quad\quad\quad (\forall E_1\texttt{.} \\
\quad\quad\quad\quad E\ -\texttt{label}\ l\!\rightarrow\ E_1 \implies \exists E_2\texttt{.}\ E'\ -\texttt{label}\ l\!\rightarrow\ E_2 \land \mathit{Exp}\ E_1\ E_2\texttt{)} \land \\
\quad\quad\quad \forall E_2\texttt{.} \\
\quad\quad\quad\quad E'\ -\texttt{label}\ l\!\rightarrow\ E_2 \implies \exists E_1\texttt{.}\ E\ =\texttt{label}\ l\!\Rightarrow\ E_1 \land \mathit{Exp}\ E_1\ E_2\texttt{)} \land \\
\quad\quad (\forall E_1\texttt{.} \\
\quad\quad\quad E\ -\tau\!\rightarrow\ E_1 \implies \mathit{Exp}\ E_1\ E' \lor \exists E_2\texttt{.}\ E'\ -\tau\!\rightarrow\ E_2 \land \mathit{Exp}\ E_1\ E_2\texttt{)} \land \\
\quad\quad \forall E_2\texttt{.}\ E'\ -\tau\!\rightarrow\ E_2 \implies \exists E_1\texttt{.}\ E\ =\tau\!\Rightarrow\ E_1 \land \mathit{Exp}\ E_1\ E_2
\end{array}$$

P expands Q, written $P \succeq_e Q$, if $P\mathcal{R}Q$ for some expansion $\mathcal{R}$.

Above definition for 'expands' can be simply characterized as the following statement:

Definition 6.1.2. (Original definition of 'expands')

$$\vdash P \succeq_e Q \iff \exists \mathit{Exp}\texttt{.}\ \mathit{Exp}\ P\ Q \land \texttt{EXPANSION}\ \mathit{Exp} \qquad \texttt{[expands\_thm]}$$

However it's not easy to derive its properties from such a definition (it's possible for sure). Instead we have used HOL's co-inductive relation package to define it co-inductively, the same way as in the case of weak equivalence. As a result, the above "original" definition now becomes a theorem as alternative definition.

We have proved that, expansion is contained in weak bisimulation and 'expands' is between strong and weak equivalence:

Proposition 6.1.1. *(Relationships between expansion, strong and weak equivalences)*

1. *expansion implies weak bisimulation:*

   $\vdash$ `EXPANSION` $\mathit{Exp} \Longrightarrow$ `WEAK_BISIM` Exp `[EXPANSION_IMP_WEAK_BISIM]`

2. *'expands' implies weak equivalence:*

   $\vdash P \succeq_e Q \Longrightarrow P \approx Q$ `[expands_IMP_WEAK_EQUIV]`

3. *Strong equivalence implies 'expands':*

   $\vdash P \sim Q \Longrightarrow P \succeq_e Q$ `[STRONG_EQUIV_IMP_expands]`

6.1.1 Expansion is pre-order

It's not hard to prove that, the 'expands' relation is pre-order, that is, transitive and reflexitive. Actually we proved these properties from the properties of the 'expansion' predicate: if the identity relation is expansion, then 'expands' is reflexitive; and if the composition of any two expansions is still an expansion, then for sure the 'expands' relation is transitive.

Proposition 6.1.2. *The composition of two expansions is still an expansion:*

`COMP_EXPANSION:`
$\vdash$ `EXPANSION` $\mathit{Exp}_1 \land$ `EXPANSION` $\mathit{Exp}_2 \Longrightarrow$ `EXPANSION` $(\mathit{Exp}_2 \circ_r \mathit{Exp}_1)$

Using this result, it's trivial to prove the transitivity of 'expands' relation:

Proposition 6.1.3. *(Transitivity of 'expands' relation)*

$\vdash x \succeq_e y \land y \succeq_e z \Longrightarrow x \succeq_e z$ `[expands_TRANS]`

The 'expands' relation is also reflexitive (accordingly the identity relation is also an expansion):

Proposition 6.1.4. *(Reflexitivity of 'expands' relation)*

$\vdash$ `EXPANSION (=)` `[IDENTITY_EXPANSION]`
$\vdash x \succeq_e x$ `[expands_TRANS]`

Combining above two results, we have proved that, 'expands' is a pre-order:

Lemma 6.1.1. *Bisimularity expandsion is a pre-order:*

$\vdash$ `PreOrder (expands)` `[expands_PreOrder]`

where

$\vdash$ `PreOrder` $R \iff$ `reflexive` $R \land$ `transitive` R `[PreOrder]`

6.1.2 Expansion is precongruence

Now we prove 'contracts' relation is a precongruence (a special version which requires guarded sum). To get this result, we have to prove that, the 'expands' relation is preserved by all CCS operators (except for `REC` which we have ignored in this project). For the case of sum operator, we can only prove it for guarded sums:

Proposition 6.1.5. *('expands' is precongruence)*

1. *'expands' is substitutive by prefix operator:*

   `expands_SUBST_PREFIX:`
   $\vdash E \succeq_e E' \Longrightarrow \forall u.\ u..E \succeq_e u..E'$

2. *'expands' is preserved by guarded sums:*

   `expands_PRESD_BY_GUARDED_SUM:`
   $\vdash E_1 \succeq_e E_1' \land E_2 \succeq_e E_2' \Longrightarrow a_1..E_1 + a_2..E_2 \succeq_e (a_1..E_1' + a_2..E_2')$

3. *'expands' is preserved by parallel composition:*

   `expands_PRESD_BY_PAR:`
   $\vdash E_1 \succeq_e E_1' \land E_2 \succeq_e E_2' \Longrightarrow E_1 \parallel E_2 \succeq_e E_1' \parallel E_2'$

4. *'expands' is substitutive by restrictions:*

   `expands_SUBST_RESTR:`
   $\vdash E \succeq_e E' \Longrightarrow \forall L.\ \nu\ L\ E \succeq_e \nu\ L\ E'$

5. *'expands' is substitutive by relabeling operator:*

   `expands_SUBST_RELAB:`
   $\vdash E \succeq_e E' \Longrightarrow \forall rf.\ \texttt{relab}\ E\ rf \succeq_e \texttt{relab}\ E'\ rf$

With above results, now we can inductively prove the 'expands' relation is preserved by any context with guarded sums, which by definition is a precongruce:

Theorem 6.1.1. *Bisimilarity expansion ('expands' relation) is precongruence, i.e. it's substitutive by semantics contexts (with restrictions of guarded sums):*

`expands_SUBST_GCONTEXT:`
$\vdash P \succeq_e Q \Longrightarrow \forall E.\ \texttt{GCONTEXT}\ E \Longrightarrow E\ P \succeq_e E\ Q$

or

`expands_precongruence:`
$\vdash$ `precongruence1 (expands)`

where

`precongruence1_def:`
$\vdash \texttt{precongruence1}\ R \iff$
$\quad \forall x\ y\ ctx.\ \texttt{GCONTEXT}\ ctx \Longrightarrow R\ x\ y \Longrightarrow R\ (ctx\ x)\ (ctx\ y)$

6.2 Contraction

Contraction is the new invention by Prof. Davide Sangiorgi. It's another non-symmetric relation. Roughly speaking, "P contracts Q" holds if "P is equivalent to Q" and, in addition, "Q has the possibility of being as efficient as P". That is, Q is capable of simulating P by performing less internal work. It is suggicient that Q has one 'efficient' path; Q could also have other paths that are slower than any path in P.

Its definition and properties are almost the same as expansions (and weak equivalence). To define contraction, we also need to follow a two-step processes: first we define a predicate of relations, called "(bisimulation) contraction", then we define a binary relation called 'contracts' (bisimilarity contraction) as the union of all contractions.

Definition 6.2.1. (Bisimulation contraction) A processes relation $\mathcal{R}$ is a *bisimulation contraction* if, whenever $P \mathcal{R} Q$,

1. $P \xrightarrow{\mu} P'$ implies that there is Q' with $Q \xrightarrow{\hat{\mu}} Q'$ and $P' \mathcal{R} Q'$;
2. $Q \xrightarrow{\mu} Q'$ implies that there is P' with $P \overset{\mu}{\Rightarrow} P'$ and $P' \approx Q'$.

or formally:

[CONTRACTION]

$$
\begin{aligned}
&\vdash \texttt{CONTRACTION}\ \mathit{Con} \iff \\
&\quad \forall E\ E'.\\
&\qquad \mathit{Con}\ E\ E' \Longrightarrow\\
&\qquad (\forall l.\\
&\qquad\quad (\forall E_1.\\
&\qquad\qquad E\ \texttt{−label}\ l\texttt{→}\ E_1 \Longrightarrow \exists E_2.\ E'\ \texttt{−label}\ l\texttt{→}\ E_2 \land \mathit{Con}\ E_1\ E_2) \land\\
&\qquad\quad \forall E_2.\ E'\ \texttt{−label}\ l\texttt{→}\ E_2 \Longrightarrow \exists E_1.\ E\ \texttt{=label}\ l\texttt{⇒}\ E_1 \land E_1 \approx E_2) \land\\
&\qquad (\forall E_1.\\
&\qquad\quad E\ -\tau\rightarrow\ E_1 \Longrightarrow \mathit{Con}\ E_1\ E' \lor \exists E_2.\ E'\ -\tau\rightarrow\ E_2 \land \mathit{Con}\ E_1\ E_2) \land\\
&\qquad \forall E_2.\ E'\ -\tau\rightarrow\ E_2 \Longrightarrow \exists E_1.\ E \overset{\epsilon}{\Rightarrow} E_1 \land E_1 \approx E_2
\end{aligned}
$$

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

Above definition for 'contracts' can be simply characterized as the following statement:

Definition 6.2.2. (Original definition of 'contracts')

$$\vdash P \succeq_{bis} Q \iff \exists \mathit{Con}.\ \mathit{Con}\ P\ Q \land \texttt{CONTRACTION}\ \mathit{Con}$$

Like the case of expansions, it's not easy to derive its properties from such a definition. Instead we have used HOL's co-inductive relation package to define it co-inductively, the same way as in the case of weak equivalence. As a result, the above "original" definition now becomes a theorem as alternative definition.

In the first clause Q is required to match P's challenge transition with at most one transition. This makes sure that Q is capable of mimicking P's work at least as efficiently as P. In contrast, the second clause of Def. 6.2.1, on the challenges from Q, entirely ignores efficiency: It is the same clause of weak bisimulation—the final derivatives are even required to be related by $\approx$ rather then by $\mathcal{R}$.

We can prove that, 'contraction' is contained in weak bisimulation, and the 'contracts' (bisimularity contraction) is between 'expands' and weak equivalence:

Proposition 6.2.1. *(Relationships between contraction, expansion and weak bisimulation)*

1. *'expands' implies 'contracts',*

   $\vdash P \succeq_e Q \Longrightarrow P \succeq_{bis} Q$ [expands_IMP_contracts]

2. *'contracts' implies weak equivalence ('contracts' is contained in weak equivalence).*

   $\vdash P \succeq_{bis} Q \Longrightarrow P \approx Q$ [contracts_IMP_WEAK_EQUIV]

The proof of all properties of contraction and 'contracts' are slightly harder than the case of expansion and 'expands', and we usually need to reduce the proof to corresponding properties of weak equivalence. Also, surprisingly, a contraction doesn't imply weak bisimulation, i.e. the following property doesn't hold:

$\forall\, Con.$ CONTRACTION $Con \Longrightarrow$ WEAK_BISIM Con

As a result, to finish the proof of contracts_IMP_WEAK_EQUIV, we do not prove Con itself is a weak bisimulation, but rather that Con "union" weak bisimilarity is a weak bisimulation. This is quite unexpected.

6.2.1 Contraction is pre-order

Following the same idea in the case of expansion, we prove that bisimularity contraction is a pre-order.

Proposition 6.2.2. *The composition of two contractions is still a contraction:*

$\vdash$ CONTRACTION $Con_1 \land$ CONTRACTION $Con_2 \Longrightarrow$
CONTRACTION ($Con_2 \circ_r Con_1$) [COMP_CONTRACTION]

Using this result, it's easy to prove the transitivity of 'contracts' relation:

Proposition 6.2.3. *The bisimularity contraction ('contrats' relation) is transitive:*

$\vdash x \succeq_{bis} y \land y \succeq_{bis} z \Longrightarrow x \succeq_{bis} z$ [contracts_TRANS]

'contracts' is also reflexitive (accordingly the identity relation is also an CONTRACTION):

Proposition 6.2.4. *Bisimilarity contraction is reflexive:*

$\vdash$ `CONTRACTION (=)` [IDENTITY_CONTRACTION]
$\vdash x \succeq_{bis} x$ [contracts_TRANS]

Combining above two results, we got:

Lemma 6.2.1. *Bisimularity contraction is a pre-order:*

$\vdash$ `PreOrder (contracts)` [contracts_PreOrder]

where

$\vdash$ `PreOrder` $R \iff$ `reflexive` $R \land$ `transitive` R [PreOrder]

6.2.2 Contraction is precongruence

Now we prove 'contracts' relation is a precongruence (a special version which requires guarded sum). To get this result, we have to prove that, the 'contracts' relation is preserved by all CCS operators (except for `REC` which we have ignored in this project). For the case of sum operator, we can only prove it for guarded sums:

Proposition 6.2.5. *('contracts' is precongruence)*

1. *'contracts' is substitutive by prefix operator:*

   `contracts_SUBST_PREFIX:`
   $\vdash E \succeq_{bis} E' \Longrightarrow \forall u.\ u..E \succeq_{bis} u..E'$

2. *'contracts' is preserved by guarded syms:*

   `contracts_PRESD_BY_GUARDED_SUM:`
   $\vdash E_1 \succeq_{bis} E_1' \land E_2 \succeq_{bis} E_2' \Longrightarrow a_1..E_1 + a_2..E_2 \succeq_{bis} a_1..E_1' + a_2..E_2'$

3. *'contracts' is preserved by parallel composition:*

   `contracts_PRESD_BY_PAR:`
   $\vdash E_1 \succeq_{bis} E_1' \land E_2 \succeq_{bis} E_2' \Longrightarrow E_1 \parallel E_2 \succeq_{bis} E_1' \parallel E_2'$

4. *'contracts' is substitutive by restrictions:*

   `contracts_SUBST_RESTR:`
   $\vdash E \succeq_{bis} E' \Longrightarrow \forall L.\ \nu\ L\ E \succeq_{bis} \nu\ L\ E'$

5. *'contracts' is substitutive by relabeling operator:*

   `contracts_SUBST_RELAB:`
   $\vdash E \succeq_{bis} E' \Longrightarrow \forall rf.$ `relab` $E\ rf \succeq_{bis}$ `relab` $E'\ rf$

With above results, now we can inductively prove the 'contracts' relation is preserved by any context with guarded sums, which by definition is a precongruence:

Theorem 6.2.1. *Bisimilarity contraction ('contracts' relation) is precongruence, i.e. it's substitutive by semantics contexts (with restrictions of guarded sums):*

`contracts_SUBST_GCONTEXT:`
$\vdash P \succeq_{bis} Q \implies \forall E.\ \texttt{GCONTEXT}\ E \implies E\ P \succeq_{bis} E\ Q$

or

`contracts_precongruence:`
$\vdash$ `precongruence1 (contracts)`

where

`precongruence1_def:`
$\vdash \texttt{precongruence1}\ R \iff$
$\quad \forall x\ y\ ctx.\ \texttt{GCONTEXT}\ ctx \implies R\ x\ y \implies R\ (ctx\ x)\ (ctx\ y)$

6.3 Step and Trace transitions

To finish the proof of "unique solutions of contractions" theorem, we need to ability to reason about the "length" of weak transitions. We want to make sure the weak transitions become smaller (or same length) after passing a weak equivalence or contraction. Such a requirement is unusual because no other theorems need it. At the beginning we defined a simple relation which only capture the "length" of general transitions between two processes, but it turns out to be useless, because between any two process there may be multiple transitions with the same length, and having the facts that "P weakly transits to Q" and "there's a n-step transition from P to Q", we actually know nothing about the length of that specific weak transition. But we kept the definition of the so-called "step" transition in case it's needed somehow in the future.

6.3.1 n-step transitions

The concept of "n-step transitions" can be defined as a numbered relation closure of its single transition:

Definition 6.3.1. (n-step transition) A n-step transition from P to Q is the numbered relation closure (NRC) of single step transition without respect to transition actions:

`STEP_def:`
$\vdash \texttt{STEP}\ P\ n\ Q \iff \texttt{NRC}\ (\lambda E\ E'.\ \exists u.\ E\ {-u\rightarrow}\ E')\ n\ P\ Q$

where

NRC:
NRC R 0 x y $\iff$ (x = y)
NRC R (SUC n) x y $\iff$ $\exists\, z.$ R x z $\land$ NRC R n z y

For this relation, we have some nice arithmetic-like properties proved here:

Proposition 6.3.1. *(Properties of n-step transitions)*

1. *0-step transition means equality:*

   $\vdash$ STEP x 0 y $\iff$ (x = y) [STEP0]

2. *1-step transition means single-step (strong) transition:*

   $\vdash$ STEP x 1 y $\iff$ $\exists\, u.$ x $-u\rightarrow$ y [STEP1]

3. *Reduce* $n+1$*-step transition to* n*-step transition:*

   STEP_SUC:
   $\vdash$ STEP x (SUC n) y $\iff$ $\exists\, z.$ ($\exists\, u.$ x $-u\rightarrow$ z) $\land$ STEP z n y

4. *Reduce* $n+1$*-step transition to* n*-step transition (another way):*

   STEP_SUC_LEFT:
   $\vdash$ STEP x (SUC n) y $\iff$ $\exists\, z.$ STEP x n z $\land$ $\exists\, u.$ z $-u\rightarrow$ y

5. *Reduce* $m+n$*-step transition to* m*-step and* n*-step transitions:*

   STEP_ADD_EQN:
   $\vdash$ STEP x (m + n) z $\iff$ $\exists\, y.$ STEP x m y $\land$ STEP y n z

What's more important is its relationship with EPS and weak transitions:

Lemma 6.3.1. *(Making n-step transitions from EPS and weak transitions)*

1. *EPS implies the existence of n-step transition with the same ends:*

   EPS_AND_STEP:
   $\vdash$ $E \overset{\epsilon}{\Rightarrow} E'$ $\implies$ $\exists\, n.$ STEP E n E'

2. *Weak transition implies the existence of n-step transition with the same ends:*

   WEAK_TRANS_AND_STEP:
   $\vdash$ E $=u\Rightarrow$ E' $\implies$ $\exists\, n.$ STEP E n E'

However, it must be understood that, there may be multiple paths (with same or different lengths) between two processes P and Q, thus the n-step transitions derived from above theorems may actually not reflect the path behind the initial weak transition (or EPS transition). This is why they're useless for proving the "unique solution of contractions" theorem. The n-step transition relation is simple, less powerfull but beautiful. It's never used in any proof in this thesis.

6.3.2 Traces

The successful formalization of Prof. Sangiorgi's "unique solution of contractions" theorem is completely based on successful definition of "Trace" in this thesis project. For an informal proof, it's not needed because the related arguments are quite obvious. Noticed that, trace equivalence is not covered by our projects. Also, the concept of "weak trace" is not defined anywhere.

Mathematically speaking, a trace transition is nothing but a special reflexive transitive closure (RTC) with a list accumulator, we call it LRTC:

Definition 6.3.2. (LRTC) A reflexitive transitive closure with list of transitions (LRTC) is a normal reflexitive transitive closure (RTC) of binary relations enhanced with a list, which accumulates all the transitions in the path:

$$\begin{array}{l}
\vdash \texttt{LRTC}\ R\ a\ l\ b \iff \\
\quad \forall P. \\
\qquad (\forall x.\ P\ x\ \texttt{[]}\ x) \land \\
\qquad (\forall x\ h\ y\ t\ z.\ R\ x\ h\ y \land P\ y\ t\ z \Longrightarrow P\ x\ (h\texttt{::}t)\ z) \Longrightarrow \\
\qquad P\ a\ l\ b
\end{array}$$

We have followed the proof idea in HOL's `relationTheory` and the proof sketches of normal RTC, and successfully proved many of its properties, including its induction, strong induction, rules and cases theorems:

Proposition 6.3.2. *(Properties of LRTC)*

1. *Induction principle of LRTC:*

[LRTC_INDUCT]

$$\begin{array}{l}
\vdash (\forall x.\ P\ x\ \texttt{[]}\ x) \land \\
\quad (\forall x\ h\ y\ t\ z.\ R\ x\ h\ y \land P\ y\ t\ z \Longrightarrow P\ x\ (h\texttt{::}t)\ z) \Longrightarrow \\
\quad \forall x\ l\ y.\ \texttt{LRTC}\ R\ x\ l\ y \Longrightarrow P\ x\ l\ y
\end{array}$$

2. *The 'rules' theorem of LRTC:*

[LRTC_RULES]

$$\begin{array}{l}
\texttt{LRTC}\ R\ x\ \texttt{[]}\ x \\
R\ x\ h\ y \land \texttt{LRTC}\ R\ y\ t\ z \Longrightarrow \texttt{LRTC}\ R\ x\ (h\texttt{::}t)\ z
\end{array}$$

3. *The strong induction principle of LRTC:*

[LRTC_STRONG_INDUCT]

$$\begin{array}{l}
\vdash (\forall x.\ P\ x\ \texttt{[]}\ x) \land \\
\quad (\forall x\ h\ y\ t\ z. \\
\qquad R\ x\ h\ y \land \texttt{LRTC}\ R\ y\ t\ z \land P\ y\ t\ z \Longrightarrow P\ x\ (h\texttt{::}t)\ z) \Longrightarrow \\
\quad \forall x\ l\ y.\ \texttt{LRTC}\ R\ x\ l\ y \Longrightarrow P\ x\ l\ y
\end{array}$$

4. *Reduce a $m+n$-step LRTC into m-step and n-step LRTCs:*

   [LRTC_LRTC]
   $\vdash$ LRTC R x m y $\Longrightarrow$ $\forall n$ z. LRTC R y n z $\Longrightarrow$ LRTC R x (m ++ n) z

5. *Transitivity of LRTC:*

   [LRTC_TRANS]
   $\vdash$ LRTC R x m y $\land$ LRTC R y n z $\Longrightarrow$ LRTC R x (m ++ n) z

6. *'cases' theorem of LRTC (using HD and TL of lists):*

   $\vdash$ LRTC R x l y $\iff$
   if NULL l **then** x = y
   else $\exists u$. R x (HD l) u $\land$ LRTC R u (TL l) y [LRTC_CASES1]

7. *'cases' theorem of LRTC (using FRONT and LAST of lists):*

   $\vdash$ LRTC R x l y $\iff$
   if NULL l **then** x = y
   else $\exists u$. LRTC R x (FRONT l) u $\land$ R u (LAST l) y [LRTC_CASES2]

Now we can simply define a trace as the LRTC of its single step transition, the "trace" between two CCS processes are stored into a list of actions. This is simpler than most of textbook which defines it inductively:

Definition 6.3.3. (Trace) A trace transition between two processes is the LRTC of single-step (strong) transition (with transition action accumulated into a list):

$\vdash$ TRACE = LRTC TRANS [TRACE_def]

Using above results of LRTC, we can easily derives the following properties of traces (which some of them are hard to get if we define the trace natively and inductively from the ground):

Proposition 6.3.3. *(Properties of trace)*

1. *The 'rules' theorem of trace:*

   [TRACE_rules]
   $\vdash$ ($\forall x$. TRACE x ϵ x) $\land$
   $\forall x$ h y t z. x $-h\rightarrow$ y $\land$ TRACE y t z $\Longrightarrow$ TRACE x (h::t) z

2. *Transitivity of traces:*

   [TRACE_trans]
   $\vdash$ TRACE x m y $\Longrightarrow$ $\forall n$ z. TRACE y n z $\Longrightarrow$ TRACE x (m ++ n) z

3. *Induction principle of traces:*

[TRACE_ind]

$$\begin{array}{l}\vdash (\forall x.\ P\ x\ \epsilon\ x) \land \\ \quad (\forall x\ h\ y\ t\ z.\ x\ {-h\rightarrow}\ y \land P\ y\ t\ z \Longrightarrow P\ x\ (h::t)\ z) \Longrightarrow \\ \quad \forall x\ l\ y.\ \texttt{TRACE}\ x\ l\ y \Longrightarrow P\ x\ l\ y\end{array}$$

4. *Strong induction principle of traces:*

[TRACE_strongind]

$$\begin{array}{l}\vdash (\forall x.\ P\ x\ \epsilon\ x) \land \\ \quad (\forall x\ h\ y\ t\ z. \\ \qquad x\ {-h\rightarrow}\ y \land \texttt{TRACE}\ y\ t\ z \land P\ y\ t\ z \Longrightarrow P\ x\ (h::t)\ z) \Longrightarrow \\ \quad \forall x\ l\ y.\ \texttt{TRACE}\ x\ l\ y \Longrightarrow P\ x\ l\ y\end{array}$$

5. *'cases' theorem of traces (using* `HD` *and* `TL` *of lists):*

[TRACE_cases1]

$$\begin{array}{l}\vdash \texttt{TRACE}\ x\ l\ y \iff \\ \quad \textbf{if}\ \texttt{NULL}\ l\ \textbf{then}\ x = y\ \textbf{else}\ \exists u.\ x\ {-\texttt{HD}\ l\rightarrow}\ u \land \texttt{TRACE}\ u\ (\texttt{TL}\ l)\ y\end{array}$$

6. *'cases' theorem of traces (using* `FRONT` *and* `LAST` *of lists):*

[TRACE_cases2]

$$\begin{array}{l}\vdash \texttt{TRACE}\ x\ l\ y \iff \\ \quad \textbf{if}\ \texttt{NULL}\ l\ \textbf{then}\ x = y \\ \quad \textbf{else}\ \exists u.\ \texttt{TRACE}\ x\ (\texttt{FRONT}\ l)\ u \land u\ {-\texttt{LAST}\ l\rightarrow}\ y\end{array}$$

7. *Breaking a trace (at any middle position) into two traces:*

[TRACE_cases_twice]

$$\begin{array}{l}\vdash \texttt{TRACE}\ x\ l\ y \iff \\ \quad \exists u\ l_1\ l_2.\ \texttt{TRACE}\ x\ l_1\ u \land \texttt{TRACE}\ u\ l_2\ y \land (l = l_1 \mathbin{+\!\!+} l_2)\end{array}$$

8. *Appending two traces:*

[TRACE_APPEND_cases]

$$\vdash \texttt{TRACE}\ x\ (l_1 \mathbin{+\!\!+} l_2)\ y \iff \exists u.\ \texttt{TRACE}\ x\ l_1\ u \land \texttt{TRACE}\ u\ l_2\ y$$

Here some list operations: $h::t$ means the list head h connected with the rest part (tail) t of the list. `HD` and `TL` are operators for getting the head and tail of a list, and $L_1 \mathbin{+\!\!+} L_2$ means the connection of two lists. `FRONT` return the parts of list without the last element, and `LAST` literally returns the last element of the list.

Our goal here is to be able to freely move between an EPS or weak transition, and the corresponding trace with exactly the same intermediate actions (thus with the same length). [1] Clearly every EPS or weak transition is also a trace, but the reverse is not always true. To become an EPS transition, all intermediate actions must be *tau*. As for weak transitions, if there're visible actions in the trace, it must be unique in the action list. To capture such properties, we have defined two helper concepts:

Definition 6.3.4. (No label and unique lebal)

1. NO_LABEL_def:
   $\vdash$ NO_LABEL $L \iff \neg\exists\, l.$ MEM (label l) L

2. UNIQUE_LABEL_def:
   $\vdash$ UNIQUE_LABEL u $L \iff$
   $\exists\, L_1\ L_2.$
   $(L_1 +\!\!+$ [u] $+\!\!+\ L_2 = L) \wedge$
   $\neg\exists\, l.$ MEM (label l) $L_1 \vee$ MEM (label l) L_2

The definition of "no label" seems straightforward, while the definition of "unique label" is quite smart.[2] Usually one can simply count the number of visible actions in the list and assert that number to be one, but the advantage of above definition is that, once we know that a visible action is unique in a list (or trace), from the definition we can immediately conclude that, the same action doesn't appear in the rest two parts of the list. And the related proofs become very straightforward.

NO_LABEL has the following lemma concerning its initial transition:

Lemma 6.3.2. *(Case of "no (visible) label") If there's no (visible) label in a list of actions, then either the first action in the list is τ, or the rest of actions has no labels:*

NO_LABEL_cases:
$\vdash$ NO_LABEL $(x::xs) \iff (x = \tau) \wedge$ NO_LABEL xs

UNIQUE_LABEL has two useful lemmas concerning its initial transition: (noticed that how a UNIQUE_LABEL becomes NO_LABEL after the first visible action)

Lemma 6.3.3. *(Two cases of "unique label")*

1. *If a (visible) label l is unique in a list of actions starting with τ (invisible action), then l is also unique in the rest of the list:*

   UNIQUE_LABEL_cases1:
   $\vdash$ UNIQUE_LABEL (label l) $(\tau::xs) \iff$ UNIQUE_LABEL (label l) xs

[1] Actually, even a trace may not be unique if we consider the possibility that, there're two different paths in the labeled transition system sharing the same initial and terminal nodes, also with the same intermediate action list. But to finish the proof we don't need to know the uniqueness at all.

[2] It's actually learnt from Robert Beers, an experienced HOL user. Now this definition has been added into HOL's `listTheory` with three common-used alternative definitions proved as equivalence theorems. This work was supported by Ramana Kumar, one of HOL maintainers.

2. *If a (visible) label l is unique in a list of actions starting with another visible label l', then $l = l'$ and the rest of actions has no (visible) labels:*

   `UNIQUE_LABEL_cases2:`

   $$\begin{array}{l} \vdash \texttt{UNIQUE\_LABEL (label } l\texttt{) (label } l'\texttt{::}xs\texttt{)} \iff \\ \quad (l \texttt{ = } l') \land \texttt{NO\_LABEL } xs \end{array}$$

What we have finally established here, is the precise condition for translating an EPS or weak transition from/to a trace transition:

Theorem 6.3.1. *(Trace, EPS and trace)*

1. *For any EPS transition, there exists a trace in which the action list has no labels:*

   [EPS_AND_TRACE]

   $$\vdash E \overset{\epsilon}{\Rightarrow} E' \iff \exists\, xs.\ \texttt{TRACE}\ E\ xs\ E' \land \texttt{NO\_LABEL}\ xs$$

2. *For any weak transition, there exists a trace in which the action list (not null) either has no labels or the label is unique:*

   [WEAK_TRANS_AND_TRACE]

   $$\begin{array}{l} \vdash E\ {=}u{\Rightarrow}\ E' \iff \\ \quad \exists\, us. \\ \qquad \texttt{TRACE}\ E\ us\ E' \land \neg\texttt{NULL}\ us \land \\ \qquad \textbf{if}\ u \texttt{ = } \tau\ \textbf{then}\ \texttt{NO\_LABEL}\ us\ \textbf{else}\ \texttt{UNIQUE\_LABEL}\ u\ us \end{array}$$

6.3.3 Traces for Expansions and Contractions

Whenever a trace (corresponding to an EPS or weak transition) passes an expansion or contraction, its length either remains the same or becomes shorter. But for our purposes here, we only care about those traces with no (visible) labels or unique (visible) labels, and the traces always pass through the expansion/contraction from left to right (i.e. from 1st to 2nd parameter). For simplicity we call them input and output traces.

Proposition 6.3.4. *(Traces passing expansions and contractions)*

1. *Whenever a no-labeled trace passes through an expansion, its length remains the same or becomes shorter, while the output trace has still no label:*

   `expands_AND_TRACE_tau:`

   $$\begin{array}{l} \vdash E \succeq_e E' \implies \\ \quad \forall\, xs\ l\ E_1. \\ \qquad \texttt{TRACE}\ E\ xs\ E_1 \land \texttt{NO\_LABEL}\ xs \implies \\ \qquad \exists\, xs'\ E_2. \\ \qquad\quad \texttt{TRACE}\ E'\ xs'\ E_2 \land E_1 \succeq_e E_2 \land \texttt{LENGTH}\ xs' \leq \texttt{LENGTH}\ xs \land \\ \qquad\quad \texttt{NO\_LABEL}\ xs' \end{array}$$

2. *Whenever an unique-labeled trace passes through an expansion, its length remains the same or becomes shorter, while the output trace has still unique label (which is the same one as input trace):*

   ```
   expands_AND_TRACE_label:
    ⊢ E ⪰_e E' ⟹
      ∀xs l E₁.
        TRACE E xs E₁ ∧ UNIQUE_LABEL (label l) xs ⟹
        ∃xs' E₂.
          TRACE E' xs' E₂ ∧ E₁ ⪰_e E₂ ∧ LENGTH xs' ≤ LENGTH xs ∧
          UNIQUE_LABEL (label l) xs'
   ```

3. *Whenever an no-labeled trace passes through a contraction, its length remains the same or becomes shorter, while the output trace has still no label:*

   ```
   contracts_AND_TRACE_tau:
    ⊢ E ⪰_bis E' ⟹
      ∀xs E₁.
        TRACE E xs E₁ ∧ NO_LABEL xs ⟹
        ∃xs' E₂.
          TRACE E' xs' E₂ ∧ E₁ ⪰_bis E₂ ∧ LENGTH xs' ≤ LENGTH xs ∧
          NO_LABEL xs'
   ```

4. *Whenever an unique-labeled trace passes through an contraction, its length remains the same or becomes shorter, while the output trace has still unique label (which is the same one as input trace):*

   ```
   contracts_AND_TRACE_label:
    ⊢ E ⪰_bis E' ⟹
      ∀xs l E₁.
        TRACE E xs E₁ ∧ UNIQUE_LABEL (label l) xs ⟹
        ∃xs' E₂.
          TRACE E' xs' E₂ ∧ E₁ ⪰_bis E₂ ∧ LENGTH xs' ≤ LENGTH xs ∧
          UNIQUE_LABEL (label l) xs'
   ```

Proof. It's only important to be noticed that, depending on the first transition, it's possible that a weak transition becomes an EPS transition after passing though the expansion/contraction. □

These results are essentially important for the proof of the "unique solutions of expansions (or contractions)" in next section.

6.4 Unique solutions of contractions

In this section, we describe the formal proof of Theorem 3.10 in Prof. Sangiorgi's paper [3]:

Theorem 6.4.1. *(unique solution of contractions for $\approx$) A system of weakly-guarded contractions has an unique solution for $\approx$.*

or formally:

UNIQUE_SOLUTIONS_OF_CONTRACTIONS:
$\vdash$ WGS $E \implies \forall P\ Q.\ P \succeq_{bis} E\ P \land Q \succeq_{bis} E\ Q \implies P \approx Q$

It must be noticed that, in our formal proof there's no "system of contractions", instead there's just one contraction, because currently we can only formalize equations (or expressions) with single process variable. Also, we must understand that "weakly-guarded" contractions with only "guarded sums". This is not a problem of the statement itself, because in Sangiorgi's paper the CCS grammar itself was defined to have only guarded sums. Our CCS grammar has general binary sums, instead, that's why we have "WGS" instead of "WG" in the formal statement of the theorem.

Here "WGS" means "weakly guarded expression (context) with only guarded sum", its recursive definition is similar with "WG":

Definition 6.4.1. (Weakly guarded expression with restriction of guarded sums) A *weakly guarded expression with restriction of guarded sums* is defined inductively as a λ-function:

WGS $(\lambda t.\ p)$
GCONTEXT $e \implies$ WGS $(\lambda t.\ a..e\ t)$
GCONTEXT $e_1 \land$ GCONTEXT $e_2 \implies$ WGS $(\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)$
WGS $e_1 \land$ WGS $e_2 \implies$ WGS $(\lambda t.\ e_1\ t \parallel e_2\ t)$
WGS $e \implies$ WGS $(\lambda t.\ \nu\ L\ (e\ t))$
WGS $e \implies$ WGS $(\lambda t.$ relab $(e\ t)\ rf)$ [WGS_rules]

where GCONTEXT is defined by

GCONTEXT $(\lambda t.\ t)$
GCONTEXT $(\lambda t.\ p)$
GCONTEXT $e \implies$ GCONTEXT $(\lambda t.\ a..e\ t)$
GCONTEXT $e_1 \land$ GCONTEXT $e_2 \implies$ GCONTEXT $(\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)$
GCONTEXT $e_1 \land$ GCONTEXT $e_2 \implies$ GCONTEXT $(\lambda t.\ e_1\ t \parallel e_2\ t)$
GCONTEXT $e \implies$ GCONTEXT $(\lambda t.\ \nu\ L\ (e\ t))$
GCONTEXT $e \implies$ GCONTEXT $(\lambda t.$ relab $(e\ t)\ rf)$ [GCONTEXT_rules]

And we have proved the following results:

Proposition 6.4.1. *A "WGS" is also an GCONTEXT, and its combination with a GCONTEXT gives another WGS:*

```
⊢ WGS e ⟹ GCONTEXT e                           [WGS_IS_GCONTEXT]
⊢ GCONTEXT c ∧ WGS e ⟹ WGS (c ∘ e)          [GCONTEXT_WGS_combin]
```

Actually the proof of above theorem itself is quite simple, what's not simple is the Lemma 3.9 before it: (also reduced to single-variable case)

Lemma 6.4.1. *Suppose P and Q are solutions for $\approx$ of a weakly-guarded contraction. For any context C, if $C[P] \overset{\mu}{\Rightarrow} R$, then there is a context C' such that $R \succeq_{bis} C'[P]$ and $C[Q] \overset{\hat{\mu}}{\approx} C'[Q]$.*

or formally:

```
UNIQUE_SOLUTIONS_OF_CONTRACTIONS_LEMMA:
 ⊢ (∃E. WGS E ∧ P ⪰bis E P ∧ Q ⪰bis E Q) ⟹
   ∀C.
     GCONTEXT C ⟹
     (∀l R.
        C P =label l⇒ R ⟹
        ∃C'.
          GCONTEXT C' ∧ R ⪰bis C' P ∧
          (WEAK_EQUIV ∘r (λx y. x =label l⇒ y)) (C Q) (C' Q)) ∧
     ∀R.
       C P =τ⇒ R ⟹
       ∃C'.
         GCONTEXT C' ∧ R ⪰bis C' P ∧
         (WEAK_EQUIV ∘r EPS) (C Q) (C' Q)
```

It's in this proof that we must use traces to capture the length of weak transitions. The formal proof also used the following four so-called "unfolding lemmas" to minimize the proof size for single theorem (otherwise it's very hard to replay and learn the entire single proof):

Lemma 6.4.2. *(Four "unfolding lemmas" used in proof of Lemma 6.4.1)*

```
unfolding_lemma1:
 ⊢ GCONTEXT E ∧ GCONTEXT C ∧ P ⪰bis E P ⟹
   ∀n. C P ⪰bis (C ∘ FUNPOW E n) P

unfolding_lemma2:
 ⊢ WGS E ⟹
   ∀P u P'.
     E P −u→ P' ⟹
     ∃C'. GCONTEXT C' ∧ (P' = C' P) ∧ ∀Q. E Q −u→ C' Q

unfolding_lemma3:
 ⊢ GCONTEXT C ∧ WGS E ⟹
   ∀P x P'.
```

```
    C (E P) −x→ P′ ⟹
    ∃ C′. GCONTEXT C′ ∧ (P′ = C′ P) ∧ ∀ Q. C (E Q) −x→ C′ Q
```

```
unfolding_lemma4:
 ⊢ GCONTEXT C ∧ WGS E ∧ TRACE ((C ∘ FUNPOW E n) P) xs P′ ∧
   LENGTH xs ≤ n ⟹
   ∃ C′.
     GCONTEXT C′ ∧ (P′ = C′ P) ∧
     ∀ Q. TRACE ((C ∘ FUNPOW E n) Q) xs (C′ Q)
```

The purpose of `unfolding_lemma1` is to make sure the contraction is preserved by n-times wrapping of contexts (`GCONTEXT` actually), and the proof only need to precongruence property of the 'contracts' relation. (Thus any precongruence or congruence relation fits the lemma). This lemma is directly used in the proof of main lemma, `UNIQUE_SOLUTIONS_OF_CONTRACTIONS_LEMMA`.

The purpose of `unfolding_lemma2` is to make sure the first transition from a weakly guarded expression do not come from the inside of its variable(s). This is very important, because otherwise we will not be able to freely change the value of process variable (and still form a valid transition). To prove this lemma, we have to do induction into the structure of a weakly-guarded expressions, and for the cases of sums, we have to do further inductions to look at each of its summands.

The purpose of `unfolding_lemma3` is a simple wrapper for `unfolding_lemma2`, to fit the use in `unfolding_lemma4`, in which the weakly-guarded expression is actually a composition of two expressions.

The purpose of `unfolding_lemma4` is to guarantee that, if the process variable is hiding deeply enough in the weakly guarded expressions, then after n times transition, it's still not touched. This actually completes the first half (most important part) of the informal proof of Lemma 3.9 of [3] (`UNIQUE_SOLUTIONS_OF_CONTRACTIONS_LEMMA`).

With all these lemmas (nicely separated out from the previous big informal proofs), the whole "unique solution of contractions" theorem get proved formally. Again, although we have only formalized its single-variable case, extending it to multiple variable cases informally is quite straightforward, so we can claim that, the formal verification of this result is a *success*.

In fact, we can also prove the "unique solutions of expansions" theorem in the same way, because in above proof, all used properties for "contracts" also hold for "expands". Therefore by almost the same steps, we have also proved the following results for expansion, together with its main lemma:

Theorem 6.4.2. *(Unique solution of expansions and its lemma)*

```
UNIQUE_SOLUTIONS_OF_EXPANSIONS_LEMMA:
 ⊢ (∃ E. WGS E ∧ P ⪰ₑ E P ∧ Q ⪰ₑ E Q) ⟹
   ∀ C.
     GCONTEXT C ⟹
     (∀ l R.
        C P =label l⇒ R ⟹
        ∃ C'.
          GCONTEXT C' ∧ R ⪰_e C' P ∧
          (WEAK_EQUIV ∘_r (λ x y. x =label l⇒ y)) (C Q) (C' Q)) ∧
    ∀ R.
       C P =τ⇒ R ⟹
       ∃ C'.
         GCONTEXT C' ∧ R ⪰_e C' P ∧
         (WEAK_EQUIV ∘_r EPS) (C Q) (C' Q)
```

```
UNIQUE_SOLUTIONS_OF_EXPANSIONS:
 ⊢ WGS E ⟹ ∀ P Q. P ⪰_e E P ∧ Q ⪰_e E Q ⟹ P ≈ Q
```

We have re-used 3 of the 4 above “unfolding lemmas”, because in their statements there’s “contracts” at all. For the first unfolding lemma, we have easily duplicated it into the following version for “expands”:

```
unfolding_lemma1':
 ⊢ GCONTEXT E ∧ GCONTEXT C ∧ P ⪰_e E P ⟹
   ∀ n. C P ⪰_e (C ∘ FUNPOW E n) P
```

But actually there’s a much easier way to prove “unique solutions of expansions” theorem, or the “unique solution” theorem for any relation contained in “contracts”. Here is the idea: (also formalized)

Proof. (Easy proof for Theorem 6.4.2) If P and Q are both solutions of an expansion, i.e. $P \succeq_e E[P]$ and $P \succeq_e E[Q]$, as we know expansion implies contraction, so we have also $P \succeq_{bis} E[P]$ and $P \succeq_{bis} E[Q]$. By “unique solutions of contractions” theorem, immediately we get $P \approx Q$. □

Here a natural question raised up: since we have also the theorem for “unique solution of expansions”, then why contractions? Is there any property which holds for only contraction but expansion? The answer is the following completeness theorem (Theorem 3.13 in [3]) that we cannot formalize right now (because it holds only for a system of contractions but single contraction):

Theorem 6.4.3. *(completeness) Suppose $\mathcal{R}$ is a bisimulation. Then there is a system of weakly-guarded pure contractions of which $\mathcal{R}_1$ and $\mathcal{R}_2$ are solutions for $\succeq_{bis}$. (Here $\mathcal{R}_i$ indicates the tuple obtained by projecting the pairs in $\mathcal{R}$ on the i-th component ($i = 1, 2$).*

We omit the proof here, while only quote the following remarks given by Prof. Sangiorgi in his paper:

Remark. In the final step of the proof above, relation $\overset{\hat{\mu_s}}{\Longrightarrow}$ comes from the definition of weak bisimulation, and could not be replaced by $\overset{\mu_s}{\Longrightarrow}$. This explains why *the completeness proof fails with expansion in place of contraction.*[3]

[3] However, Prof. Sangiorgi also said (on Oct 10, 2017) in a private mail that, “Is the difference

6.5 Observational contraction

In this thesis, we have slightly gone beyond the paper of Prof. Sangiorgi [3] by finding a slightly "better" contraction relation which is real precongruence beside all properties of the existing one, such that the restriction on guarded sums can be removed from all levels (CCS grammar, weakly guarded contractions, etc.). Fortunately such a relation does exist and was found in just one day. The quick finding is based on the following two principles:

1. Its definition must NOT be recursive (co-inductive), just like the definition of observation congruence ($\approx^c$);

2. It must be based on the existing 'contracts' relation, which we believe it's the 'right' one (become its completeness), just like weak equivalence (*approx*).

We call this new contraction relation "observational contraction" with the symbol $\succeq^c_{bis}$ (the small letter c indicates it's a (pre)congruence): (`OBS_contracts` in HOL), here is its definition:

Definition 6.5.1. (Observational contraction) Two processes E and E' has observational contraction relation (called "E observational contracts E'") if and only if

1. $E \overset{\mu}{\rightarrow} E_1$ implies that there is E_2 with $E' \overset{\mu}{\rightarrow} E_2$ and $E_1 \succeq_{\text{bis}} E_2$,

2. $E' \overset{\mu}{\rightarrow} E_2$ implies that there is E_1 with $E \overset{\mu}{\Rightarrow} E_1$ and $E_1 \approx E_2$.

or formally:

$$
\begin{array}{l}
\vdash E \succeq^c_{bis} E' \iff \\
\quad \forall u. \\
\quad\quad (\forall E_1.\ E \ -u\rightarrow\ E_1 \implies \exists E_2.\ E' \ -u\rightarrow\ E_2 \land E_1 \succeq_{bis} E_2) \land \\
\quad\quad \forall E_2.\ E' \ -u\rightarrow\ E_2 \implies \exists E_1.\ E \ =u\Rightarrow\ E_1 \land E_1 \approx E_2
\end{array}
$$

The primary property we wanted from this new relation is to make sure that it's preserved by direct sums. This is indeed true for this new relation:

Proposition 6.5.1. *Observational contraction is preserved by (direct) sums:*

`OBS_contracts_PRESD_BY_SUM`:

$$\vdash E_1 \succeq^c_{bis} E'_1 \land E_2 \succeq^c_{bis} E'_2 \implies E_1 + E_2 \succeq^c_{bis} E'_1 + E'_2$$

Following standard techniques (especially the proof ideas for "observational congruence"), we have also proved its preserving properties on other CCS constructions:

Proposition 6.5.2. *(Other properties of observational contraction)*

important for applications? I doubt. Contractions are a variation of expansion. However, completeness to me shows that contractions are the 'right' relation. I wish I had further evidence for that."

1. *Observational contraction is substitutive by prefix*

   `OBS_contracts_SUBST_PREFIX:`
   $\vdash E \succeq^{c}_{bis} E' \implies \forall u.\ u\texttt{..}E \succeq^{c}_{bis} u\texttt{..}E'$

2. *Observational contraction is preserved by parallel composition:*

   `OBS_contracts_PRESD_BY_PAR:`
   $\vdash E_1 \succeq^{c}_{bis} E'_1 \land E_2 \succeq^{c}_{bis} E'_2 \implies E_1 \parallel E_2 \succeq^{c}_{bis} E'_1 \parallel E'_2$

3. *Observational contraction is substitutive by restrictions:*

   `OBS_contracts_SUBST_RESTR:`
   $\vdash E \succeq^{c}_{bis} E' \implies \forall L.\ \nu\ L\ E \succeq^{c}_{bis} \nu\ L\ E'$

4. *Observational contraction is substitutive by relabeling operators:*

   `OBS_contracts_SUBST_RELAB:`
   $\vdash E \succeq^{c}_{bis} E' \implies \forall rf.\ \texttt{relab}\ E\ rf \succeq^{c}_{bis} \texttt{relab}\ E'\ rf$

Putting all together, observational contraction relation is a (real) precongruence:

Lemma 6.5.1. *Observational contraction is precongruence, i.e. it's substitutive by semantics contexts:*

`OBS_contracts_SUBST_CONTEXT:`
$\vdash P \succeq^{c}_{bis} Q \implies \forall E.\ \texttt{CONTEXT}\ E \implies E\ P \succeq^{c}_{bis} E\ Q$

or

$\vdash$ `precongruence OBS_contracts` [OBS_contracts_precongruence]

where

$\vdash \texttt{precongruence}\ R \iff$
$\quad \forall x\ y\ ctx.\ \texttt{CONTEXT}\ ctx \implies R\ x\ y \implies R\ (ctx\ x)\ (ctx\ y)$

The other potential problematic property is the transitivity, but fortunately this is also proven:

Proposition 6.5.3. *(Transitivity of observational contraction)*

$\vdash E \succeq^{c}_{bis} E' \land E' \succeq^{c}_{bis} E'' \implies E \succeq^{c}_{bis} E''$ [OBS_contracts_TRANS]

Together with the easily proved reflexivity, we show the "observational contraction" is indeed a pre-order:

Proposition 6.5.4. *Observational contraction is a pre-order:*

$\vdash$ `PreOrder (contracts)` [contracts_PreOrder]

On the relationship with other relations, we have the following results:

Proposition 6.5.5. *(Observational contraction and other relations)*

1. *Observational contraction implies contraction:*

   $\vdash E \succeq^c_{bis} E' \Longrightarrow E \succeq_{bis} E'$ [OBS_contracts_IMP_contracts]

2. *Observational contraction implies observational congruence:*

   $\vdash E \succeq^c_{bis} E' \Longrightarrow E \approx^c E'$ [OBS_contracts_IMP_OBS_CONGR]

Thus observational contraction is also contained in the existing 'contracts' relation, plus it's contained in observational congruence. The overall relationships we know so far looks like this:

$$\succeq_e \subset \succeq_{bis} \subset \approx$$
$$\succeq^c_{bis} \subset \succeq_{bis} \subset \approx$$
$$\succeq^c_{bis} \subset \approx^c \subset \approx$$

Noticed that, the strict subset relationship is not proved in our project, but they're indeed true (by showing some counterexamples). Also, we don't know if there's a subset relationship between expansion and observational contraction but we believe they have no such relationship at all.

Also, so far we haven't found any relation which is *coarser* than $\succeq_{bis}$, this probably means it's the "coarsest one" in some sense (although we don't no such results so far), and since $\succeq^c_{bis}$ is contained in $\succeq_{bis}$, its "unique solution" theorem is immediately available from the "unique solution of contractions" theorem, just like the case of "expansion":

`UNIQUE_SOLUTIONS_OF_OBS_CONTRACTIONS'`:
$\vdash$ `WGS` $E \Longrightarrow \forall P\ Q.\ P \succeq^c_{bis} E\ P \land Q \succeq^c_{bis} E\ Q \Longrightarrow P \approx Q$

However, we can do certainly much better: all restrictions on guarded sums can be removed from the proof of "unique solution of contractions" theorem, while we still have essentially the same proof. What we have finally proved is the following one, together with its main lemma:

```
UNIQUE_SOLUTIONS_OF_OBS_CONTRACTIONS_LEMMA:
 ⊢ (∃E. WG E ∧ P ⪰ᶜ_bis E P ∧ Q ⪰ᶜ_bis E Q) ⟹
   ∀C.
     CONTEXT C ⟹
     (∀l R.
        C P =label l⇒ R ⟹
        ∃C'.
```

$$\begin{array}{l}
\qquad\quad \texttt{CONTEXT}\ C' \land R \succeq_{bis} C'\ P\ \land \\
\qquad\quad (\texttt{WEAK\_EQUIV} \circ_r (\lambda x\ y.\ x \ \texttt{=label}\ l\texttt{⇒}\ y))\ (C\ Q)\ (C'\ Q)) \land \\
\forall R. \\
\quad C\ P\ \texttt{=}\tau\texttt{⇒}\ R \Longrightarrow \\
\quad \exists C'. \\
\qquad \texttt{CONTEXT}\ C' \land R \succeq_{bis} C'\ P\ \land \\
\qquad (\texttt{WEAK\_EQUIV} \circ_r \texttt{EPS})\ (C\ Q)\ (C'\ Q)
\end{array}$$

UNIQUE_SOLUTIONS_OF_OBS_CONTRACTIONS:

$$\vdash \texttt{WG}\ E \Longrightarrow \forall P\ Q.\ P \succeq^c_{bis} E\ P \land Q \succeq^c_{bis} E\ Q \Longrightarrow P \approx Q$$

The last theorem above is a beautiful and concise result, because it requires only a normal weakly-guarded expression, which is the same as in Milner's "unique solution of equations" theorem for strong equivalence.

To finish this formal proof, we have almost used everything, please notice that, in the conclusion of above main lemma, it still requires the normal "contracts" relation. This is because our definition of "observation contraction" is not recursive: after passing the first transition, it "downgrades" to the normal contraction, thus we need to use the properties of normal contractions to finish the rest of the proof. What's also needed, is the following properties related to trace:

OBS_contracts_AND_TRACE_tau:

$$\begin{array}{l}
\vdash E \succeq^c_{bis} E' \Longrightarrow \\
\quad \forall xs\ l\ E_1. \\
\qquad \texttt{TRACE}\ E\ xs\ E_1 \land \texttt{NO\_LABEL}\ xs \Longrightarrow \\
\qquad \exists xs'\ E_2. \\
\qquad\quad \texttt{TRACE}\ E'\ xs'\ E_2 \land E_1 \succeq_{bis} E_2 \land \texttt{LENGTH}\ xs' \le \texttt{LENGTH}\ xs\ \land \\
\qquad\quad \texttt{NO\_LABEL}\ xs'
\end{array}$$

OBS_contracts_AND_TRACE_label:

$$\begin{array}{l}
\vdash E \succeq^c_{bis} E' \Longrightarrow \\
\quad \forall xs\ l\ E_1. \\
\qquad \texttt{TRACE}\ E\ xs\ E_1 \land \texttt{UNIQUE\_LABEL (label}\ l\texttt{)}\ xs \Longrightarrow \\
\qquad \exists xs'\ E_2. \\
\qquad\quad \texttt{TRACE}\ E'\ xs'\ E_2 \land E_1 \succeq_{bis} E_2 \land \texttt{LENGTH}\ xs' \le \texttt{LENGTH}\ xs\ \land \\
\qquad\quad \texttt{UNIQUE\_LABEL (label}\ l\texttt{)}\ xs'
\end{array}$$

OBS_contracts_WEAK_TRANS_label':

$$\begin{array}{l}
\vdash E \succeq^c_{bis} E' \Longrightarrow \\
\quad \forall l\ E_2.\ E'\ \texttt{=label}\ l\texttt{⇒}\ E_2 \Longrightarrow \exists E_1.\ E\ \texttt{=label}\ l\texttt{⇒}\ E_1 \land E_1 \approx E_2
\end{array}$$

OBS_contracts_EPS':

$$\vdash E \succeq^c_{bis} E' \Longrightarrow \forall E_2.\ E' \overset{\epsilon}{\Rightarrow} E_2 \Longrightarrow \exists E_1.\ E \overset{\epsilon}{\Rightarrow} E_1 \land E_1 \approx E_2$$

Noticed how "observational contraction" becomes normal contraction in above statements, after passing the first transition step.

6.5.1 coarsest precongruence contained in $\succeq_{bis}$

So far we haven't examined any practical application in which our new "observational contraction" is meaningful and useful. However, it's natural to ask if there's another relation having all the properties of "observational contraction" while containing more process pairs (in another word, coarser than $\succeq^c_{bis}$. We want to prove our finding is the *coarsest* precongruence contained in $\succeq_{bis}$, just like observation congruence is the coarsest congruence contained in weak equivalence. Following the theory of congruence we have built for CCS, this proposition can be reduce to the following simpler form:

Proposition 6.5.6. *The "observational contraction" ($\succeq^c_{bis}$) is the coarsest precongruence contained in $\succeq_{bis}$ if and only if:*

$$\forall p\, q.\, p \succeq^c_{bis} q \Longleftrightarrow \forall r.\, p + r \succeq_{bis} q + r. \tag{6.1}$$

In the following formalizations, we only how to reduce the initial concern to above form, and formally prove above proposition from left to right (which is quite easy).

First of all, we can define a new relation (`C_contracts`) as the context closure of $\succeq_{bis}$ (also recall the definition of context closure),

$\vdash$ `C_contracts = CC (contracts)` [C_contracts]

$\vdash$ `CC` R = $(\lambda g\ h.\ \forall c.$ `CONTEXT` $c \Longrightarrow R\ (c\ g)\ (c\ h))$ [CC_def]

Then we can immediately show that, this new relation is indeed a (real) precongruence (because any congruence closure is automatically precongruence):

$\vdash$ `precongruence C_contracts` [C_contracts_precongruence]

We can also easily prove that, above new relation contains $\succeq^c_{bis}$, because it's the coarsest one (recall a congruence closure is always coarsest):

`OBS_contracts_IMP_C_contracts:`
$\vdash p \succeq^c_{bis} q \Longrightarrow$ `C_contracts` $p\ q$

Next, we define another new relation, which seems much larger, as the closure closed only on direct sums: (here we don't even know if it's a precongruence)

`SUM_contracts:`
$\vdash$ `SUM_contracts` = $(\lambda p\ q.\ \forall r.\ p + r \succeq_{bis} q + r)$

However, what we do know is, this new relation contains the previous one: (this actually proves the "left-to-right part" of our earlier proposition!)

`C_contracts_IMP_SUM_contracts:`
$\vdash$ `C_contracts` $p\ q \Longrightarrow$ `SUM_contracts` $p\ q$

Now we have a relation chain: OBS_contracts ⊆ C_contracts ⊆ SUM_contracts. To prove our "observational contraction" is the "coarsest one", i.e. it coincides with "C_contracts", it's sufficient to prove "SUM_contracts" implies "OBS_contracts":

$$\forall p\ q.\ \texttt{SUM\_contracts}\ p\ q \Longrightarrow p \succeq^{c}_{bis} q$$

which equals to the "right-to-left" part of our earlier proposition. This proof is actually quite like the case of "coarsest congruence contained in ≈" (Section 3.10). Whenever the two process didn't use up all available actions, we can prove the following easy version:

COARSEST_PRECONGR_THM:
$\vdash \texttt{free\_action}\ p \land \texttt{free\_action}\ q \Longrightarrow (p \succeq^{c}_{bis} q \iff \texttt{SUM\_contracts}\ p\ q)$

COARSEST_PRECONGR_THM':
$\vdash \texttt{free\_action}\ p \land \texttt{free\_action}\ q \Longrightarrow (p \succeq^{c}_{bis} q \iff \forall r.\ p + r \succeq_{bis} q + r)$

A further extension of this result for finite state CCS (based on Klop functions) should be also possible, but in this project we didn't try to prove it due to time limits.

Chapter 7

Towards multi-variable equations

As we have explained in the Introduction chapter, the existing CCS datatype in our formalization project can be seen as *expressions* with some free variables. Suppose E is such an expression with free variables $X_1, X_2, \ldots, X_n$ (there must be finite number of them because any CCS expression can only be finitary given there's no infinite sums, which is the current case), abbreviated as $E[\tilde{X}]$, then $X_1 \sim E[\tilde{X}]$ is CCS equations for strong equivalence. And suppose we have a system of expressions $\tilde{E}$, then $\tilde{X} \sim \tilde{E}[\tilde{X}]$ is a system of CCS equations.

To actually prove things about these equations, we need to at least two devices: 1) the ability to find all free variables in any CCS term, and 2) the ability to replace any free variable with specific CCS process (which shouldn't further contain any free variable).

The following recursive function can be used to retrieve a set of free variables in any CCS term:

```
FV nil = ∅
FV (u..p) = FV p
FV (p + q) = FV p ∪ FV q
FV (p || q) = FV p ∪ FV q
FV (ν L p) = FV p
FV (relab p rf) = FV p
FV (var X) = {X}
FV (rec X p) = FV p DIFF {X}
```

Similarity, we can also define a function for retrieving the set of all bound variables:

```
BV nil = ∅
BV (u..p) = BV p
BV (p + q) = BV p ∪ BV q
BV (p || q) = BV p ∪ BV q
BV (ν L p) = BV p
BV (relab p rf) = BV p
BV (var X) = ∅
BV (rec X p) = X INSERT BV p
```

However, the set of bound variables are not quite interesting, because a variable bounded somewhere in the term may be also free at another position in the term, thus the set of free and bound variables are not disjoint.

Sometimes we need to if a CCS term is an expression or process, which can be checked by the following function based on FV:

```
IS_PROC_def:
 ⊢ IS_PROC E ⟺ (FV E = ∅)
```

To replace the free variables in CCS terms with processes, we must clearly specify the correspondence between each variable and its replacement, thus the resulting (single) CCS process after all its free variables have been replaced, should be in forms like $E\{\tilde{p}/\tilde{X}\}$. In case some variables in $\tilde{X}$ didn't appear in E, it's natural to simple ignore the replacement operation, or the operation has no effects. Using constants defined in HOL's listTheory, such a substitution operation could be defined simply upon the existing CCS_Subst. However there will be all kinds of issues regarding the order of substitutions and it's hard to prove that the resulting process is unique for whatever permutations of the variable list. Another solution is to use finite maps to represent variable substitutions, to get rid of the ordering issues (suggested by Konrad Slind from HOL community):

```
CCS_Subst1 nil fm = nil
CCS_Subst1 (u..E) fm = u..CCS_Subst1 E fm
CCS_Subst1 (E1 + E2) fm = CCS_Subst1 E1 fm + CCS_Subst1 E2 fm
CCS_Subst1 (E1 ∥ E2) fm = CCS_Subst1 E1 fm ∥ CCS_Subst1 E2 fm
CCS_Subst1 (ν L E) fm = ν L (CCS_Subst1 E fm)
CCS_Subst1 (relab E rf) fm = relab (CCS_Subst1 E fm) rf
CCS_Subst1 (var Y) fm = if Y ∈ FDOM fm then fm ’ Y else var Y
CCS_Subst1 (rec Y E) fm =
if Y ∈ FDOM fm then rec Y E else rec Y (CCS_Subst1 E fm)
```

To further do substitutions on a list of CCS terms, we can define CCS_Subst2 which depends on CCS_Subst1:

```
CCS_Subst2_def:
 ⊢ CCS_Subst2 Es fm = MAP (λ e. CCS_Subst1 e fm) Es
```

Here we have used the constant MAP, which has the following recursive definition:

```
MAP f [] = []
MAP f (h::t) = f h::MAP f t
```

Thus it will call CCS_Subst1 with the same list of variables and their substitutions on each given CCS term, and then collect the resulting processes into a returning list. The type of CCS_Subst2 is

```
:(’a, ’b) CCS list -> (’a, ’b) CCS list -> ’a list -> (’a, ’b) CCS list
```

To define weakly guarded expressions, say E, we must make sure that all free variables in E are weakly guarded. If we ignore all other free variables and focus on just one of them, say, X, then it's possible to define weak guardedness of any CCS term in the following way:

```
weakly_guarded1_def:
 ⊢ weakly_guarded1 E ⇔
   ∀X. X ∈ FV E ⇒ ∀e. CONTEXT e ∧ (e (var X) = E) ⇒ WG e
```

Similarily we can also define other guardedness concepts in the same way. With these new devices, it's possible to formalize all "unique solution of equations" theorems with multi-variable equations, without touching existing formalization framework.

Chapter 8

Conclusions

In this thesis project, we have further formalized Milner's Calculus of Communicating Systems (CCS) using HOL theorem prover (HOL4). Beside classical results like the properties and algebraic laws of three equivalence relations (strong/weak bisimilarities and observational congruence), this work also includes a comprehensive formalization of "bisimulation up to", expansions and contractions, a theory of congruence for CCS, up to several versions of the "unique solution of equations (or contractions)" theorem. Some of the concepts and theorems were introduced by Prof. Davide Sangiorgi in his recent paper [3], and during this thesis we have further introduced new concept (observational contraction) and proved an more elegant form of its "unique solution of contraction" theorem. Therefore, to some extent, this thesis work has touched the current frontier in the research of concurrency theory and process algebra.

Although we chose to focus on single-variable equations in all "unique solutions" theorems, to minimize the development efforts, the resulting formal proofs actually have the same steps with the informal proofs, therefore we have confidence that the related informal proofs were all correct (so is our formalization). An extension to multi-variable cases is possible, without touching the CCS grammar (datatype) definition, as the free variables in CCS terms can be treated as equation variables, although this is not the current standard viewpoint.

Our formalization is based on the work done by Prof. Monica Nesi during 1992-1995 on Hol88 theorem prover, and by porting them into latest HOL4, we have successfully preserved this important work and make sure it's availability in the future (by submitting them into HOL's official code base). The use of HOL4's co-inductive relation and many builtin theories has demonstrated that HOL4's advantages over other theorem provers on the formalization of CCS or other process algebras.

8.1 Further directions

The *proof engineering* research in Concurrency Theory is far from fully resolved. Although we have quickly achieve a more-than-ever depth on the formalization of a simple process algebra (CCS), by covering several deep theorems, some important foundations in the work are still not perfectly built. And the author is planning to make further perfections in the future. Below is a list of plans:

1. **Infinite sum of CCS processes**. We want to precisely implement Milner's original CCS theory in which infinite sum of processes is included. Infinite sum is necessary for the proof of some theoretical results, and it's also needed for the encoding of Value-Passing CCS into Pure CCS. Currently this is limited by the datatype package of HOL4. Instead of manually define an CCS datatype with infinite sums in higher order logic, the author would like to improve HOL4's datatype defining abilities by implementing [12] (or porting its existing implementation in Isabelle/HOL), a compositional (co) datatype support based on category theory.

2. **Extending to multi-variable equations**. Currently we have only proved all "unique solutions of equations/contractions" theorems for single-variable equations. It is possible to prove the same theorems for systems of multi-variable equations in current framework. The congruence of equivalence with respect to recursive constructions, and supporting lemma for multi-variable substitutions must be formally proved first.

3. **Decision procedures for equivalence checking**. We want to implement the symbolic equivalence checking for CCS processes, and turn theorem prover into a model checking utility like the Concurrency Workbench (CWB). Instead of implementing the underlying algorithms directly in Standard ML, we would like to reduce the problem into BDD (Binary Decision Diagram) problems and call HOL's BDD library for the actual model checking.

Chapter 9

References

This chapter contains all definitions and theorems we have proved in this thesis project. They're separated by theories with reasonable names, ordered by their logical dependencies.

Some theorems are generated from existing theorems in forward way, they're also exported because from the view of HOL there's no difference between them and other manually proved theorems in backward ways. Thus there're some duplications in the sense that some theorems are just other theorems in separated (or combined) forms.

On the other side, each theorem has only its statement in HOL presented here. Their formal proofs are NOT presented here. (This is different with TEXexporting facility found in Isabelle, which exports both the statements and (usually human-readable) proofs into papers) This project has roughly 20,000 lines of proof scripts, which are exported into about 100 pages. Please keep in mind that, the number of these pages has no strict relationship with the complexity of the theorems being proved: it's possible for a very complicated theorem to have several thousands of line of proof code (which is not a good style though) but when exporting it into TEXpages, it's just a few lines! In this thesis project, the author tends to break large theorems into small pieces and prove them increasingly, and most of the meaningful intermediate theorems were also saved and exported.

Also noticed that, due to issues in HOL's TEXexporting facility, some theorems have slightly different forms with the same theroems appeared in previous chapters, with Unicode symbols replaced by their original ASCII-based names.

9.1 CCS Theory

Built: 08 Dicembre 2017
Parent Theories: real

9.1.1 Datatypes

```
CCS =
    nil
  | var 'a
  | prefix ('b Action) (('a, 'b) CCS)
  | (+) (('a, 'b) CCS) (('a, 'b) CCS)
  | par (('a, 'b) CCS) (('a, 'b) CCS)
  | ν ('b Label → bool) (('a, 'b) CCS)
  | relab (('a, 'b) CCS) ('b Relabeling)
  | rec 'a (('a, 'b) CCS)
```

```
Label = name 'b | coname 'b
```

9.1.2 Definitions

[ALL_IDENTICAL_def]

```
⊢ ∀t. ALL_IDENTICAL t ⇔ ∃x. ∀y. MEM y t ⇒ (y = x)
```

[ALL_PROC_def]

```
⊢ ∀Es. ALL_PROC Es ⇔ EVERY IS_PROC Es
```

[Apply_Relab_def]

```
⊢ (∀l. Apply_Relab [] l = l) ∧
  ∀newold ls l.
    Apply_Relab (newold::ls) l =
    if SND newold = l then FST newold
    else if COMPL (SND newold) = l then COMPL (FST newold)
    else Apply_Relab ls l
```

[BN_def]

```
⊢ (∀J. BN nil J = { }) ∧ (∀u p J. BN (u..p) J = BN p J) ∧
  (∀p q J. BN (p + q) J = BN p J ∪ BN q J) ∧
  (∀p q J. BN (p ∥ q) J = BN p J ∪ BN q J) ∧
  (∀L p J. BN (ν L p) J = BN p J ∪ L) ∧
  (∀p rf J. BN (relab p rf) J = BN p J) ∧
  (∀X J. BN (var X) J = { }) ∧
  ∀X p J.
    BN (rec X p) J =
    if MEM X J then
      FN (CCS_Subst p (rec X p) X) (DELETE_ELEMENT X J)
    else { }
```

[bound_names_def]

```
⊢ ∀p. bound_names p = BN p (SET_TO_LIST (BV p))
```

[BV_def]

```
⊢ (BV nil = { }) ∧ (∀u p. BV (u..p) = BV p) ∧
  (∀p q. BV (p + q) = BV p ∪ BV q) ∧
  (∀p q. BV (p ‖ q) = BV p ∪ BV q) ∧
  (∀L p. BV (ν L p) = BV p) ∧ (∀p rf. BV (relab p rf) = BV p) ∧
  (∀X. BV (var X) = { }) ∧ ∀X p. BV (rec X p) = X INSERT BV p
```

[CCS_Subst1_def]

```
⊢ (∀fm. CCS_Subst1 nil fm = nil) ∧
  (∀u E fm. CCS_Subst1 (u..E) fm = u..CCS_Subst1 E fm) ∧
  (∀E₁ E₂ fm.
     CCS_Subst1 (E₁ + E₂) fm =
     CCS_Subst1 E₁ fm + CCS_Subst1 E₂ fm) ∧
  (∀E₁ E₂ fm.
     CCS_Subst1 (E₁ ‖ E₂) fm =
     CCS_Subst1 E₁ fm ‖ CCS_Subst1 E₂ fm) ∧
  (∀L E fm. CCS_Subst1 (ν L E) fm = ν L (CCS_Subst1 E fm)) ∧
  (∀E rf fm.
     CCS_Subst1 (relab E rf) fm = relab (CCS_Subst1 E fm) rf) ∧
  (∀Y fm.
     CCS_Subst1 (var Y) fm =
     if Y ∈ FDOM fm then fm ' Y else var Y) ∧
  ∀Y E fm.
    CCS_Subst1 (rec Y E) fm =
    if Y ∈ FDOM fm then rec Y E else rec Y (CCS_Subst1 E fm)
```

[CCS_Subst2_def]

```
⊢ ∀Es fm. CCS_Subst2 Es fm = MAP (λe. CCS_Subst1 e fm) Es
```

[CCS_Subst_def]

```
⊢ (∀E′ X. CCS_Subst nil E′ X = nil) ∧
  (∀u E E′ X. CCS_Subst (u..E) E′ X = u..CCS_Subst E E′ X) ∧
  (∀E₁ E₂ E′ X.
     CCS_Subst (E₁ + E₂) E′ X =
     CCS_Subst E₁ E′ X + CCS_Subst E₂ E′ X) ∧
  (∀E₁ E₂ E′ X.
     CCS_Subst (E₁ ‖ E₂) E′ X =
     CCS_Subst E₁ E′ X ‖ CCS_Subst E₂ E′ X) ∧
  (∀L E E′ X.
     CCS_Subst (ν L E) E′ X = ν L (CCS_Subst E E′ X)) ∧
  (∀E f E′ X.
     CCS_Subst (relab E f) E′ X = relab (CCS_Subst E E′ X) f) ∧
  (∀Y E′ X.
     CCS_Subst (var Y) E′ X = if Y = X then E′ else var Y) ∧
   ∀ Y E E′ X.
     CCS_Subst (rec Y E) E′ X =
     if Y = X then rec Y E else rec Y (CCS_Subst E E′ X)
```

[COMPL_ACT_def]

```
 ⊢ (∀ l. COMPL (label l) = label (COMPL l)) ∧ (COMPL τ = τ)
```

[COMPL_LAB_def]

```
 ⊢ (∀ s. COMPL (name s) = coname s) ∧
   ∀ s. COMPL (coname s) = name s
```

[DELETE_ELEMENT_def]

```
 ⊢ (∀ e. DELETE_ELEMENT e [] = []) ∧
   ∀ e x l.
     DELETE_ELEMENT e (x::l) =
     if e = x then DELETE_ELEMENT e l else x::DELETE_ELEMENT e l
```

[free_names_def]

```
 ⊢ ∀ p. free_names p = FN p (SET_TO_LIST (BV p))
```

[FV_def]

```
 ⊢ (FV nil = { }) ∧ (∀ u p. FV (u..p) = FV p) ∧
   (∀ p q. FV (p + q) = FV p ∪ FV q) ∧
   (∀ p q. FV (p ∥ q) = FV p ∪ FV q) ∧
   (∀ L p. FV (ν L p) = FV p) ∧ (∀ p rf. FV (relab p rf) = FV p) ∧
   (∀ X. FV (var X) = {X}) ∧ ∀ X p. FV (rec X p) = FV p DIFF {X}
```

[IS_PROC_def]

```
 ⊢ ∀ E. IS_PROC E ⟺ (FV E = { })
```

[Is_Relabeling_def]

```
 ⊢ ∀ f. Is_Relabeling f ⟺ ∀ s. f (coname s) = COMPL (f (name s))
```

[RELAB_def]

```
 ⊢ ∀ labl. RELAB labl = ABS_Relabeling (Apply_Relab labl)
```

[relabel_def]

```
 ⊢ (∀ rf. relabel rf τ = τ) ∧
   ∀ rf l. relabel rf (label l) = label (REP_Relabeling rf l)
```

[Relabeling_ISO_DEF]

```
 ⊢ (∀ a. ABS_Relabeling (REP_Relabeling a) = a) ∧
   ∀ r.
     Is_Relabeling r ⟺ (REP_Relabeling (ABS_Relabeling r) = r)
```

[Relabeling_TY_DEF]

$$\vdash \exists\, rep.\ \texttt{TYPE\_DEFINITION Is\_Relabeling}\ rep$$

[size_def]

$$\vdash \forall\, p.\ \texttt{size}\ p = \texttt{CCS\_size}\ (\lambda\, x.\ 0)\ (\lambda\, x.\ 0)\ p$$

[TRANS_def]

$$\begin{array}{l}
\vdash \texttt{TRANS} = \\
\quad (\lambda\, a_0\ a_1\ a_2. \\
\qquad \forall\, \mathit{TRANS}'. \\
\qquad\quad (\forall\, a_0\ a_1\ a_2. \\
\qquad\qquad (a_0 = a_1 .. a_2) \lor \\
\qquad\qquad (\exists\, E\ E'.\ (a_0 = E + E') \land \mathit{TRANS}'\ E\ a_1\ a_2) \lor \\
\qquad\qquad (\exists\, E\ E'.\ (a_0 = E' + E) \land \mathit{TRANS}'\ E\ a_1\ a_2) \lor \\
\qquad\qquad (\exists\, E\ E_1\ E'. \\
\qquad\qquad\quad (a_0 = E \parallel E') \land (a_2 = E_1 \parallel E') \land \mathit{TRANS}'\ E\ a_1\ E_1) \lor \\
\qquad\qquad (\exists\, E\ E_1\ E'. \\
\qquad\qquad\quad (a_0 = E' \parallel E) \land (a_2 = E' \parallel E_1) \land \mathit{TRANS}'\ E\ a_1\ E_1) \lor \\
\qquad\qquad (\exists\, E\ l\ E_1\ E'\ E_2. \\
\qquad\qquad\quad (a_0 = E \parallel E') \land (a_1 = \tau) \land (a_2 = E_1 \parallel E_2) \land \\
\qquad\qquad\quad \mathit{TRANS}'\ E\ (\texttt{label}\ l)\ E_1 \land \\
\qquad\qquad\quad \mathit{TRANS}'\ E'\ (\texttt{label}\ (\texttt{COMPL}\ l))\ E_2) \lor \\
\qquad\qquad (\exists\, E\ E'\ l\ L. \\
\qquad\qquad\quad (a_0 = \nu\ L\ E) \land (a_2 = \nu\ L\ E') \land \mathit{TRANS}'\ E\ a_1\ E' \land \\
\qquad\qquad\quad ((a_1 = \tau) \lor \\
\qquad\qquad\quad\ (a_1 = \texttt{label}\ l) \land l \notin L \land \texttt{COMPL}\ l \notin L)) \lor \\
\qquad\qquad (\exists\, E\ u\ E'\ \mathit{rf}. \\
\qquad\qquad\quad (a_0 = \texttt{relab}\ E\ \mathit{rf}) \land (a_1 = \texttt{relabel}\ \mathit{rf}\ u) \land \\
\qquad\qquad\quad (a_2 = \texttt{relab}\ E'\ \mathit{rf}) \land \mathit{TRANS}'\ E\ u\ E') \lor \\
\qquad\qquad (\exists\, E\ X. \\
\qquad\qquad\quad (a_0 = \texttt{rec}\ X\ E) \land \\
\qquad\qquad\quad \mathit{TRANS}'\ (\texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X)\ a_1\ a_2) \implies \\
\qquad\qquad \mathit{TRANS}'\ a_0\ a_1\ a_2) \implies \\
\qquad\quad \mathit{TRANS}'\ a_0\ a_1\ a_2)
\end{array}$$

9.1.3 Theorems

[Action_11]

$$\vdash \forall\, x\ y.\ (\texttt{label}\ x = \texttt{label}\ y) \iff (x = y)$$

[Action_distinct]

$$\vdash \forall\, x.\ \tau \neq \texttt{label}\ x$$

[Action_distinct_label]

$$\vdash \forall\, x.\ \texttt{label}\ x \neq \tau$$

[Action_induction]

$\vdash \forall P.\ P\ \tau \land (\forall a.\ P\ (\texttt{label}\ a)) \implies \forall x.\ P\ x$

[Action_nchotomy]

$\vdash \forall \mathit{opt}.\ (\mathit{opt} = \tau) \lor \exists x.\ \mathit{opt} = \texttt{label}\ x$

[Action_no_tau_is_Label]

$\vdash \forall A.\ A \neq \tau \implies \exists x.\ A = \texttt{label}\ x$

[Apply_Relab_COMPL_THM]

$\vdash \forall \mathit{labl}\ s.$
$\quad \texttt{Apply\_Relab}\ \mathit{labl}\ (\texttt{coname}\ s) =$
$\quad \texttt{COMPL}\ (\texttt{Apply\_Relab}\ \mathit{labl}\ (\texttt{name}\ s))$

[APPLY_RELAB_THM]

$\vdash \forall \mathit{labl}'\ \mathit{labl}.$
$\quad (\texttt{RELAB}\ \mathit{labl}' = \texttt{RELAB}\ \mathit{labl}) \iff$
$\quad (\texttt{Apply\_Relab}\ \mathit{labl}' = \texttt{Apply\_Relab}\ \mathit{labl})$

[CCS_caseeq]

$\vdash (\texttt{CCS\_CASE}\ x\ v\ f\ f_1\ f_2\ f_3\ f_4\ f_5\ f_6 = v') \iff$
$\quad (x = \texttt{nil}) \land (v = v') \lor (\exists a.\ (x = \texttt{var}\ a) \land (f\ a = v')) \lor$
$\quad (\exists o'\ C'.\ (x = o'..C') \land (f_1\ o'\ C' = v')) \lor$
$\quad (\exists C'\ C_0.\ (x = C' + C_0) \land (f_2\ C'\ C_0 = v')) \lor$
$\quad (\exists C'\ C_0.\ (x = C' \parallel C_0) \land (f_3\ C'\ C_0 = v')) \lor$
$\quad (\exists f'\ C'.\ (x = \nu\ f'\ C') \land (f_4\ f'\ C' = v')) \lor$
$\quad (\exists C'\ R.\ (x = \texttt{relab}\ C'\ R) \land (f_5\ C'\ R = v')) \lor$
$\quad \exists a\ C'.\ (x = \texttt{rec}\ a\ C') \land (f_6\ a\ C' = v')$

[CCS_COND_CLAUSES]

$\vdash \forall t_1\ t_2.$
$\quad ((\textbf{if}\ \texttt{T}\ \textbf{then}\ t_1\ \textbf{else}\ t_2) = t_1) \land$
$\quad ((\textbf{if}\ \texttt{F}\ \textbf{then}\ t_1\ \textbf{else}\ t_2) = t_2)$

[CCS_distinct']

$\vdash (\forall a.\ \texttt{nil} \neq \texttt{var}\ a) \land (\forall a_1\ a_0.\ \texttt{nil} \neq a_0..a_1) \land$
$\quad (\forall a_1\ a_0.\ \texttt{nil} \neq a_0 + a_1) \land (\forall a_1\ a_0.\ \texttt{nil} \neq a_0 \parallel a_1) \land$
$\quad (\forall a_1\ a_0.\ \texttt{nil} \neq \nu\ a_0\ a_1) \land (\forall a_1\ a_0.\ \texttt{nil} \neq \texttt{relab}\ a_0\ a_1) \land$
$\quad (\forall a_1\ a_0.\ \texttt{nil} \neq \texttt{rec}\ a_0\ a_1) \land (\forall a_1\ a_0\ a.\ \texttt{var}\ a \neq a_0..a_1) \land$
$\quad (\forall a_1\ a_0\ a.\ \texttt{var}\ a \neq a_0 + a_1) \land (\forall a_1\ a_0\ a.\ \texttt{var}\ a \neq a_0 \parallel a_1) \land$
$\quad (\forall a_1\ a_0\ a.\ \texttt{var}\ a \neq \nu\ a_0\ a_1) \land$
$\quad (\forall a_1\ a_0\ a.\ \texttt{var}\ a \neq \texttt{relab}\ a_0\ a_1) \land$
$\quad (\forall a_1\ a_0\ a.\ \texttt{var}\ a \neq \texttt{rec}\ a_0\ a_1) \land$
$\quad (\forall a_1'\ a_1\ a_0'\ a_0.\ a_0..a_1 \neq a_0' + a_1') \land$
$\quad (\forall a_1'\ a_1\ a_0'\ a_0.\ a_0..a_1 \neq a_0' \parallel a_1') \land$
$\quad (\forall a_1'\ a_1\ a_0'\ a_0.\ a_0..a_1 \neq \nu\ a_0'\ a_1') \land$

$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0\texttt{..}a_1 \neq \texttt{relab}\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0\texttt{..}a_1 \neq \texttt{rec}\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0\ \texttt{+}\ a_1 \neq a_0' \parallel a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0\ \texttt{+}\ a_1 \neq \nu\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0\ \texttt{+}\ a_1 \neq \texttt{relab}\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0\ \texttt{+}\ a_1 \neq \texttt{rec}\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0 \parallel a_1 \neq \nu\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0 \parallel a_1 \neq \texttt{relab}\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0 \parallel a_1 \neq \texttt{rec}\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \nu\ a_0\ a_1 \neq \texttt{relab}\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \nu\ a_0\ a_1 \neq \texttt{rec}\ a_0'\ a_1') \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{relab}\ a_0\ a_1 \neq \texttt{rec}\ a_0'\ a_1') \land$
$(\forall a.\ \texttt{var}\ a \neq \texttt{nil}) \land (\forall a_1\ a_0.\ a_0\texttt{..}a_1 \neq \texttt{nil}) \land$
$(\forall a_1\ a_0.\ a_0\ \texttt{+}\ a_1 \neq \texttt{nil}) \land (\forall a_1\ a_0.\ a_0 \parallel a_1 \neq \texttt{nil}) \land$
$(\forall a_1\ a_0.\ \nu\ a_0\ a_1 \neq \texttt{nil}) \land (\forall a_1\ a_0.\ \texttt{relab}\ a_0\ a_1 \neq \texttt{nil}) \land$
$(\forall a_1\ a_0.\ \texttt{rec}\ a_0\ a_1 \neq \texttt{nil}) \land (\forall a_1\ a_0\ a.\ a_0\texttt{..}a_1 \neq \texttt{var}\ a) \land$
$(\forall a_1\ a_0\ a.\ a_0\ \texttt{+}\ a_1 \neq \texttt{var}\ a) \land (\forall a_1\ a_0\ a.\ a_0 \parallel a_1 \neq \texttt{var}\ a) \land$
$(\forall a_1\ a_0\ a.\ \nu\ a_0\ a_1 \neq \texttt{var}\ a) \land$
$(\forall a_1\ a_0\ a.\ \texttt{relab}\ a_0\ a_1 \neq \texttt{var}\ a) \land$
$(\forall a_1\ a_0\ a.\ \texttt{rec}\ a_0\ a_1 \neq \texttt{var}\ a) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0'\ \texttt{+}\ a_1' \neq a_0\texttt{..}a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0' \parallel a_1' \neq a_0\texttt{..}a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \nu\ a_0'\ a_1' \neq a_0\texttt{..}a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{relab}\ a_0'\ a_1' \neq a_0\texttt{..}a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{rec}\ a_0'\ a_1' \neq a_0\texttt{..}a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ a_0' \parallel a_1' \neq a_0\ \texttt{+}\ a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \nu\ a_0'\ a_1' \neq a_0\ \texttt{+}\ a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{relab}\ a_0'\ a_1' \neq a_0\ \texttt{+}\ a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{rec}\ a_0'\ a_1' \neq a_0\ \texttt{+}\ a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \nu\ a_0'\ a_1' \neq a_0 \parallel a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{relab}\ a_0'\ a_1' \neq a_0 \parallel a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{rec}\ a_0'\ a_1' \neq a_0 \parallel a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{relab}\ a_0'\ a_1' \neq \nu\ a_0\ a_1) \land$
$(\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{rec}\ a_0'\ a_1' \neq \nu\ a_0\ a_1) \land$
$\forall a_1'\ a_1\ a_0'\ a_0.\ \texttt{rec}\ a_0'\ a_1' \neq \texttt{relab}\ a_0\ a_1$

[CCS_Subst_rec]

$\vdash \forall X\ E\ E'.\ \texttt{CCS\_Subst}\ (\texttt{rec}\ X\ E)\ E'\ X\ \texttt{=}\ \texttt{rec}\ X\ E$

[CCS_Subst_var]

$\vdash \forall X\ E.\ \texttt{CCS\_Subst}\ (\texttt{var}\ X)\ E\ X\ \texttt{=}\ E$

[COMPL_COMPL_ACT]

$\vdash \forall a.\ \texttt{COMPL}\ (\texttt{COMPL}\ a)\ \texttt{=}\ a$

[COMPL_COMPL_LAB]

$\vdash \forall l.\ \texttt{COMPL}\ (\texttt{COMPL}\ l)\ \texttt{=}\ l$

```
[COMPL_THM]
 ⊢ ∀l s.
     (l ≠ name s ⟹ COMPL l ≠ coname s) ∧
     (l ≠ coname s ⟹ COMPL l ≠ name s)

[coname_COMPL]
 ⊢ ∀s. coname s = COMPL (name s)

[DELETE_ELEMENT_APPEND]
 ⊢ ∀a L L′.
     DELETE_ELEMENT a (L ++ L′) =
     DELETE_ELEMENT a L ++ DELETE_ELEMENT a L′

[DELETE_ELEMENT_FILTER]
 ⊢ ∀e L. DELETE_ELEMENT e L = FILTER (λy. e ≠ y) L

[EVERY_DELETE_ELEMENT]
 ⊢ ∀e L P. P e ∧ EVERY P (DELETE_ELEMENT e L) ⟹ EVERY P L

[EXISTS_Relabeling]
 ⊢ ∃f. Is_Relabeling f

[FN_def]
 ⊢ (∀J. FN nil J = {}) ∧
   (∀p l J. FN (label l..p) J = l INSERT FN p J) ∧
   (∀p J. FN (τ..p) J = FN p J) ∧
   (∀q p J. FN (p + q) J = FN p J ∪ FN q J) ∧
   (∀q p J. FN (p ∥ q) J = FN p J ∪ FN q J) ∧
   (∀p L J. FN (ν L p) J = FN p J DIFF (L ∪ IMAGE COMPL L)) ∧
   (∀rf p J.
      FN (relab p rf) J = IMAGE (REP_Relabeling rf) (FN p J)) ∧
   (∀X J. FN (var X) J = {}) ∧
   ∀p X J.
     FN (rec X p) J =
     if MEM X J then
       FN (CCS_Subst p (rec X p) X) (DELETE_ELEMENT X J)
     else {}

[FN_ind]
 ⊢ ∀P.
     (∀J. P nil J) ∧ (∀l p J. P p J ⟹ P (label l..p) J) ∧
     (∀p J. P p J ⟹ P (τ..p) J) ∧
     (∀p q J. P p J ∧ P q J ⟹ P (p + q) J) ∧
     (∀p q J. P p J ∧ P q J ⟹ P (p ∥ q) J) ∧
     (∀L p J. P p J ⟹ P (ν L p) J) ∧
     (∀p rf J. P p J ⟹ P (relab p rf) J) ∧ (∀X J. P (var X) J) ∧
     (∀X p J.
        (MEM X J ⟹
         P (CCS_Subst p (rec X p) X) (DELETE_ELEMENT X J)) ⟹
        P (rec X p) J) ⟹
     ∀v v₁. P v v₁
```

[FN_UNIV1]

$$\vdash \forall p.\ \texttt{free\_names}\ p \neq \mathcal{U}(\texttt{:'b Label}) \Longrightarrow \exists a.\ a \notin \texttt{free\_names}\ p$$

[FN_UNIV2]

$$\begin{array}{l}\vdash \forall p\ q.\\ \qquad \texttt{free\_names}\ p \cup \texttt{free\_names}\ q \neq \mathcal{U}(\texttt{:'b Label}) \Longrightarrow\\ \qquad \exists a.\ a \notin \texttt{free\_names}\ p \land a \notin \texttt{free\_names}\ q\end{array}$$

[IS_LABEL_def]

$$\vdash (\forall x.\ \texttt{IS\_SOME}\ (\texttt{label}\ x) \iff \texttt{T}) \land (\texttt{IS\_SOME}\ \tau \iff \texttt{F})$$

[IS_RELABELING]

$$\vdash \forall \mathit{labl}.\ \texttt{Is\_Relabeling}\ (\texttt{Apply\_Relab}\ \mathit{labl})$$

[Label_caseeq]

$$\begin{array}{l}\vdash (\texttt{Label\_CASE}\ x\ f\ f_1 = v) \iff\\ \quad (\exists b.\ (x = \texttt{name}\ b) \land (f\ b = v)) \lor\\ \quad \exists b.\ (x = \texttt{coname}\ b) \land (f_1\ b = v)\end{array}$$

[LABEL_def]

$$\vdash \forall x.\ \texttt{LABEL}\ (\texttt{label}\ x) = x$$

[Label_distinct']

$$\vdash \forall a'\ a.\ \texttt{coname}\ a' \neq \texttt{name}\ a$$

[Label_not_eq]

$$\vdash \forall a'\ a.\ (\texttt{name}\ a = \texttt{coname}\ a') \iff \texttt{F}$$

[Label_not_eq']

$$\vdash \forall a'\ a.\ (\texttt{coname}\ a' = \texttt{name}\ a) \iff \texttt{F}$$

[LENGTH_DELETE_ELEMENT_LE]

$$\vdash \forall e\ L.\ \texttt{MEM}\ e\ L \Longrightarrow \texttt{LENGTH}\ (\texttt{DELETE\_ELEMENT}\ e\ L) < \texttt{LENGTH}\ L$$

[LENGTH_DELETE_ELEMENT_LEQ]

$$\vdash \forall e\ L.\ \texttt{LENGTH}\ (\texttt{DELETE\_ELEMENT}\ e\ L) \leq \texttt{LENGTH}\ L$$

[NIL_NO_TRANS]

$$\vdash \forall u\ E.\ \neg(\texttt{nil}\ -u\rightarrow\ E)$$

[NIL_NO_TRANS_EQF]

$$\vdash \forall u\ E.\ \texttt{nil}\ -u\rightarrow\ E \iff \texttt{F}$$

[NOT_IN_DELETE_ELEMENT]

$$\vdash \forall e\ L.\ \neg\texttt{MEM}\ e\ (\texttt{DELETE\_ELEMENT}\ e\ L)$$

[PAR1]

$$\vdash \forall E\ u\ E_1\ E'.\ E\ -u\rightarrow\ E_1 \implies E \parallel E'\ -u\rightarrow\ E_1 \parallel E'$$

[PAR2]

$$\vdash \forall E\ u\ E_1\ E'.\ E\ -u\rightarrow\ E_1 \implies E' \parallel E\ -u\rightarrow\ E' \parallel E_1$$

[PAR3]

$$\begin{array}{l}\vdash \forall E\ l\ E_1\ E'\ E_2.\\ \qquad E\ -\texttt{label}\ l\rightarrow\ E_1 \land E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\rightarrow\ E_2 \implies\\ \qquad E \parallel E'\ -\tau\rightarrow\ E_1 \parallel E_2\end{array}$$

[PAR_cases]

$$\begin{array}{l}\vdash \forall D\ D'\ u\ D''.\\ \qquad D \parallel D'\ -u\rightarrow\ D'' \implies\\ \qquad (\exists E\ E_1\ E'.\\ \qquad\quad ((D = E) \land (D' = E')) \land (D'' = E_1 \parallel E') \land E\ -u\rightarrow\ E_1) \lor\\ \qquad (\exists E\ E_1\ E'.\\ \qquad\quad ((D = E') \land (D' = E)) \land (D'' = E' \parallel E_1) \land E\ -u\rightarrow\ E_1) \lor\\ \qquad \exists E\ l\ E_1\ E'\ E_2.\\ \qquad\ \ ((D = E) \land (D' = E')) \land (u = \tau) \land (D'' = E_1 \parallel E_2) \land\\ \qquad\ \ E\ -\texttt{label}\ l\rightarrow\ E_1 \land E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\rightarrow\ E_2\end{array}$$

[PAR_cases_EQ]

$$\begin{array}{l}\vdash \forall D\ D'\ u\ D''.\\ \qquad D \parallel D'\ -u\rightarrow\ D'' \iff\\ \qquad (\exists E\ E_1\ E'.\\ \qquad\quad ((D = E) \land (D' = E')) \land (D'' = E_1 \parallel E') \land E\ -u\rightarrow\ E_1) \lor\\ \qquad (\exists E\ E_1\ E'.\\ \qquad\quad ((D = E') \land (D' = E)) \land (D'' = E' \parallel E_1) \land E\ -u\rightarrow\ E_1) \lor\\ \qquad \exists E\ l\ E_1\ E'\ E_2.\\ \qquad\ \ ((D = E) \land (D' = E')) \land (u = \tau) \land (D'' = E_1 \parallel E_2) \land\\ \qquad\ \ E\ -\texttt{label}\ l\rightarrow\ E_1 \land E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\rightarrow\ E_2\end{array}$$

[PREFIX]

$$\vdash \forall E\ u.\ u..E\ -u\rightarrow\ E$$

[REC]

$$\vdash \forall E\ u\ X\ E_1.\ \texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X\ -u\rightarrow\ E_1 \implies \texttt{rec}\ X\ E\ -u\rightarrow\ E_1$$

[REC_cases]

$$\begin{array}{l}\vdash \forall X\ E\ u\ E''.\\ \qquad \texttt{rec}\ X\ E\ -u\rightarrow\ E'' \implies\\ \qquad \exists E'\ X'.\\ \qquad\ \ ((X = X') \land (E = E')) \land\\ \qquad\ \ \texttt{CCS\_Subst}\ E'\ (\texttt{rec}\ X'\ E')\ X'\ -u\rightarrow\ E''\end{array}$$

[REC_cases_EQ]

$$\begin{array}{l}\vdash \forall X\ E\ u\ E''.\\ \quad \texttt{rec}\ X\ E\ -u\rightarrow\ E'' \iff\\ \quad \exists E'\ X'.\\ \qquad ((X = X') \land (E = E')) \land\\ \qquad \texttt{CCS\_Subst}\ E'\ (\texttt{rec}\ X'\ E')\ X'\ -u\rightarrow\ E''\end{array}$$

[RELAB_cases]

$$\begin{array}{l}\vdash \forall E\ rf\ a_1\ a_2.\\ \quad \texttt{relab}\ E\ rf\ -a_1\rightarrow\ a_2 \implies\\ \quad \exists E'\ u\ E''\ rf'.\\ \qquad ((E = E') \land (rf = rf')) \land (a_1 = \texttt{relabel}\ rf'\ u) \land\\ \qquad (a_2 = \texttt{relab}\ E''\ rf') \land E'\ -u\rightarrow\ E''\end{array}$$

[RELAB_cases_EQ]

$$\begin{array}{l}\vdash \forall E\ rf\ a_1\ a_2.\\ \quad \texttt{relab}\ E\ rf\ -a_1\rightarrow\ a_2 \iff\\ \quad \exists E'\ u\ E''\ rf'.\\ \qquad ((E = E') \land (rf = rf')) \land (a_1 = \texttt{relabel}\ rf'\ u) \land\\ \qquad (a_2 = \texttt{relab}\ E''\ rf') \land E'\ -u\rightarrow\ E''\end{array}$$

[Relab_label]

$$\vdash \forall rf\ u\ l.\ (\texttt{relabel}\ rf\ u = \texttt{label}\ l) \implies \exists l'.\ u = \texttt{label}\ l'$$

[RELAB_NIL_NO_TRANS]

$$\vdash \forall rf\ u\ E.\ \neg(\texttt{relab nil}\ rf\ -u\rightarrow\ E)$$

[Relab_tau]

$$\vdash \forall rf\ u.\ (\texttt{relabel}\ rf\ u = \tau) \implies (u = \tau)$$

[RELABELING]

$$\vdash \forall E\ u\ E'\ rf.\ E\ -u\rightarrow\ E' \implies \texttt{relab}\ E\ rf\ -\texttt{relabel}\ rf\ u\rightarrow\ \texttt{relab}\ E'\ rf$$

[REP_Relabeling_THM]

$$\vdash \forall rf.\ \texttt{Is\_Relabeling}\ (\texttt{REP\_Relabeling}\ rf)$$

[RESTR]

$$\begin{array}{l}\vdash \forall E\ u\ E'\ l\ L.\\ \quad E\ -u\rightarrow\ E' \land\\ \quad ((u = \tau) \lor (u = \texttt{label}\ l) \land l \notin L \land \texttt{COMPL}\ l \notin L) \implies\\ \quad \nu\ L\ E\ -u\rightarrow\ \nu\ L\ E'\end{array}$$

[RESTR_cases]

$$\begin{array}{l}\vdash \forall D'\ u\ L\ D.\\ \qquad \nu\ L\ D\ -u\rightarrow\ D' \implies\\ \qquad \exists E\ E'\ l\ L'.\\ \qquad\quad ((L = L') \land (D = E)) \land (D' = \nu\ L'\ E') \land E\ -u\rightarrow\ E' \land\\ \qquad\quad ((u = \tau) \lor (u = \texttt{label}\ l) \land l \notin L' \land \texttt{COMPL}\ l \notin L')\end{array}$$

[RESTR_cases_EQ]

$$\begin{array}{l}\vdash \forall D'\ u\ L\ D.\\ \qquad \nu\ L\ D\ -u\rightarrow\ D' \iff\\ \qquad \exists E\ E'\ l\ L'.\\ \qquad\quad ((L = L') \land (D = E)) \land (D' = \nu\ L'\ E') \land E\ -u\rightarrow\ E' \land\\ \qquad\quad ((u = \tau) \lor (u = \texttt{label}\ l) \land l \notin L' \land \texttt{COMPL}\ l \notin L')\end{array}$$

[RESTR_LABEL_NO_TRANS]

$$\begin{array}{l}\vdash \forall l\ L.\\ \qquad l \in L \lor \texttt{COMPL}\ l \in L \implies \forall E\ u\ E'.\ \neg(\nu\ L\ (\texttt{label}\ l..E)\ -u\rightarrow\ E')\end{array}$$

[RESTR_NIL_NO_TRANS]

$$\vdash \forall L\ u\ E.\ \neg(\nu\ L\ \texttt{nil}\ -u\rightarrow\ E)$$

[SUM1]

$$\vdash \forall E\ u\ E_1\ E'.\ E\ -u\rightarrow\ E_1 \implies E + E'\ -u\rightarrow\ E_1$$

[SUM2]

$$\vdash \forall E\ u\ E_1\ E'.\ E\ -u\rightarrow\ E_1 \implies E' + E\ -u\rightarrow\ E_1$$

[SUM_cases]

$$\begin{array}{l}\vdash \forall D\ D'\ u\ D''.\\ \qquad D + D'\ -u\rightarrow\ D'' \implies\\ \qquad (\exists E\ E'.\ ((D = E) \land (D' = E')) \land E\ -u\rightarrow\ D'') \lor\\ \qquad \exists E\ E'.\ ((D = E') \land (D' = E)) \land E\ -u\rightarrow\ D''\end{array}$$

[SUM_cases_EQ]

$$\begin{array}{l}\vdash \forall D\ D'\ u\ D''.\\ \qquad D + D'\ -u\rightarrow\ D'' \iff\\ \qquad (\exists E\ E'.\ ((D = E) \land (D' = E')) \land E\ -u\rightarrow\ D'') \lor\\ \qquad \exists E\ E'.\ ((D = E') \land (D' = E)) \land E\ -u\rightarrow\ D''\end{array}$$

[TRANS_ASSOC_EQ]

$$\vdash \forall E\ E'\ E''\ E_1\ u.\ E + E' + E''\ -u\rightarrow\ E_1 \iff E + (E' + E'')\ -u\rightarrow\ E_1$$

[TRANS_ASSOC_RL]

$$\vdash \forall E\ E'\ E''\ E_1\ u.\ E + (E' + E'')\ -u\rightarrow\ E_1 \implies E + E' + E''\ -u\rightarrow\ E_1$$

[TRANS_cases]

$$\begin{array}{l}
\vdash \forall a_0\ a_1\ a_2.\\
\quad a_0\ -a_1\rightarrow\ a_2 \iff\\
\quad (a_0 = a_1\texttt{..}a_2) \lor (\exists E\ E'.\ (a_0 = E + E') \land E\ -a_1\rightarrow\ a_2) \lor\\
\quad (\exists E\ E'.\ (a_0 = E' + E) \land E\ -a_1\rightarrow\ a_2) \lor\\
\quad (\exists E\ E_1\ E'.\ (a_0 = E \parallel E') \land (a_2 = E_1 \parallel E') \land E\ -a_1\rightarrow\ E_1) \lor\\
\quad (\exists E\ E_1\ E'.\ (a_0 = E' \parallel E) \land (a_2 = E' \parallel E_1) \land E\ -a_1\rightarrow\ E_1) \lor\\
\quad (\exists E\ l\ E_1\ E'\ E_2.\\
\qquad (a_0 = E \parallel E') \land (a_1 = \tau) \land (a_2 = E_1 \parallel E_2) \land\\
\qquad E\ -\texttt{label}\ l\rightarrow\ E_1 \land E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\rightarrow\ E_2) \lor\\
\quad (\exists E\ E'\ l\ L.\\
\qquad (a_0 = \nu\ L\ E) \land (a_2 = \nu\ L\ E') \land E\ -a_1\rightarrow\ E' \land\\
\qquad ((a_1 = \tau) \lor (a_1 = \texttt{label}\ l) \land l \notin L \land \texttt{COMPL}\ l \notin L)) \lor\\
\quad (\exists E\ u\ E'\ rf.\\
\qquad (a_0 = \texttt{relab}\ E\ rf) \land (a_1 = \texttt{relabel}\ rf\ u) \land\\
\qquad (a_2 = \texttt{relab}\ E'\ rf) \land E\ -u\rightarrow\ E') \lor\\
\quad \exists E\ X.\ (a_0 = \texttt{rec}\ X\ E) \land \texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X\ -a_1\rightarrow\ a_2
\end{array}$$

[TRANS_COMM_EQ]

$$\vdash \forall E\ E'\ E''\ u.\ E + E'\ -u\rightarrow\ E'' \iff E' + E\ -u\rightarrow\ E''$$

[TRANS_IMP_NO_NIL]

$$\vdash \forall E\ u\ E'.\ E\ -u\rightarrow\ E' \implies E \neq \texttt{nil}$$

[TRANS_IMP_NO_NIL']

$$\vdash \forall E\ u\ E'.\ E\ -u\rightarrow\ E' \implies E \neq \texttt{nil}$$

[TRANS_IMP_NO_RESTR_NIL]

$$\vdash \forall E\ u\ E'.\ E\ -u\rightarrow\ E' \implies \forall L.\ E \neq \nu\ L\ \texttt{nil}$$

[TRANS_ind]

$$\begin{array}{l}
\vdash \forall \mathit{TRANS}'.\\
\quad (\forall E\ u.\ \mathit{TRANS}'\ (u\texttt{..}E)\ u\ E) \land\\
\quad (\forall E\ u\ E_1\ E'.\ \mathit{TRANS}'\ E\ u\ E_1 \implies \mathit{TRANS}'\ (E + E')\ u\ E_1) \land\\
\quad (\forall E\ u\ E_1\ E'.\ \mathit{TRANS}'\ E\ u\ E_1 \implies \mathit{TRANS}'\ (E' + E)\ u\ E_1) \land\\
\quad (\forall E\ u\ E_1\ E'.\ \mathit{TRANS}'\ E\ u\ E_1 \implies \mathit{TRANS}'\ (E \parallel E')\ u\ (E_1 \parallel E')) \land\\
\quad (\forall E\ u\ E_1\ E'.\ \mathit{TRANS}'\ E\ u\ E_1 \implies \mathit{TRANS}'\ (E' \parallel E)\ u\ (E' \parallel E_1)) \land\\
\quad (\forall E\ l\ E_1\ E'\ E_2.\\
\qquad \mathit{TRANS}'\ E\ (\texttt{label}\ l)\ E_1 \land \mathit{TRANS}'\ E'\ (\texttt{label}\ (\texttt{COMPL}\ l))\ E_2 \implies\\
\qquad \mathit{TRANS}'\ (E \parallel E')\ \tau\ (E_1 \parallel E_2)) \land\\
\quad (\forall E\ u\ E'\ l\ L.\\
\qquad \mathit{TRANS}'\ E\ u\ E' \land\\
\qquad ((u = \tau) \lor (u = \texttt{label}\ l) \land l \notin L \land \texttt{COMPL}\ l \notin L) \implies\\
\qquad \mathit{TRANS}'\ (\nu\ L\ E)\ u\ (\nu\ L\ E')) \land\\
\quad (\forall E\ u\ E'\ rf.\\
\qquad \mathit{TRANS}'\ E\ u\ E' \implies
\end{array}$$

$\quad\quad \mathit{TRANS}'\ (\texttt{relab}\ E\ \mathit{rf})\ (\texttt{relabel}\ \mathit{rf}\ u)\ (\texttt{relab}\ E'\ \mathit{rf}))\ \land$
$\quad (\forall E\ u\ X\ E_1.$
$\quad\quad \mathit{TRANS}'\ (\texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X)\ u\ E_1 \Longrightarrow$
$\quad\quad \mathit{TRANS}'\ (\texttt{rec}\ X\ E)\ u\ E_1) \Longrightarrow$
$\quad \forall a_0\ a_1\ a_2.\ a_0\ -a_1\rightarrow\ a_2 \Longrightarrow \mathit{TRANS}'\ a_0\ a_1\ a_2$

[TRANS_P_RESTR]

$\vdash \forall E\ u\ E'\ L.\ \nu\ L\ E\ -u\rightarrow\ \nu\ L\ E' \Longrightarrow E\ -u\rightarrow\ E'$

[TRANS_P_SUM_P]

$\vdash \forall E\ u\ E'.\ E\ \texttt{+}\ E\ -u\rightarrow\ E' \Longrightarrow E\ -u\rightarrow\ E'$

[TRANS_P_SUM_P_EQ]

$\vdash \forall E\ u\ E'.\ E\ \texttt{+}\ E\ -u\rightarrow\ E' \iff E\ -u\rightarrow\ E'$

[TRANS_PAR]

$\vdash \forall E\ E'\ u\ E''.$
$\quad E \parallel E'\ -u\rightarrow\ E'' \Longrightarrow$
$\quad (\exists E_1.\ (E'' = E_1 \parallel E') \land E\ -u\rightarrow\ E_1) \lor$
$\quad (\exists E_1.\ (E'' = E \parallel E_1) \land E'\ -u\rightarrow\ E_1) \lor$
$\quad \exists E_1\ E_2\ l.$
$\quad\quad (u = \tau) \land (E'' = E_1 \parallel E_2) \land E\ -\texttt{label}\ l\rightarrow\ E_1\ \land$
$\quad\quad E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\rightarrow\ E_2$

[TRANS_PAR_EQ]

$\vdash \forall E\ E'\ u\ E''.$
$\quad E \parallel E'\ -u\rightarrow\ E'' \iff$
$\quad (\exists E_1.\ (E'' = E_1 \parallel E') \land E\ -u\rightarrow\ E_1) \lor$
$\quad (\exists E_1.\ (E'' = E \parallel E_1) \land E'\ -u\rightarrow\ E_1) \lor$
$\quad \exists E_1\ E_2\ l.$
$\quad\quad (u = \tau) \land (E'' = E_1 \parallel E_2) \land E\ -\texttt{label}\ l\rightarrow\ E_1\ \land$
$\quad\quad E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\rightarrow\ E_2$

[TRANS_PAR_NO_SYNCR]

$\vdash \forall l\ l'.$
$\quad l \neq \texttt{COMPL}\ l' \Longrightarrow$
$\quad \forall E\ E'\ E''.\ \neg(\texttt{label}\ l..E \parallel \texttt{label}\ l'..E'\ -\tau\rightarrow\ E'')$

[TRANS_PAR_P_NIL]

$\vdash \forall E\ u\ E'.\ E \parallel \texttt{nil}\ -u\rightarrow\ E' \Longrightarrow \exists E''.\ E\ -u\rightarrow\ E'' \land (E' = E'' \parallel \texttt{nil})$

[TRANS_PREFIX]

$\vdash \forall u\ E\ u'\ E'.\ u..E\ -u'\rightarrow\ E' \Longrightarrow (u' = u) \land (E' = E)$

[TRANS_PREFIX_EQ]

$\vdash \forall u\ E\ u'\ E'.\ u..E\ -u'\rightarrow\ E' \iff (u' = u) \land (E' = E)$

[TRANS_REC]

$$\vdash \forall X\ E\ u\ E'.\ \texttt{rec}\ X\ E\ {-u\rightarrow}\ E' \Longrightarrow \texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X\ {-u\rightarrow}\ E'$$

[TRANS_REC_EQ]

$$\vdash \forall X\ E\ u\ E'.\ \texttt{rec}\ X\ E\ {-u\rightarrow}\ E' \iff \texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X\ {-u\rightarrow}\ E'$$

[TRANS_RELAB]

$$\begin{array}{l}\vdash \forall E\ \mathit{rf}\ u\ E'.\\ \quad\quad \texttt{relab}\ E\ \mathit{rf}\ {-u\rightarrow}\ E' \Longrightarrow\\ \quad\quad \exists u'\ E''.\\ \quad\quad\quad (u = \texttt{relabel}\ \mathit{rf}\ u') \land (E' = \texttt{relab}\ E''\ \mathit{rf}) \land E\ {-u'\rightarrow}\ E''\end{array}$$

[TRANS_RELAB_EQ]

$$\begin{array}{l}\vdash \forall E\ \mathit{rf}\ u\ E'.\\ \quad\quad \texttt{relab}\ E\ \mathit{rf}\ {-u\rightarrow}\ E' \iff\\ \quad\quad \exists u'\ E''.\\ \quad\quad\quad (u = \texttt{relabel}\ \mathit{rf}\ u') \land (E' = \texttt{relab}\ E''\ \mathit{rf}) \land E\ {-u'\rightarrow}\ E''\end{array}$$

[TRANS_RELAB_labl]

$$\begin{array}{l}\vdash \forall E\ \mathit{labl}\ u\ E'.\\ \quad\quad \texttt{relab}\ E\ (\texttt{RELAB}\ \mathit{labl})\ {-u\rightarrow}\ E' \Longrightarrow\\ \quad\quad \exists u'\ E''.\\ \quad\quad\quad (u = \texttt{relabel}\ (\texttt{RELAB}\ \mathit{labl})\ u') \land\\ \quad\quad\quad (E' = \texttt{relab}\ E''\ (\texttt{RELAB}\ \mathit{labl})) \land E\ {-u'\rightarrow}\ E''\end{array}$$

[TRANS_RESTR]

$$\begin{array}{l}\vdash \forall E\ L\ u\ E'.\\ \quad\quad \nu\ L\ E\ {-u\rightarrow}\ E' \Longrightarrow\\ \quad\quad \exists E''\ l.\\ \quad\quad\quad (E' = \nu\ L\ E'') \land E\ {-u\rightarrow}\ E'' \land\\ \quad\quad\quad ((u = \tau) \lor (u = \texttt{label}\ l) \land l \notin L \land \texttt{COMPL}\ l \notin L)\end{array}$$

[TRANS_RESTR_EQ]

$$\begin{array}{l}\vdash \forall E\ L\ u\ E'.\\ \quad\quad \nu\ L\ E\ {-u\rightarrow}\ E' \iff\\ \quad\quad \exists E''\ l.\\ \quad\quad\quad (E' = \nu\ L\ E'') \land E\ {-u\rightarrow}\ E'' \land\\ \quad\quad\quad ((u = \tau) \lor (u = \texttt{label}\ l) \land l \notin L \land \texttt{COMPL}\ l \notin L)\end{array}$$

[TRANS_RESTR_NO_NIL]

$$\vdash \forall E\ L\ u\ E'.\ \nu\ L\ E\ {-u\rightarrow}\ \nu\ L\ E' \Longrightarrow E \neq \texttt{nil}$$

[TRANS_rules]

$$\begin{array}{l}
\vdash (\forall E\ u.\ u\texttt{..}E\ -u\!\rightarrow\ E)\ \land\ (\forall E\ u\ E_1\ E'.\ E\ -u\!\rightarrow\ E_1 \implies E\ \texttt{+}\ E'\ -u\!\rightarrow\ E_1)\ \land \\
\quad (\forall E\ u\ E_1\ E'.\ E\ -u\!\rightarrow\ E_1 \implies E'\ \texttt{+}\ E\ -u\!\rightarrow\ E_1)\ \land \\
\quad (\forall E\ u\ E_1\ E'.\ E\ -u\!\rightarrow\ E_1 \implies E \parallel E'\ -u\!\rightarrow\ E_1 \parallel E')\ \land \\
\quad (\forall E\ u\ E_1\ E'.\ E\ -u\!\rightarrow\ E_1 \implies E' \parallel E\ -u\!\rightarrow\ E' \parallel E_1)\ \land \\
\quad (\forall E\ l\ E_1\ E'\ E_2. \\
\qquad E\ -\texttt{label}\ l\!\rightarrow\ E_1\ \land\ E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\!\rightarrow\ E_2 \implies \\
\qquad E \parallel E'\ -\tau\!\rightarrow\ E_1 \parallel E_2)\ \land \\
\quad (\forall E\ u\ E'\ l\ L. \\
\qquad E\ -u\!\rightarrow\ E'\ \land \\
\qquad ((u\ \texttt{=}\ \tau)\ \lor\ (u\ \texttt{=}\ \texttt{label}\ l)\ \land\ l \notin L\ \land\ \texttt{COMPL}\ l \notin L) \implies \\
\qquad \nu\ L\ E\ -u\!\rightarrow\ \nu\ L\ E')\ \land \\
\quad (\forall E\ u\ E'\ \mathit{rf}. \\
\qquad E\ -u\!\rightarrow\ E' \implies \texttt{relab}\ E\ \mathit{rf}\ -\texttt{relabel}\ \mathit{rf}\ u\!\rightarrow\ \texttt{relab}\ E'\ \mathit{rf})\ \land \\
\quad \forall E\ u\ X\ E_1.\ \texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X\ -u\!\rightarrow\ E_1 \implies \texttt{rec}\ X\ E\ -u\!\rightarrow\ E_1
\end{array}$$

[TRANS_strongind]

$$\begin{array}{l}
\vdash \forall\, \mathit{TRANS}'. \\
\qquad (\forall E\ u.\ \mathit{TRANS}'\ (u\texttt{..}E)\ u\ E)\ \land \\
\qquad (\forall E\ u\ E_1\ E'. \\
\qquad\quad E\ -u\!\rightarrow\ E_1\ \land\ \mathit{TRANS}'\ E\ u\ E_1 \implies \mathit{TRANS}'\ (E\ \texttt{+}\ E')\ u\ E_1)\ \land \\
\qquad (\forall E\ u\ E_1\ E'. \\
\qquad\quad E\ -u\!\rightarrow\ E_1\ \land\ \mathit{TRANS}'\ E\ u\ E_1 \implies \mathit{TRANS}'\ (E'\ \texttt{+}\ E)\ u\ E_1)\ \land \\
\qquad (\forall E\ u\ E_1\ E'. \\
\qquad\quad E\ -u\!\rightarrow\ E_1\ \land\ \mathit{TRANS}'\ E\ u\ E_1 \implies \\
\qquad\quad \mathit{TRANS}'\ (E \parallel E')\ u\ (E_1 \parallel E'))\ \land \\
\qquad (\forall E\ u\ E_1\ E'. \\
\qquad\quad E\ -u\!\rightarrow\ E_1\ \land\ \mathit{TRANS}'\ E\ u\ E_1 \implies \\
\qquad\quad \mathit{TRANS}'\ (E' \parallel E)\ u\ (E' \parallel E_1))\ \land \\
\qquad (\forall E\ l\ E_1\ E'\ E_2. \\
\qquad\quad E\ -\texttt{label}\ l\!\rightarrow\ E_1\ \land\ \mathit{TRANS}'\ E\ (\texttt{label}\ l)\ E_1\ \land \\
\qquad\quad E'\ -\texttt{label}\ (\texttt{COMPL}\ l)\!\rightarrow\ E_2\ \land \\
\qquad\quad \mathit{TRANS}'\ E'\ (\texttt{label}\ (\texttt{COMPL}\ l))\ E_2 \implies \\
\qquad\quad \mathit{TRANS}'\ (E \parallel E')\ \tau\ (E_1 \parallel E_2))\ \land \\
\qquad (\forall E\ u\ E'\ l\ L. \\
\qquad\quad E\ -u\!\rightarrow\ E'\ \land\ \mathit{TRANS}'\ E\ u\ E'\ \land \\
\qquad\quad ((u\ \texttt{=}\ \tau)\ \lor\ (u\ \texttt{=}\ \texttt{label}\ l)\ \land\ l \notin L\ \land\ \texttt{COMPL}\ l \notin L) \implies \\
\qquad\quad \mathit{TRANS}'\ (\nu\ L\ E)\ u\ (\nu\ L\ E'))\ \land \\
\qquad (\forall E\ u\ E'\ \mathit{rf}. \\
\qquad\quad E\ -u\!\rightarrow\ E'\ \land\ \mathit{TRANS}'\ E\ u\ E' \implies \\
\qquad\quad \mathit{TRANS}'\ (\texttt{relab}\ E\ \mathit{rf})\ (\texttt{relabel}\ \mathit{rf}\ u)\ (\texttt{relab}\ E'\ \mathit{rf}))\ \land \\
\qquad (\forall E\ u\ X\ E_1. \\
\qquad\quad \texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X\ -u\!\rightarrow\ E_1\ \land \\
\qquad\quad \mathit{TRANS}'\ (\texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X)\ u\ E_1 \implies \\
\qquad\quad \mathit{TRANS}'\ (\texttt{rec}\ X\ E)\ u\ E_1) \implies \\
\qquad \forall a_0\ a_1\ a_2.\ a_0\ -a_1\!\rightarrow\ a_2 \implies \mathit{TRANS}'\ a_0\ a_1\ a_2
\end{array}$$

[TRANS_SUM]

$\vdash \forall E\ E'\ u\ E''.\ E\ \texttt{+}\ E'\ -u\rightarrow\ E'' \Longrightarrow E\ -u\rightarrow\ E'' \lor E'\ -u\rightarrow\ E''$

[TRANS_SUM_EQ]

$\vdash \forall E\ E'\ u\ E''.\ E\ \texttt{+}\ E'\ -u\rightarrow\ E'' \iff E\ -u\rightarrow\ E'' \lor E'\ -u\rightarrow\ E''$

[TRANS_SUM_EQ']

$\vdash \forall E_1\ E_2\ u\ E.\ E_1\ \texttt{+}\ E_2\ -u\rightarrow\ E \iff E_1\ -u\rightarrow\ E \lor E_2\ -u\rightarrow\ E$

[TRANS_SUM_NIL]

$\vdash \forall E\ u\ E'.\ E\ \texttt{+}\ \texttt{nil}\ -u\rightarrow\ E' \Longrightarrow E\ -u\rightarrow\ E'$

[TRANS_SUM_NIL_EQ]

$\vdash \forall E\ u\ E'.\ E\ \texttt{+}\ \texttt{nil}\ -u\rightarrow\ E' \iff E\ -u\rightarrow\ E'$

[VAR_NO_TRANS]

$\vdash \forall X\ u\ E.\ \neg(\texttt{var}\ X\ -u\rightarrow\ E)$

9.2 StrongEQ Theory

Built: 08 Dicembre 2017
Parent Theories: CCS

9.2.1 Definitions

[STRONG_BISIM_def]

$\vdash \forall R.\ \texttt{STRONG\_BISIM}\ R \iff \texttt{STRONG\_SIM}\ R \land \texttt{STRONG\_SIM}\ (\texttt{relinv}\ R)$

[STRONG_EQUIV_def]

$$\begin{array}{l}
\vdash \texttt{STRONG\_EQUIV} = \\
\quad (\lambda\, a_0\ a_1. \\
\qquad \exists\, \mathit{STRONG\_EQUIV}'. \\
\qquad\quad \mathit{STRONG\_EQUIV}'\ a_0\ a_1 \land \\
\qquad\quad \forall\, a_0\ a_1. \\
\qquad\qquad \mathit{STRONG\_EQUIV}'\ a_0\ a_1 \Longrightarrow \\
\qquad\qquad \forall\, u. \\
\qquad\qquad\quad (\forall\, E_1. \\
\qquad\qquad\qquad a_0\ -u\rightarrow\ E_1 \Longrightarrow \\
\qquad\qquad\qquad \exists\, E_2.\ a_1\ -u\rightarrow\ E_2 \land \mathit{STRONG\_EQUIV}'\ E_1\ E_2) \land \\
\qquad\qquad\quad \forall\, E_2. \\
\qquad\qquad\qquad a_1\ -u\rightarrow\ E_2 \Longrightarrow \exists\, E_1.\ a_0\ -u\rightarrow\ E_1 \land \mathit{STRONG\_EQUIV}'\ E_1\ E_2)
\end{array}$$

[STRONG_SIM_def]

$$\begin{array}{l}
\vdash \forall R. \\
\qquad \texttt{STRONG\_SIM}\ R \iff \\
\qquad \forall E\ E'.\ R\ E\ E' \Longrightarrow \forall u\ E_1.\ E\ -u\rightarrow\ E_1 \Longrightarrow \exists E_2.\ E'\ -u\rightarrow\ E_2 \land R\ E_1\ E_2
\end{array}$$

9.2.2 Theorems

[COMP_STRONG_BISIM]

$$\begin{array}{l}
\vdash \forall Bsm_1\ Bsm_2.\\
\qquad \texttt{STRONG\_BISIM}\ Bsm_1 \land \texttt{STRONG\_BISIM}\ Bsm_2 \implies\\
\qquad \texttt{STRONG\_BISIM}\ (Bsm_2\ \texttt{O}\ Bsm_1)
\end{array}$$

[CONVERSE_STRONG_BISIM]

$$\vdash \forall Bsm.\ \texttt{STRONG\_BISIM}\ Bsm \implies \texttt{STRONG\_BISIM}\ (\texttt{relinv}\ Bsm)$$

[EQUAL_IMP_STRONG_EQUIV]

$$\vdash \forall E\ E'.\ (E = E') \implies \texttt{STRONG\_EQUIV}\ E\ E'$$

[IDENTITY_STRONG_BISIM]

$$\vdash \texttt{STRONG\_BISIM}\ (=)$$

[PROPERTY_STAR]

$$\begin{array}{l}
\vdash \forall a_0\ a_1.\\
\qquad \texttt{STRONG\_EQUIV}\ a_0\ a_1 \iff\\
\qquad \forall u.\\
\qquad\quad (\forall E_1.\ a_0\ -u\rightarrow\ E_1 \implies \exists E_2.\ a_1\ -u\rightarrow\ E_2 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2) \land\\
\qquad\quad \forall E_2.\ a_1\ -u\rightarrow\ E_2 \implies \exists E_1.\ a_0\ -u\rightarrow\ E_1 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2
\end{array}$$

[PROPERTY_STAR_LEFT]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\qquad \texttt{STRONG\_EQUIV}\ E\ E' \implies\\
\qquad \forall u\ E_1.\ E\ -u\rightarrow\ E_1 \implies \exists E_2.\ E'\ -u\rightarrow\ E_2 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2
\end{array}$$

[PROPERTY_STAR_RIGHT]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\qquad \texttt{STRONG\_EQUIV}\ E\ E' \implies\\
\qquad \forall u\ E_2.\ E'\ -u\rightarrow\ E_2 \implies \exists E_1.\ E\ -u\rightarrow\ E_1 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2
\end{array}$$

[STRONG_BISIM]

$$\begin{array}{l}
\vdash \texttt{STRONG\_BISIM}\ Bsm \iff\\
\quad \forall E\ E'.\\
\qquad Bsm\ E\ E' \implies\\
\qquad \forall u.\\
\qquad\quad (\forall E_1.\ E\ -u\rightarrow\ E_1 \implies \exists E_2.\ E'\ -u\rightarrow\ E_2 \land Bsm\ E_1\ E_2) \land\\
\qquad\quad \forall E_2.\ E'\ -u\rightarrow\ E_2 \implies \exists E_1.\ E\ -u\rightarrow\ E_1 \land Bsm\ E_1\ E_2
\end{array}$$

[STRONG_BISIM_SUBSET_STRONG_EQUIV]

$$\vdash \forall Bsm.\ \texttt{STRONG\_BISIM}\ Bsm \implies Bsm\ \texttt{RSUBSET}\ \texttt{STRONG\_EQUIV}$$

[STRONG_EQUIV]

$$\vdash \forall E\ E'.\ \texttt{STRONG\_EQUIV}\ E\ E' \iff \exists Bsm.\ Bsm\ E\ E' \land \texttt{STRONG\_BISIM}\ Bsm$$

[STRONG_EQUIV_cases]

$$\begin{array}{l}
\vdash \forall a_0\ a_1.\\
\quad \texttt{STRONG\_EQUIV}\ a_0\ a_1 \iff\\
\quad \forall u.\\
\qquad (\forall E_1.\ a_0\ -u\rightarrow\ E_1 \implies \exists E_2.\ a_1\ -u\rightarrow\ E_2 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2) \land\\
\qquad \forall E_2.\ a_1\ -u\rightarrow\ E_2 \implies \exists E_1.\ a_0\ -u\rightarrow\ E_1 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2
\end{array}$$

[STRONG_EQUIV_coind]

$$\begin{array}{l}
\vdash \forall \mathit{STRONG\_EQUIV}'.\\
\quad (\forall a_0\ a_1.\\
\qquad \mathit{STRONG\_EQUIV}'\ a_0\ a_1 \implies\\
\qquad \forall u.\\
\qquad\quad (\forall E_1.\\
\qquad\qquad a_0\ -u\rightarrow\ E_1 \implies \exists E_2.\ a_1\ -u\rightarrow\ E_2 \land \mathit{STRONG\_EQUIV}'\ E_1\ E_2) \land\\
\qquad\quad \forall E_2.\\
\qquad\qquad a_1\ -u\rightarrow\ E_2 \implies \exists E_1.\ a_0\ -u\rightarrow\ E_1 \land \mathit{STRONG\_EQUIV}'\ E_1\ E_2) \implies\\
\quad \forall a_0\ a_1.\ \mathit{STRONG\_EQUIV}'\ a_0\ a_1 \implies \texttt{STRONG\_EQUIV}\ a_0\ a_1
\end{array}$$

[STRONG_EQUIV_equivalence]

$$\vdash \texttt{equivalence STRONG\_EQUIV}$$

[STRONG_EQUIV_IS_STRONG_BISIM]

$$\vdash \texttt{STRONG\_BISIM STRONG\_EQUIV}$$

[STRONG_EQUIV_PRESD_BY_PAR]

$$\begin{array}{l}
\vdash \forall E_1\ E_1'\ E_2\ E_2'.\\
\quad \texttt{STRONG\_EQUIV}\ E_1\ E_1' \land \texttt{STRONG\_EQUIV}\ E_2\ E_2' \implies\\
\quad \texttt{STRONG\_EQUIV}\ (E_1 \parallel E_2)\ (E_1' \parallel E_2')
\end{array}$$

[STRONG_EQUIV_PRESD_BY_SUM]

$$\begin{array}{l}
\vdash \forall E_1\ E_1'\ E_2\ E_2'.\\
\quad \texttt{STRONG\_EQUIV}\ E_1\ E_1' \land \texttt{STRONG\_EQUIV}\ E_2\ E_2' \implies\\
\quad \texttt{STRONG\_EQUIV}\ (E_1\ \texttt{+}\ E_2)\ (E_1'\ \texttt{+}\ E_2')
\end{array}$$

[STRONG_EQUIV_REFL]

$$\vdash \forall E.\ \texttt{STRONG\_EQUIV}\ E\ E$$

[STRONG_EQUIV_rules]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad (\forall u.\\
\qquad (\forall E_1.\ E\ -u\rightarrow\ E_1 \implies \exists E_2.\ E'\ -u\rightarrow\ E_2 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2) \land\\
\qquad \forall E_2.\ E'\ -u\rightarrow\ E_2 \implies \exists E_1.\ E\ -u\rightarrow\ E_1 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2) \implies\\
\quad \texttt{STRONG\_EQUIV}\ E\ E'
\end{array}$$

[STRONG_EQUIV_SUBST_PAR_L]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{STRONG\_EQUIV}\ E\ E' \Longrightarrow \forall E''.\ \texttt{STRONG\_EQUIV}\ (E'' \parallel E)\ (E'' \parallel E')\end{array}$$

[STRONG_EQUIV_SUBST_PAR_R]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{STRONG\_EQUIV}\ E\ E' \Longrightarrow \forall E''.\ \texttt{STRONG\_EQUIV}\ (E \parallel E'')\ (E' \parallel E'')\end{array}$$

[STRONG_EQUIV_SUBST_PREFIX]

$$\vdash \forall E\ E'.\ \texttt{STRONG\_EQUIV}\ E\ E' \Longrightarrow \forall u.\ \texttt{STRONG\_EQUIV}\ (u..E)\ (u..E')$$

[STRONG_EQUIV_SUBST_RELAB]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{STRONG\_EQUIV}\ E\ E' \Longrightarrow\\ \quad \forall rf.\ \texttt{STRONG\_EQUIV}\ (\texttt{relab}\ E\ rf)\ (\texttt{relab}\ E'\ rf)\end{array}$$

[STRONG_EQUIV_SUBST_RESTR]

$$\vdash \forall E\ E'.\ \texttt{STRONG\_EQUIV}\ E\ E' \Longrightarrow \forall L.\ \texttt{STRONG\_EQUIV}\ (\nu\ L\ E)\ (\nu\ L\ E')$$

[STRONG_EQUIV_SUBST_SUM_L]

$$\begin{array}{l}\vdash \forall E'\ E.\\ \quad \texttt{STRONG\_EQUIV}\ E\ E' \Longrightarrow \forall E''.\ \texttt{STRONG\_EQUIV}\ (E''\ \texttt{+}\ E)\ (E''\ \texttt{+}\ E')\end{array}$$

[STRONG_EQUIV_SUBST_SUM_R]

$$\begin{array}{l}\vdash \forall E'\ E.\\ \quad \texttt{STRONG\_EQUIV}\ E\ E' \Longrightarrow \forall E''.\ \texttt{STRONG\_EQUIV}\ (E\ \texttt{+}\ E'')\ (E'\ \texttt{+}\ E'')\end{array}$$

[STRONG_EQUIV_SYM]

$$\vdash \forall E\ E'.\ \texttt{STRONG\_EQUIV}\ E\ E' \Longrightarrow \texttt{STRONG\_EQUIV}\ E'\ E$$

[STRONG_EQUIV_TRANS]

$$\begin{array}{l}\vdash \forall E\ E'\ E''.\\ \quad \texttt{STRONG\_EQUIV}\ E\ E' \land \texttt{STRONG\_EQUIV}\ E'\ E'' \Longrightarrow\\ \quad \texttt{STRONG\_EQUIV}\ E\ E''\end{array}$$

[UNION_STRONG_BISIM]

$$\begin{array}{l}\vdash \forall Bsm_1\ Bsm_2.\\ \quad \texttt{STRONG\_BISIM}\ Bsm_1 \land \texttt{STRONG\_BISIM}\ Bsm_2 \Longrightarrow\\ \quad \texttt{STRONG\_BISIM}\ (Bsm_1\ \texttt{RUNION}\ Bsm_2)\end{array}$$

9.3 StrongLaws Theory

Built: 08 Dicembre 2017
Parent Theories: StrongEQ

9.3.1 Definitions

[ALL_SYNC_def]

```
⊢ (∀f f′ m.
      ALL_SYNC f 0 f′ m =
      SYNC (PREF_ACT (f 0)) (PREF_PROC (f 0)) f′ m) ∧
   ∀f n f′ m.
     ALL_SYNC f (SUC n) f′ m =
     ALL_SYNC f n f′ m +
     SYNC (PREF_ACT (f (SUC n))) (PREF_PROC (f (SUC n))) f′ m
```

[CCS_COMP_def]

```
⊢ (∀f. PI f 0 = f 0) ∧ ∀f n. PI f (SUC n) = PI f n ‖ f (SUC n)
```

[CCS_SIGMA_def]

```
⊢ (∀f. SIGMA f 0 = f 0) ∧
   ∀f n. SIGMA f (SUC n) = SIGMA f n + f (SUC n)
```

[Is_Prefix_def]

```
⊢ ∀E. Is_Prefix E ⇔ ∃u E′. E = u..E′
```

[PREF_ACT_def]

```
⊢ ∀u E. PREF_ACT (u..E) = u
```

[PREF_PROC_def]

```
⊢ ∀u E. PREF_PROC (u..E) = E
```

[SYNC_def]

```
⊢ (∀u P f.
      SYNC u P f 0 =
      if (u = τ) ∨ (PREF_ACT (f 0) = τ) then nil
      else if LABEL u = COMPL (LABEL (PREF_ACT (f 0))) then
        τ..(P ‖ PREF_PROC (f 0))
      else nil) ∧
   ∀u P f n.
     SYNC u P f (SUC n) =
     if (u = τ) ∨ (PREF_ACT (f (SUC n)) = τ) then SYNC u P f n
     else if LABEL u = COMPL (LABEL (PREF_ACT (f (SUC n)))) then
       τ..(P ‖ PREF_PROC (f (SUC n))) + SYNC u P f n
     else SYNC u P f n
```

9.3.2 Theorems

[ALL_SYNC_BASE]

```
⊢ ∀f f′ m.
     ALL_SYNC f 0 f′ m =
     SYNC (PREF_ACT (f 0)) (PREF_PROC (f 0)) f′ m
```

[ALL_SYNC_def_compute]

```
⊢ (∀f f′ m.
     ALL_SYNC f 0 f′ m =
     SYNC (PREF_ACT (f 0)) (PREF_PROC (f 0)) f′ m) ∧
  (∀f n f′ m.
     ALL_SYNC f (NUMERAL (BIT1 n)) f′ m =
     ALL_SYNC f (NUMERAL (BIT1 n) - 1) f′ m +
     SYNC (PREF_ACT (f (NUMERAL (BIT1 n))))
       (PREF_PROC (f (NUMERAL (BIT1 n)))) f′ m) ∧
  ∀f n f′ m.
    ALL_SYNC f (NUMERAL (BIT2 n)) f′ m =
    ALL_SYNC f (NUMERAL (BIT1 n)) f′ m +
    SYNC (PREF_ACT (f (NUMERAL (BIT2 n))))
      (PREF_PROC (f (NUMERAL (BIT2 n)))) f′ m
```

[ALL_SYNC_INDUCT]

```
⊢ ∀f n f′ m.
    ALL_SYNC f (SUC n) f′ m =
    ALL_SYNC f n f′ m +
    SYNC (PREF_ACT (f (SUC n))) (PREF_PROC (f (SUC n))) f′ m
```

[ALL_SYNC_TRANS_THM]

```
⊢ ∀n m f f′ u E.
    ALL_SYNC f n f′ m −u→ E ⟹
    ∃k k′ l.
      k ≤ n ∧ k′ ≤ m ∧ (PREF_ACT (f k) = label l) ∧
      (PREF_ACT (f′ k′) = label (COMPL l)) ∧ (u = τ) ∧
      (E = PREF_PROC (f k) ∥ PREF_PROC (f′ k′))
```

[ALL_SYNC_TRANS_THM_EQ]

```
⊢ ∀n m f f′ u E.
    ALL_SYNC f n f′ m −u→ E ⇔
    ∃k k′ l.
      k ≤ n ∧ k′ ≤ m ∧ (PREF_ACT (f k) = label l) ∧
      (PREF_ACT (f′ k′) = label (COMPL l)) ∧ (u = τ) ∧
      (E = PREF_PROC (f k) ∥ PREF_PROC (f′ k′))
```

[CCS_COMP_def_compute]

```
⊢ (∀f. PI f 0 = f 0) ∧
  (∀f n.
     PI f (NUMERAL (BIT1 n)) =
     PI f (NUMERAL (BIT1 n) - 1) ∥ f (NUMERAL (BIT1 n))) ∧
  ∀f n.
    PI f (NUMERAL (BIT2 n)) =
    PI f (NUMERAL (BIT1 n)) ∥ f (NUMERAL (BIT2 n))
```

[CCS_SIGMA_def_compute]

```
⊢ (∀f. SIGMA f 0 = f 0) ∧
  (∀f n.
     SIGMA f (NUMERAL (BIT1 n)) =
     SIGMA f (NUMERAL (BIT1 n) - 1) + f (NUMERAL (BIT1 n))) ∧
  ∀f n.
    SIGMA f (NUMERAL (BIT2 n)) =
    SIGMA f (NUMERAL (BIT1 n)) + f (NUMERAL (BIT2 n))
```

[COMP_BASE]

```
⊢ ∀f. PI f 0 = f 0
```

[COMP_INDUCT]

```
⊢ ∀f n. PI f (SUC n) = PI f n ∥ f (SUC n)
```

[PREF_IS_PREFIX]

```
⊢ ∀u E. Is_Prefix (u..E)
```

[SIGMA_BASE]

```
⊢ ∀f. SIGMA f 0 = f 0
```

[SIGMA_INDUCT]

```
⊢ ∀f n. SIGMA f (SUC n) = SIGMA f n + f (SUC n)
```

[SIGMA_TRANS_THM]

```
⊢ ∀n f u E. SIGMA f n −u→ E ⟹ ∃k. k ≤ n ∧ f k −u→ E
```

[SIGMA_TRANS_THM_EQ]

```
⊢ ∀n f u E. SIGMA f n −u→ E ⟺ ∃k. k ≤ n ∧ f k −u→ E
```

[STRONG_EXPANSION_LAW]

```
⊢ ∀f n f′ m.
    (∀i. i ≤ n ⟹ Is_Prefix (f i)) ∧
    (∀j. j ≤ m ⟹ Is_Prefix (f′ j)) ⟹
    STRONG_EQUIV (SIGMA f n ∥ SIGMA f′ m)
      (SIGMA
         (λi. PREF_ACT (f i)..(PREF_PROC (f i) ∥ SIGMA f′ m))
         n +
       SIGMA
         (λj. PREF_ACT (f′ j)..(SIGMA f n ∥ PREF_PROC (f′ j)))
         m + ALL_SYNC f n f′ m)
```

[STRONG_LEFT_SUM_MID_IDEMP]

```
⊢ ∀E E′ E″. STRONG_EQUIV (E + E′ + E″ + E′) (E + E″ + E′)
```

[STRONG_PAR_ASSOC]

```
⊢ ∀E E′ E″. STRONG_EQUIV (E ∥ E′ ∥ E″) (E ∥ (E′ ∥ E″))
```

[STRONG_PAR_COMM]

```
⊢ ∀E E′. STRONG_EQUIV (E ∥ E′) (E′ ∥ E)
```

[STRONG_PAR_IDENT_L]

```
⊢ ∀E. STRONG_EQUIV (nil ∥ E) E
```

[STRONG_PAR_IDENT_R]

```
⊢ ∀E. STRONG_EQUIV (E ∥ nil) E
```

[STRONG_PAR_PREF_NO_SYNCR]

```
⊢ ∀l l′.
     l ≠ COMPL l′ ⟹
     ∀E E′.
       STRONG_EQUIV (label l..E ∥ label l′..E′)
         (label l..(E ∥ label l′..E′) +
          label l′..(label l..E ∥ E′))
```

[STRONG_PAR_PREF_SYNCR]

```
⊢ ∀l l′.
     (l = COMPL l′) ⟹
     ∀E E′.
       STRONG_EQUIV (label l..E ∥ label l′..E′)
         (label l..(E ∥ label l′..E′) +
          label l′..(label l..E ∥ E′) + τ..(E ∥ E′))
```

[STRONG_PAR_PREF_TAU]

```
⊢ ∀u E E′.
     STRONG_EQUIV (u..E ∥ τ..E′)
       (u..(E ∥ τ..E′) + τ..(u..E ∥ E′))
```

[STRONG_PAR_TAU_PREF]

```
⊢ ∀E u E′.
     STRONG_EQUIV (τ..E ∥ u..E′)
       (τ..(E ∥ u..E′) + u..(τ..E ∥ E′))
```

[STRONG_PAR_TAU_TAU]

```
⊢ ∀E E′.
     STRONG_EQUIV (τ..E ∥ τ..E′)
       (τ..(E ∥ τ..E′) + τ..(τ..E ∥ E′))
```

[STRONG_PREF_REC_EQUIV]

```
⊢ ∀u s v.
     STRONG_EQUIV (u..rec s (v..u..var s)) (rec s (u..v..var s))
```

[STRONG_REC_ACT2]

```
⊢ ∀s u. STRONG_EQUIV (rec s (u..u..var s)) (rec s (u..var s))
```

[STRONG_RELAB_NIL]

```
⊢ ∀rf. STRONG_EQUIV (relab nil rf) nil
```

[STRONG_RELAB_PREFIX]

```
⊢ ∀u E labl.
     STRONG_EQUIV (relab (u..E) (RELAB labl))
       (relabel (RELAB labl) u..relab E (RELAB labl))
```

[STRONG_RELAB_SUM]

```
⊢ ∀E E′ rf.
     STRONG_EQUIV (relab (E + E′) rf) (relab E rf + relab E′ rf)
```

[STRONG_RESTR_NIL]

```
⊢ ∀L. STRONG_EQUIV (ν L nil) nil
```

[STRONG_RESTR_PR_LAB_NIL]

```
⊢ ∀l L.
     l ∈ L ∨ COMPL l ∈ L ⟹
     ∀E. STRONG_EQUIV (ν L (label l..E)) nil
```

[STRONG_RESTR_PREFIX_LABEL]

```
⊢ ∀l L.
     l ∉ L ∧ COMPL l ∉ L ⟹
     ∀E. STRONG_EQUIV (ν L (label l..E)) (label l..ν L E)
```

[STRONG_RESTR_PREFIX_TAU]

```
⊢ ∀E L. STRONG_EQUIV (ν L (τ..E)) (τ..ν L E)
```

[STRONG_RESTR_SUM]

```
⊢ ∀E E′ L. STRONG_EQUIV (ν L (E + E′)) (ν L E + ν L E′)
```

[STRONG_SUM_ASSOC_L]

```
⊢ ∀E E′ E″. STRONG_EQUIV (E + (E′ + E″)) (E + E′ + E″)
```

[STRONG_SUM_ASSOC_R]

```
⊢ ∀E E′ E″. STRONG_EQUIV (E + E′ + E″) (E + (E′ + E″))
```

[STRONG_SUM_COMM]

```
⊢ ∀E E′. STRONG_EQUIV (E + E′) (E′ + E)
```

[STRONG_SUM_IDEMP]

```
⊢ ∀E. STRONG_EQUIV (E + E) E
```

[STRONG_SUM_IDENT_L]

```
⊢ ∀E. STRONG_EQUIV (nil + E) E
```

[STRONG_SUM_IDENT_R]

```
⊢ ∀E. STRONG_EQUIV (E + nil) E
```

[STRONG_SUM_MID_IDEMP]

```
⊢ ∀E E′. STRONG_EQUIV (E + E′ + E) (E′ + E)
```

[STRONG_UNFOLDING]

```
⊢ ∀X E. STRONG_EQUIV (rec X E) (CCS_Subst E (rec X E) X)
```

[SYNC_BASE]

```
⊢ ∀u P f.
     SYNC u P f 0 =
     if (u = τ) ∨ (PREF_ACT (f 0) = τ) then nil
     else if LABEL u = COMPL (LABEL (PREF_ACT (f 0))) then
       τ..(P ‖ PREF_PROC (f 0))
     else nil
```

[SYNC_def_compute]

```
⊢ (∀u P f.
      SYNC u P f 0 =
      if (u = τ) ∨ (PREF_ACT (f 0) = τ) then nil
      else if LABEL u = COMPL (LABEL (PREF_ACT (f 0))) then
        τ..(P ‖ PREF_PROC (f 0))
      else nil) ∧
  (∀u P f n.
      SYNC u P f (NUMERAL (BIT1 n)) =
      if (u = τ) ∨ (PREF_ACT (f (NUMERAL (BIT1 n))) = τ) then
        SYNC u P f (NUMERAL (BIT1 n) - 1)
      else if
        LABEL u =
        COMPL (LABEL (PREF_ACT (f (NUMERAL (BIT1 n)))))
      then
        τ..(P ‖ PREF_PROC (f (NUMERAL (BIT1 n)))) +
        SYNC u P f (NUMERAL (BIT1 n) - 1)
      else SYNC u P f (NUMERAL (BIT1 n) - 1)) ∧
  ∀u P f n.
    SYNC u P f (NUMERAL (BIT2 n)) =
    if (u = τ) ∨ (PREF_ACT (f (NUMERAL (BIT2 n))) = τ) then
      SYNC u P f (NUMERAL (BIT1 n))
    else if
```

```
      LABEL u = COMPL (LABEL (PREF_ACT (f (NUMERAL (BIT2 n)))))
    then
      τ..(P ‖ PREF_PROC (f (NUMERAL (BIT2 n)))) +
      SYNC u P f (NUMERAL (BIT1 n))
    else SYNC u P f (NUMERAL (BIT1 n))
```

[SYNC_INDUCT]

```
 ⊢ ∀u P f n.
     SYNC u P f (SUC n) =
     if (u = τ) ∨ (PREF_ACT (f (SUC n)) = τ) then SYNC u P f n
     else if LABEL u = COMPL (LABEL (PREF_ACT (f (SUC n)))) then
       τ..(P ‖ PREF_PROC (f (SUC n))) + SYNC u P f n
     else SYNC u P f n
```

[SYNC_TRANS_THM]

```
 ⊢ ∀m u P f v Q.
     SYNC u P f m −v→ Q ⟹
     ∃j l.
       j ≤ m ∧ (u = label l) ∧
       (PREF_ACT (f j) = label (COMPL l)) ∧ (v = τ) ∧
       (Q = P ‖ PREF_PROC (f j))
```

[SYNC_TRANS_THM_EQ]

```
 ⊢ ∀m u P f v Q.
     SYNC u P f m −v→ Q ⟺
     ∃j l.
       j ≤ m ∧ (u = label l) ∧
       (PREF_ACT (f j) = label (COMPL l)) ∧ (v = τ) ∧
       (Q = P ‖ PREF_PROC (f j))
```

9.4 WeakEQ Theory

Built: 08 Dicembre 2017
Parent Theories: StrongEQ

9.4.1 Definitions

[EPS_def]

```
 ⊢ EPS = (λE E′. E −τ→ E′)*
```

[STABLE]

```
 ⊢ ∀E. STABLE E ⟺ ∀u E′. E −u→ E′ ⟹ u ≠ τ
```

[WEAK_BISIM_def]

```
 ⊢ ∀R. WEAK_BISIM R ⟺ WEAK_SIM R ∧ WEAK_SIM (relinv R)
```

[WEAK_EQUIV_def]

```
⊢ WEAK_EQUIV =
    (λ a₀ a₁.
       ∃ WEAK_EQUIV′.
         WEAK_EQUIV′ a₀ a₁ ∧
         ∀ a₀ a₁.
           WEAK_EQUIV′ a₀ a₁ ⟹
           (∀ l.
              (∀ E₁.
                 a₀ −label l→ E₁ ⟹
                 ∃ E₂. a₁ =label l⇒ E₂ ∧ WEAK_EQUIV′ E₁ E₂) ∧
              ∀ E₂.
                a₁ −label l→ E₂ ⟹
                ∃ E₁. a₀ =label l⇒ E₁ ∧ WEAK_EQUIV′ E₁ E₂) ∧
           (∀ E₁.
              a₀ −τ→ E₁ ⟹ ∃ E₂. EPS a₁ E₂ ∧ WEAK_EQUIV′ E₁ E₂) ∧
           ∀ E₂. a₁ −τ→ E₂ ⟹ ∃ E₁. EPS a₀ E₁ ∧ WEAK_EQUIV′ E₁ E₂)
```

[WEAK_SIM_def]

```
⊢ ∀ R.
    WEAK_SIM R ⟺
    ∀ E E′.
      R E E′ ⟹
      (∀ l E₁.
         E −label l→ E₁ ⟹ ∃ E₂. E′ =label l⇒ E₂ ∧ R E₁ E₂) ∧
      ∀ E₁. E −τ→ E₁ ⟹ ∃ E₂. EPS E′ E₂ ∧ R E₁ E₂
```

[WEAK_TRANS]

```
⊢ ∀ E u E′. E =u⇒ E′ ⟺ ∃ E₁ E₂. EPS E E₁ ∧ E₁ −u→ E₂ ∧ EPS E₂ E′
```

9.4.2 Theorems

[COMP_WEAK_BISIM]

```
⊢ ∀ Wbsm₁ Wbsm₂.
    WEAK_BISIM Wbsm₁ ∧ WEAK_BISIM Wbsm₂ ⟹
    WEAK_BISIM (Wbsm₂ O Wbsm₁)
```

[CONVERSE_WEAK_BISIM]

```
⊢ ∀ Wbsm. WEAK_BISIM Wbsm ⟹ WEAK_BISIM (relinv Wbsm)
```

[EPS_AND_WEAK_TRANS]

```
⊢ ∀ E E₁ u E₂. EPS E E₁ ∧ E₁ =u⇒ E₂ ⟹ E =u⇒ E₂
```

[EPS_cases]

```
⊢ ∀ E E′.
    EPS E E′ ⟺ E −τ→ E′ ∨ (E = E′) ∨ ∃ E₁. EPS E E₁ ∧ EPS E₁ E′
```

[EPS_cases1]

$\vdash \forall x\ y.\ \texttt{EPS}\ x\ y \iff (x = y) \lor \exists u.\ x\ {-\tau\rightarrow}\ u \land \texttt{EPS}\ u\ y$

[EPS_cases2]

$\vdash \forall x\ y.\ \texttt{EPS}\ x\ y \iff (x = y) \lor \exists u.\ \texttt{EPS}\ x\ u \land u\ {-\tau\rightarrow}\ y$

[EPS_IMP_WEAK_TRANS]

$\vdash \forall E\ E'.\ \texttt{EPS}\ E\ E' \Longrightarrow (E = E') \lor E\ {=\tau\Rightarrow}\ E'$

[EPS_ind]

$$\begin{aligned}
&\vdash \forall P.\\
&\quad (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ x\ {-\tau\rightarrow}\ y \land P\ y\ z \Longrightarrow P\ x\ z) \Longrightarrow\\
&\quad \forall x\ y.\ \texttt{EPS}\ x\ y \Longrightarrow P\ x\ y
\end{aligned}$$

[EPS_ind_right]

$$\begin{aligned}
&\vdash \forall P.\\
&\quad (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ P\ x\ y \land y\ {-\tau\rightarrow}\ z \Longrightarrow P\ x\ z) \Longrightarrow\\
&\quad \forall x\ y.\ \texttt{EPS}\ x\ y \Longrightarrow P\ x\ y
\end{aligned}$$

[EPS_INDUCT]

$$\begin{aligned}
&\vdash \forall P.\\
&\quad (\forall E\ E'.\ E\ {-\tau\rightarrow}\ E' \Longrightarrow P\ E\ E') \land (\forall E.\ P\ E\ E) \land\\
&\quad (\forall E\ E_1\ E'.\ P\ E\ E_1 \land P\ E_1\ E' \Longrightarrow P\ E\ E') \Longrightarrow\\
&\quad \forall x\ y.\ \texttt{EPS}\ x\ y \Longrightarrow P\ x\ y
\end{aligned}$$

[EPS_PAR]

$$\begin{aligned}
&\vdash \forall E\ E'.\\
&\quad \texttt{EPS}\ E\ E' \Longrightarrow\\
&\quad \forall E''.\ \texttt{EPS}\ (E \parallel E'')\ (E' \parallel E'') \land \texttt{EPS}\ (E'' \parallel E)\ (E'' \parallel E')
\end{aligned}$$

[EPS_PAR_PAR]

$\vdash \forall E_1\ E_2\ F_1\ F_2.\ \texttt{EPS}\ E_1\ E_2 \land \texttt{EPS}\ F_1\ F_2 \Longrightarrow \texttt{EPS}\ (E_1 \parallel F_1)\ (E_2 \parallel F_2)$

[EPS_REFL]

$\vdash \forall E.\ \texttt{EPS}\ E\ E$

[EPS_RELAB]

$$\begin{aligned}
&\vdash \forall E\ E'.\\
&\quad \texttt{EPS}\ E\ E' \Longrightarrow\\
&\quad \forall \mathit{labl}.\ \texttt{EPS}\ (\texttt{relab}\ E\ (\texttt{RELAB}\ \mathit{labl}))\ (\texttt{relab}\ E'\ (\texttt{RELAB}\ \mathit{labl}))
\end{aligned}$$

[EPS_RELAB_rf]

$\vdash \forall E\ E'.\ \texttt{EPS}\ E\ E' \Longrightarrow \forall \mathit{rf}.\ \texttt{EPS}\ (\texttt{relab}\ E\ \mathit{rf})\ (\texttt{relab}\ E'\ \mathit{rf})$

[EPS_RESTR]

$\vdash \forall E\ E'.\ \texttt{EPS}\ E\ E' \Longrightarrow \forall L.\ \texttt{EPS}\ (\nu\ L\ E)\ (\nu\ L\ E')$

[EPS_STABLE]

$\vdash \forall E\ E'.\ \texttt{EPS}\ E\ E' \Longrightarrow \texttt{STABLE}\ E \Longrightarrow (E' = E)$

[EPS_STABLE']

$\vdash \forall E\ E'.\ \texttt{EPS}\ E\ E' \land \texttt{STABLE}\ E \Longrightarrow (E' = E)$

[EPS_strongind]

$$\begin{array}{l}\vdash \forall P. \\ \quad (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ x\ {-\tau\rightarrow}\ y \land \texttt{EPS}\ y\ z \land P\ y\ z \Longrightarrow P\ x\ z) \Longrightarrow \\ \quad \forall x\ y.\ \texttt{EPS}\ x\ y \Longrightarrow P\ x\ y\end{array}$$

[EPS_strongind_right]

$$\begin{array}{l}\vdash \forall P. \\ \quad (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ P\ x\ y \land \texttt{EPS}\ x\ y \land y\ {-\tau\rightarrow}\ z \Longrightarrow P\ x\ z) \Longrightarrow \\ \quad \forall x\ y.\ \texttt{EPS}\ x\ y \Longrightarrow P\ x\ y\end{array}$$

[EPS_TRANS]

$\vdash \forall x\ y\ z.\ \texttt{EPS}\ x\ y \land \texttt{EPS}\ y\ z \Longrightarrow \texttt{EPS}\ x\ z$

[EPS_TRANS_AUX]

$$\begin{array}{l}\vdash \forall E\ E_1. \\ \quad \texttt{EPS}\ E\ E_1 \Longrightarrow \\ \quad \forall \mathit{Wbsm}\ E'. \\ \quad\quad \texttt{WEAK\_BISIM}\ \mathit{Wbsm} \land \mathit{Wbsm}\ E\ E' \Longrightarrow \exists E_2.\ \texttt{EPS}\ E'\ E_2 \land \mathit{Wbsm}\ E_1\ E_2\end{array}$$

[EPS_TRANS_AUX_SYM]

$$\begin{array}{l}\vdash \forall E'\ E_1. \\ \quad \texttt{EPS}\ E'\ E_1 \Longrightarrow \\ \quad \forall \mathit{Wbsm}\ E. \\ \quad\quad \texttt{WEAK\_BISIM}\ \mathit{Wbsm} \land \mathit{Wbsm}\ E\ E' \Longrightarrow \exists E_2.\ \texttt{EPS}\ E\ E_2 \land \mathit{Wbsm}\ E_2\ E_1\end{array}$$

[EPS_WEAK_EPS]

$\vdash \forall E\ E_1\ u\ E_2\ E'.\ \texttt{EPS}\ E\ E_1 \land E_1\ {=u\Rightarrow}\ E_2 \land \texttt{EPS}\ E_2\ E' \Longrightarrow E\ {=u\Rightarrow}\ E'$

[EQUAL_IMP_WEAK_EQUIV]

$\vdash \forall E\ E'.\ (E = E') \Longrightarrow \texttt{WEAK\_EQUIV}\ E\ E'$

[IDENTITY_WEAK_BISIM]

$\vdash \texttt{WEAK\_BISIM}\ (\lambda x\ y.\ x = y)$

[INVERSE_REL]

$\vdash \forall R\ x\ y.\ \texttt{relinv}\ R\ x\ y \iff R\ y\ x$

[ONE_TAU]

$\vdash \forall E\ E'.\ E\ -\tau\rightarrow\ E' \implies \texttt{EPS}\ E\ E'$

[STABLE_cases]

$\vdash \forall E.\ \texttt{STABLE}\ E \lor \neg\texttt{STABLE}\ E$

[STABLE_NO_TRANS_TAU]

$\vdash \forall E.\ \texttt{STABLE}\ E \implies \forall E'.\ \neg(E\ -\tau\rightarrow\ E')$

[STABLE_NO_WEAK_TRANS_TAU]

$\vdash \forall E.\ \texttt{STABLE}\ E \implies \forall E'.\ \neg(E\ =\tau\Rightarrow\ E')$

[STRONG_EQUIV_EPS]

$$\begin{aligned}
&\vdash \forall E\ E'.\\
&\qquad \texttt{STRONG\_EQUIV}\ E\ E' \implies\\
&\qquad \forall E_1.\ \texttt{EPS}\ E\ E_1 \implies \exists E_2.\ \texttt{EPS}\ E'\ E_2 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2
\end{aligned}$$

[STRONG_EQUIV_EPS']

$$\begin{aligned}
&\vdash \forall E\ E'.\\
&\qquad \texttt{STRONG\_EQUIV}\ E\ E' \implies\\
&\qquad \forall E_2.\ \texttt{EPS}\ E'\ E_2 \implies \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2
\end{aligned}$$

[STRONG_EQUIV_WEAK_TRANS]

$$\begin{aligned}
&\vdash \forall E\ E'.\\
&\qquad \texttt{STRONG\_EQUIV}\ E\ E' \implies\\
&\qquad \forall u\ E_1.\ E\ =u\Rightarrow\ E_1 \implies \exists E_2.\ E'\ =u\Rightarrow\ E_2 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2
\end{aligned}$$

[STRONG_EQUIV_WEAK_TRANS']

$$\begin{aligned}
&\vdash \forall E\ E'.\\
&\qquad \texttt{STRONG\_EQUIV}\ E\ E' \implies\\
&\qquad \forall u\ E_2.\ E'\ =u\Rightarrow\ E_2 \implies \exists E_1.\ E\ =u\Rightarrow\ E_1 \land \texttt{STRONG\_EQUIV}\ E_1\ E_2
\end{aligned}$$

[STRONG_IMP_WEAK_BISIM]

$\vdash \forall Bsm.\ \texttt{STRONG\_BISIM}\ Bsm \implies \texttt{WEAK\_BISIM}\ Bsm$

[STRONG_IMP_WEAK_EQUIV]

$\vdash \forall E\ E'.\ \texttt{STRONG\_EQUIV}\ E\ E' \implies \texttt{WEAK\_EQUIV}\ E\ E'$

[TAU_PREFIX_EPS]

$\vdash \forall E\ E'.\ \texttt{EPS}\ E\ E' \implies \texttt{EPS}\ (\tau..E)\ E'$

[TAU_PREFIX_WEAK_TRANS]

$$\vdash \forall E\ u\ E'.\ E\ {=}u{\Rightarrow}\ E' \implies \tau..E\ {=}u{\Rightarrow}\ E'$$

[TRANS_AND_EPS]

$$\vdash \forall E\ u\ E_1\ E'.\ E\ {-}u{\rightarrow}\ E_1 \land \texttt{EPS}\ E_1\ E' \implies E\ {=}u{\Rightarrow}\ E'$$

[TRANS_IMP_WEAK_TRANS]

$$\vdash \forall E\ u\ E'.\ E\ {-}u{\rightarrow}\ E' \implies E\ {=}u{\Rightarrow}\ E'$$

[TRANS_TAU_AND_WEAK]

$$\vdash \forall E\ E_1\ u\ E'.\ E\ {-}\tau{\rightarrow}\ E_1 \land E_1\ {=}u{\Rightarrow}\ E' \implies E\ {=}u{\Rightarrow}\ E'$$

[TRANS_TAU_IMP_EPS]

$$\vdash \forall E\ E'.\ E\ {-}\tau{\rightarrow}\ E' \implies \texttt{EPS}\ E\ E'$$

[UNION_WEAK_BISIM]

$$\begin{array}{l}
\vdash \forall\, \mathit{Wbsm}_1\ \mathit{Wbsm}_2.\\
\qquad \texttt{WEAK\_BISIM}\ \mathit{Wbsm}_1 \land \texttt{WEAK\_BISIM}\ \mathit{Wbsm}_2 \implies\\
\qquad \texttt{WEAK\_BISIM}\ (\mathit{Wbsm}_1\ \texttt{RUNION}\ \mathit{Wbsm}_2)
\end{array}$$

[WEAK_BISIM]

$$\begin{array}{l}
\vdash \texttt{WEAK\_BISIM}\ \mathit{Wbsm} \iff\\
\quad \forall E\ E'.\\
\qquad \mathit{Wbsm}\ E\ E' \implies\\
\qquad (\forall l.\\
\qquad\quad (\forall E_1.\\
\qquad\qquad E\ {-}\texttt{label}\ l{\rightarrow}\ E_1 \implies \exists E_2.\ E'\ {=}\texttt{label}\ l{\Rightarrow}\ E_2 \land \mathit{Wbsm}\ E_1\ E_2) \land\\
\qquad\quad \forall E_2.\\
\qquad\qquad E'\ {-}\texttt{label}\ l{\rightarrow}\ E_2 \implies \exists E_1.\ E\ {=}\texttt{label}\ l{\Rightarrow}\ E_1 \land \mathit{Wbsm}\ E_1\ E_2) \land\\
\qquad (\forall E_1.\ E\ {-}\tau{\rightarrow}\ E_1 \implies \exists E_2.\ \texttt{EPS}\ E'\ E_2 \land \mathit{Wbsm}\ E_1\ E_2) \land\\
\qquad \forall E_2.\ E'\ {-}\tau{\rightarrow}\ E_2 \implies \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \mathit{Wbsm}\ E_1\ E_2
\end{array}$$

[WEAK_BISIM_SUBSET_WEAK_EQUIV]

$$\vdash \forall\, \mathit{Wbsm}.\ \texttt{WEAK\_BISIM}\ \mathit{Wbsm} \implies \mathit{Wbsm}\ \texttt{RSUBSET}\ \texttt{WEAK\_EQUIV}$$

[WEAK_EQUIV]

$$\vdash \forall E\ E'.\ \texttt{WEAK\_EQUIV}\ E\ E' \iff \exists\, \mathit{Wbsm}.\ \mathit{Wbsm}\ E\ E' \land \texttt{WEAK\_BISIM}\ \mathit{Wbsm}$$

[WEAK_EQUIV_cases]

$$\begin{array}{l}
\vdash \forall a_0\ a_1.\\
\qquad \texttt{WEAK\_EQUIV}\ a_0\ a_1 \iff\\
\qquad (\forall l.\\
\qquad\quad (\forall E_1.\\
\qquad\qquad a_0\ {-}\texttt{label}\ l{\rightarrow}\ E_1 \implies\\
\qquad\qquad \exists E_2.\ a_1\ {=}\texttt{label}\ l{\Rightarrow}\ E_2 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2) \land\\
\qquad\quad \forall E_2.\\
\qquad\qquad a_1\ {-}\texttt{label}\ l{\rightarrow}\ E_2 \implies\\
\qquad\qquad \exists E_1.\ a_0\ {=}\texttt{label}\ l{\Rightarrow}\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2) \land\\
\qquad (\forall E_1.\ a_0\ {-}\tau{\rightarrow}\ E_1 \implies \exists E_2.\ \texttt{EPS}\ a_1\ E_2 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2) \land\\
\qquad \forall E_2.\ a_1\ {-}\tau{\rightarrow}\ E_2 \implies \exists E_1.\ \texttt{EPS}\ a_0\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[WEAK_EQUIV_coind]

$$\begin{array}{l}
\vdash \forall\, \mathit{WEAK\_EQUIV}'.\\
\quad\quad (\forall\, a_0\ \ a_1.\\
\quad\quad\quad\quad \mathit{WEAK\_EQUIV}'\ \ a_0\ \ a_1 \implies\\
\quad\quad\quad\quad (\forall\, l.\\
\quad\quad\quad\quad\quad\quad (\forall\, E_1.\\
\quad\quad\quad\quad\quad\quad\quad\quad a_0\ \ -\texttt{label}\ \ l\!\rightarrow\ E_1 \implies\\
\quad\quad\quad\quad\quad\quad\quad\quad \exists\, E_2.\ \ a_1\ \ =\!\texttt{label}\ \ l\!\Rightarrow\ E_2\ \wedge\ \mathit{WEAK\_EQUIV}'\ \ E_1\ \ E_2)\ \wedge\\
\quad\quad\quad\quad\quad\quad \forall\, E_2.\\
\quad\quad\quad\quad\quad\quad\quad a_1\ \ -\texttt{label}\ \ l\!\rightarrow\ E_2 \implies\\
\quad\quad\quad\quad\quad\quad\quad \exists\, E_1.\ \ a_0\ \ =\!\texttt{label}\ \ l\!\Rightarrow\ E_1\ \wedge\ \mathit{WEAK\_EQUIV}'\ \ E_1\ \ E_2)\ \wedge\\
\quad\quad\quad\quad (\forall\, E_1.\ \ a_0\ \ -\tau\!\rightarrow\ E_1 \implies \exists\, E_2.\ \ \texttt{EPS}\ \ a_1\ \ E_2\ \wedge\ \mathit{WEAK\_EQUIV}'\ \ E_1\ \ E_2)\ \wedge\\
\quad\quad\quad\quad \forall\, E_2.\ \ a_1\ \ -\tau\!\rightarrow\ E_2 \implies \exists\, E_1.\ \ \texttt{EPS}\ \ a_0\ \ E_1\ \wedge\ \mathit{WEAK\_EQUIV}'\ \ E_1\ \ E_2) \implies\\
\quad\quad \forall\, a_0\ \ a_1.\ \ \mathit{WEAK\_EQUIV}'\ \ a_0\ \ a_1 \implies \texttt{WEAK\_EQUIV}\ \ a_0\ \ a_1
\end{array}$$

[WEAK_EQUIV_EPS]

$$\begin{array}{l}
\vdash \forall\, E\ \ E'.\\
\quad\quad \texttt{WEAK\_EQUIV}\ \ E\ \ E' \implies\\
\quad\quad \forall\, E_1.\ \ \texttt{EPS}\ \ E\ \ E_1 \implies \exists\, E_2.\ \ \texttt{EPS}\ \ E'\ \ E_2\ \wedge\ \texttt{WEAK\_EQUIV}\ \ E_1\ \ E_2
\end{array}$$

[WEAK_EQUIV_EPS']

$$\begin{array}{l}
\vdash \forall\, E\ \ E'.\\
\quad\quad \texttt{WEAK\_EQUIV}\ \ E\ \ E' \implies\\
\quad\quad \forall\, E_2.\ \ \texttt{EPS}\ \ E'\ \ E_2 \implies \exists\, E_1.\ \ \texttt{EPS}\ \ E\ \ E_1\ \wedge\ \texttt{WEAK\_EQUIV}\ \ E_1\ \ E_2
\end{array}$$

[WEAK_EQUIV_equivalence]

$$\vdash \texttt{equivalence WEAK\_EQUIV}$$

[WEAK_EQUIV_IS_WEAK_BISIM]

$$\vdash \texttt{WEAK\_BISIM WEAK\_EQUIV}$$

[WEAK_EQUIV_PRESD_BY_GUARDED_SUM]

$$\begin{array}{l}
\vdash \forall\, E_1\ \ E_1'\ \ E_2\ \ E_2'\ \ a_1\ \ a_2.\\
\quad\quad \texttt{WEAK\_EQUIV}\ \ E_1\ \ E_1'\ \wedge\ \texttt{WEAK\_EQUIV}\ \ E_2\ \ E_2' \implies\\
\quad\quad \texttt{WEAK\_EQUIV}\ \ (a_1\,..\,E_1\ \ \texttt{+}\ \ a_2\,..\,E_2)\ \ (a_1\,..\,E_1'\ \ \texttt{+}\ \ a_2\,..\,E_2')
\end{array}$$

[WEAK_EQUIV_PRESD_BY_PAR]

$$\begin{array}{l}
\vdash \forall\, E_1\ \ E_1'\ \ E_2\ \ E_2'.\\
\quad\quad \texttt{WEAK\_EQUIV}\ \ E_1\ \ E_1'\ \wedge\ \texttt{WEAK\_EQUIV}\ \ E_2\ \ E_2' \implies\\
\quad\quad \texttt{WEAK\_EQUIV}\ \ (E_1\ \parallel\ E_2)\ \ (E_1'\ \parallel\ E_2')
\end{array}$$

[WEAK_EQUIV_PRESD_BY_SUM]

$$\begin{array}{l}
\vdash \forall\, E_1\ \ E_1'\ \ E_2\ \ E_2'.\\
\quad\quad \texttt{WEAK\_EQUIV}\ \ E_1\ \ E_1'\ \wedge\ \texttt{STABLE}\ \ E_1\ \wedge\ \texttt{STABLE}\ \ E_1'\ \wedge\\
\quad\quad \texttt{WEAK\_EQUIV}\ \ E_2\ \ E_2'\ \wedge\ \texttt{STABLE}\ \ E_2\ \wedge\ \texttt{STABLE}\ \ E_2' \implies\\
\quad\quad \texttt{WEAK\_EQUIV}\ \ (E_1\ \ \texttt{+}\ \ E_2)\ \ (E_1'\ \ \texttt{+}\ \ E_2')
\end{array}$$

[WEAK_EQUIV_REFL]

```
⊢ ∀E. WEAK_EQUIV E E
```

[WEAK_EQUIV_rules]

```
⊢ ∀E E′.
    (∀l.
       (∀E1.
          E −label l→ E1 ⟹
          ∃E2. E′ =label l⇒ E2 ∧ WEAK_EQUIV E1 E2) ∧
       ∀E2.
         E′ −label l→ E2 ⟹
         ∃E1. E =label l⇒ E1 ∧ WEAK_EQUIV E1 E2) ∧
    (∀E1. E −τ→ E1 ⟹ ∃E2. EPS E′ E2 ∧ WEAK_EQUIV E1 E2) ∧
    (∀E2. E′ −τ→ E2 ⟹ ∃E1. EPS E E1 ∧ WEAK_EQUIV E1 E2) ⟹
    WEAK_EQUIV E E′
```

[WEAK_EQUIV_SUBST_PAR_L]

```
⊢ ∀E E′.
    WEAK_EQUIV E E′ ⟹ ∀E″. WEAK_EQUIV (E″ ‖ E) (E″ ‖ E′)
```

[WEAK_EQUIV_SUBST_PAR_R]

```
⊢ ∀E E′.
    WEAK_EQUIV E E′ ⟹ ∀E″. WEAK_EQUIV (E ‖ E″) (E′ ‖ E″)
```

[WEAK_EQUIV_SUBST_PREFIX]

```
⊢ ∀E E′. WEAK_EQUIV E E′ ⟹ ∀u. WEAK_EQUIV (u..E) (u..E′)
```

[WEAK_EQUIV_SUBST_RELAB]

```
⊢ ∀E E′.
    WEAK_EQUIV E E′ ⟹
    ∀rf. WEAK_EQUIV (relab E rf) (relab E′ rf)
```

[WEAK_EQUIV_SUBST_RESTR]

```
⊢ ∀E E′. WEAK_EQUIV E E′ ⟹ ∀L. WEAK_EQUIV (ν L E) (ν L E′)
```

[WEAK_EQUIV_SUBST_SUM_R]

```
⊢ ∀E E′.
    WEAK_EQUIV E E′ ∧ STABLE E ∧ STABLE E′ ⟹
    ∀E″. WEAK_EQUIV (E + E″) (E′ + E″)
```

[WEAK_EQUIV_SYM]

```
⊢ ∀E E′. WEAK_EQUIV E E′ ⟹ WEAK_EQUIV E′ E
```

[WEAK_EQUIV_SYM’]

```
⊢ ∀E E′. WEAK_EQUIV E E′ ⟺ WEAK_EQUIV E′ E
```

[WEAK_EQUIV_TRANS]

$$\begin{array}{l}\vdash \forall E\ E'\ E''.\\ \qquad \texttt{WEAK\_EQUIV}\ E\ E' \land \texttt{WEAK\_EQUIV}\ E'\ E'' \Longrightarrow \texttt{WEAK\_EQUIV}\ E\ E''\end{array}$$

[WEAK_EQUIV_TRANS_label]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \qquad \texttt{WEAK\_EQUIV}\ E\ E' \Longrightarrow\\ \qquad \forall l\ E_1.\\ \qquad\quad E\ -\texttt{label}\ l\!\rightarrow\ E_1 \Longrightarrow \exists E_2.\ E'\ =\texttt{label}\ l\!\Rightarrow\ E_2 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[WEAK_EQUIV_TRANS_label']

$$\begin{array}{l}\vdash \forall E\ E'.\\ \qquad \texttt{WEAK\_EQUIV}\ E\ E' \Longrightarrow\\ \qquad \forall l\ E_2.\\ \qquad\quad E'\ -\texttt{label}\ l\!\rightarrow\ E_2 \Longrightarrow \exists E_1.\ E\ =\texttt{label}\ l\!\Rightarrow\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[WEAK_EQUIV_TRANS_tau]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \qquad \texttt{WEAK\_EQUIV}\ E\ E' \Longrightarrow\\ \qquad \forall E_1.\ E\ -\tau\!\rightarrow\ E_1 \Longrightarrow \exists E_2.\ \texttt{EPS}\ E'\ E_2 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[WEAK_EQUIV_TRANS_tau']

$$\begin{array}{l}\vdash \forall E\ E'.\\ \qquad \texttt{WEAK\_EQUIV}\ E\ E' \Longrightarrow\\ \qquad \forall E_2.\ E'\ -\tau\!\rightarrow\ E_2 \Longrightarrow \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[WEAK_EQUIV_WEAK_TRANS_label]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \qquad \texttt{WEAK\_EQUIV}\ E\ E' \Longrightarrow\\ \qquad \forall l\ E_1.\\ \qquad\quad E\ =\texttt{label}\ l\!\Rightarrow\ E_1 \Longrightarrow \exists E_2.\ E'\ =\texttt{label}\ l\!\Rightarrow\ E_2 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[WEAK_EQUIV_WEAK_TRANS_label']

$$\begin{array}{l}\vdash \forall E\ E'.\\ \qquad \texttt{WEAK\_EQUIV}\ E\ E' \Longrightarrow\\ \qquad \forall l\ E_2.\\ \qquad\quad E'\ =\texttt{label}\ l\!\Rightarrow\ E_2 \Longrightarrow \exists E_1.\ E\ =\texttt{label}\ l\!\Rightarrow\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[WEAK_EQUIV_WEAK_TRANS_tau]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \qquad \texttt{WEAK\_EQUIV}\ E\ E' \Longrightarrow\\ \qquad \forall E_1.\ E\ =\tau\!\Rightarrow\ E_1 \Longrightarrow \exists E_2.\ \texttt{EPS}\ E'\ E_2 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[WEAK_EQUIV_WEAK_TRANS_tau']

$$\begin{array}{l}\vdash \forall E\ E'.\\ \qquad \texttt{WEAK\_EQUIV}\ E\ E' \Longrightarrow\\ \qquad \forall E_2.\ E'\ =\tau\!\Rightarrow\ E_2 \Longrightarrow \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[WEAK_PAR]

$$\begin{array}{l}\vdash \forall E\ u\ E'.\\ \quad E\ {=}u{\Rightarrow}\ E' \implies\\ \quad \forall E''.\ E \parallel E''\ {=}u{\Rightarrow}\ E' \parallel E'' \land E'' \parallel E\ {=}u{\Rightarrow}\ E'' \parallel E'\end{array}$$

[WEAK_PREFIX]

$$\vdash \forall E\ u.\ u..E\ {=}u{\Rightarrow}\ E$$

[WEAK_PROPERTY_STAR]

$$\begin{array}{l}\vdash \forall a_0\ a_1.\\ \quad \mathtt{WEAK\_EQUIV}\ a_0\ a_1 \iff\\ \quad (\forall l.\\ \qquad (\forall E_1.\\ \qquad\quad a_0\ {-}\mathtt{label}\ l{\rightarrow}\ E_1 \implies\\ \qquad\quad \exists E_2.\ a_1\ {=}\mathtt{label}\ l{\Rightarrow}\ E_2 \land \mathtt{WEAK\_EQUIV}\ E_1\ E_2) \land\\ \qquad \forall E_2.\\ \qquad\quad a_1\ {-}\mathtt{label}\ l{\rightarrow}\ E_2 \implies\\ \qquad\quad \exists E_1.\ a_0\ {=}\mathtt{label}\ l{\Rightarrow}\ E_1 \land \mathtt{WEAK\_EQUIV}\ E_1\ E_2) \land\\ \quad (\forall E_1.\ a_0\ {-}\tau{\rightarrow}\ E_1 \implies \exists E_2.\ \mathtt{EPS}\ a_1\ E_2 \land \mathtt{WEAK\_EQUIV}\ E_1\ E_2) \land\\ \quad \forall E_2.\ a_1\ {-}\tau{\rightarrow}\ E_2 \implies \exists E_1.\ \mathtt{EPS}\ a_0\ E_1 \land \mathtt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[WEAK_RELAB]

$$\begin{array}{l}\vdash \forall E\ u\ E'.\\ \quad E\ {=}u{\Rightarrow}\ E' \implies\\ \quad \forall \mathit{labl}.\\ \qquad \mathtt{relab}\ E\ (\mathtt{RELAB}\ \mathit{labl})\\ \qquad {=}\mathtt{relabel}\ (\mathtt{RELAB}\ \mathit{labl})\ u{\Rightarrow}\\ \qquad \mathtt{relab}\ E'\ (\mathtt{RELAB}\ \mathit{labl})\end{array}$$

[WEAK_RELAB_rf]

$$\begin{array}{l}\vdash \forall E\ u\ E'.\\ \quad E\ {=}u{\Rightarrow}\ E' \implies \forall \mathit{rf}.\ \mathtt{relab}\ E\ \mathit{rf}\ {=}\mathtt{relabel}\ \mathit{rf}\ u{\Rightarrow}\ \mathtt{relab}\ E'\ \mathit{rf}\end{array}$$

[WEAK_RESTR_label]

$$\begin{array}{l}\vdash \forall l\ L\ E\ E'.\\ \quad l \notin L \land \mathtt{COMPL}\ l \notin L \land E\ {=}\mathtt{label}\ l{\Rightarrow}\ E' \implies\\ \quad \nu\ L\ E\ {=}\mathtt{label}\ l{\Rightarrow}\ \nu\ L\ E'\end{array}$$

[WEAK_RESTR_tau]

$$\vdash \forall E\ E'.\ E\ {=}\tau{\Rightarrow}\ E' \implies \forall L.\ \nu\ L\ E\ {=}\tau{\Rightarrow}\ \nu\ L\ E'$$

[WEAK_SUM1]

$$\vdash \forall E\ u\ E_1\ E'.\ E\ {=}u{\Rightarrow}\ E_1 \implies E\ \mathtt{+}\ E'\ {=}u{\Rightarrow}\ E_1$$

[WEAK_SUM2]

$$\vdash \forall E\ u\ E_1\ E'.\ E\ {=}u{\Rightarrow}\ E_1 \Longrightarrow E'\ \mathtt{+}\ E\ {=}u{\Rightarrow}\ E_1$$

[WEAK_TRANS_AND_EPS]

$$\vdash \forall E_1\ u\ E_2\ E'.\ E_1\ {=}u{\Rightarrow}\ E_2 \wedge \mathtt{EPS}\ E_2\ E' \Longrightarrow E_1\ {=}u{\Rightarrow}\ E'$$

[WEAK_TRANS_AUX]

$$\begin{array}{l}
\vdash \forall E\ l\ E_1.\\
\qquad E\ {=}\mathtt{label}\ l{\Rightarrow}\ E_1 \Longrightarrow\\
\qquad \forall \mathit{Wbsm}\ E'.\\
\qquad\quad \mathtt{WEAK\_BISIM}\ \mathit{Wbsm} \wedge \mathit{Wbsm}\ E\ E' \Longrightarrow\\
\qquad\quad \exists E_2.\ E'\ {=}\mathtt{label}\ l{\Rightarrow}\ E_2 \wedge \mathit{Wbsm}\ E_1\ E_2
\end{array}$$

[WEAK_TRANS_AUX_SYM]

$$\begin{array}{l}
\vdash \forall E'\ l\ E_1.\\
\qquad E'\ {=}\mathtt{label}\ l{\Rightarrow}\ E_1 \Longrightarrow\\
\qquad \forall \mathit{Wbsm}\ E.\\
\qquad\quad \mathtt{WEAK\_BISIM}\ \mathit{Wbsm} \wedge \mathit{Wbsm}\ E\ E' \Longrightarrow\\
\qquad\quad \exists E_2.\ E\ {=}\mathtt{label}\ l{\Rightarrow}\ E_2 \wedge \mathit{Wbsm}\ E_2\ E_1
\end{array}$$

[WEAK_TRANS_cases1]

$$\begin{array}{l}
\vdash \forall E\ u\ E_1.\\
\qquad E\ {=}u{\Rightarrow}\ E_1 \Longrightarrow\\
\qquad (\exists E'.\ E\ {-}\tau{\rightarrow}\ E' \wedge E'\ {=}u{\Rightarrow}\ E_1) \vee \exists E'.\ E\ {-}u{\rightarrow}\ E' \wedge \mathtt{EPS}\ E'\ E_1
\end{array}$$

[WEAK_TRANS_cases2]

$$\begin{array}{l}
\vdash \forall E\ l\ E_1.\\
\qquad E\ {=}\mathtt{label}\ l{\Rightarrow}\ E_1 \Longrightarrow\\
\qquad (\exists E'.\ E\ {-}\tau{\rightarrow}\ E' \wedge E'\ {=}\mathtt{label}\ l{\Rightarrow}\ E_1) \vee\\
\qquad \exists E'.\ E\ {-}\mathtt{label}\ l{\rightarrow}\ E' \wedge \mathtt{EPS}\ E'\ E_1
\end{array}$$

[WEAK_TRANS_IMP_EPS]

$$\vdash \forall E\ E'.\ E\ {=}\tau{\Rightarrow}\ E' \Longrightarrow \mathtt{EPS}\ E\ E'$$

[WEAK_TRANS_STABLE]

$$\begin{array}{l}
\vdash \forall E\ l\ E'.\\
\qquad E\ {=}\mathtt{label}\ l{\Rightarrow}\ E' \wedge \mathtt{STABLE}\ E \Longrightarrow\\
\qquad \exists E''.\ E\ {-}\mathtt{label}\ l{\rightarrow}\ E'' \wedge \mathtt{EPS}\ E''\ E'
\end{array}$$

[WEAK_TRANS_TAU]

$$\vdash \forall E\ E_1.\ E\ {=}\tau{\Rightarrow}\ E_1 \iff \exists E'.\ E\ {-}\tau{\rightarrow}\ E' \wedge \mathtt{EPS}\ E'\ E_1$$

[WEAK_TRANS_TAU_IMP_TRANS_TAU]

$$\vdash \forall E\ E'.\ E\ {=}\tau{\Rightarrow}\ E' \Longrightarrow \exists E_1.\ E\ {-}\tau{\rightarrow}\ E_1 \wedge \mathtt{EPS}\ E_1\ E'$$

9.5 WeakLaws Theory

Built: 08 Dicembre 2017
Parent Theories: WeakEQ, StrongLaws

9.5.1 Theorems

[TAU_WEAK]

```
⊢ ∀E. WEAK_EQUIV (τ..E) E
```

[WEAK_EQUIV_SUBST_SUM_L]

```
⊢ ∀E E′.
    WEAK_EQUIV E E′ ∧ STABLE E ∧ STABLE E′ ⟹
    ∀E″. WEAK_EQUIV (E″ + E) (E″ + E′)
```

[WEAK_EXPANSION_LAW]

```
⊢ ∀f n f′ m.
    (∀i. i ≤ n ⟹ Is_Prefix (f i)) ∧
    (∀j. j ≤ m ⟹ Is_Prefix (f′ j)) ⟹
    WEAK_EQUIV (SIGMA f n ∥ SIGMA f′ m)
      (SIGMA
         (λi. PREF_ACT (f i)..(PREF_PROC (f i) ∥ SIGMA f′ m))
         n +
       SIGMA
         (λj. PREF_ACT (f′ j)..(SIGMA f n ∥ PREF_PROC (f′ j)))
         m + ALL_SYNC f n f′ m)
```

[WEAK_PAR_ASSOC]

```
⊢ ∀E E′ E″. WEAK_EQUIV (E ∥ E′ ∥ E″) (E ∥ (E′ ∥ E″))
```

[WEAK_PAR_COMM]

```
⊢ ∀E E′. WEAK_EQUIV (E ∥ E′) (E′ ∥ E)
```

[WEAK_PAR_IDENT_L]

```
⊢ ∀E. WEAK_EQUIV (nil ∥ E) E
```

[WEAK_PAR_IDENT_R]

```
⊢ ∀E. WEAK_EQUIV (E ∥ nil) E
```

[WEAK_PAR_PREF_NO_SYNCR]

```
⊢ ∀l l′.
    l ≠ COMPL l′ ⟹
    ∀E E′.
      WEAK_EQUIV (label l..E ∥ label l′..E′)
        (label l..(E ∥ label l′..E′) +
         label l′..(label l..E ∥ E′))
```

[WEAK_PAR_PREF_SYNCR]

```
⊢ ∀l l′.
    (l = COMPL l′) ⟹
    ∀E E′.
      WEAK_EQUIV (label l..E ∥ label l′..E′)
        (label l..(E ∥ label l′..E′) +
         label l′..(label l..E ∥ E′) + τ..(E ∥ E′))
```

[WEAK_PAR_PREF_TAU]

```
⊢ ∀u E E′.
    WEAK_EQUIV (u..E ∥ τ..E′) (u..(E ∥ τ..E′) + τ..(u..E ∥ E′))
```

[WEAK_PAR_TAU_PREF]

```
⊢ ∀E u E′.
    WEAK_EQUIV (τ..E ∥ u..E′) (τ..(E ∥ u..E′) + u..(τ..E ∥ E′))
```

[WEAK_PAR_TAU_TAU]

```
⊢ ∀E E′.
    WEAK_EQUIV (τ..E ∥ τ..E′) (τ..(E ∥ τ..E′) + τ..(τ..E ∥ E′))
```

[WEAK_PREF_REC_EQUIV]

```
⊢ ∀u s v.
    WEAK_EQUIV (u..rec s (v..u..var s)) (rec s (u..v..var s))
```

[WEAK_RELAB_NIL]

```
⊢ ∀rf. WEAK_EQUIV (relab nil rf) nil
```

[WEAK_RELAB_PREFIX]

```
⊢ ∀u E labl.
    WEAK_EQUIV (relab (u..E) (RELAB labl))
      (relabel (RELAB labl) u..relab E (RELAB labl))
```

[WEAK_RELAB_SUM]

```
⊢ ∀E E′ rf.
    WEAK_EQUIV (relab (E + E′) rf) (relab E rf + relab E′ rf)
```

[WEAK_RESTR_NIL]

```
⊢ ∀L. WEAK_EQUIV (ν L nil) nil
```

[WEAK_RESTR_PR_LAB_NIL]

```
⊢ ∀l L.
    l ∈ L ∨ COMPL l ∈ L ⟹ ∀E. WEAK_EQUIV (ν L (label l..E)) nil
```

[WEAK_RESTR_PREFIX_LABEL]

$\vdash \forall l\ L.$
$\quad l \notin L \land \texttt{COMPL}\ l \notin L \implies$
$\quad \forall E.\ \texttt{WEAK\_EQUIV}\ (\nu\ L\ (\texttt{label}\ l\texttt{..}E))\ (\texttt{label}\ l\texttt{..}\nu\ L\ E)$

[WEAK_RESTR_PREFIX_TAU]

$\vdash \forall E\ L.\ \texttt{WEAK\_EQUIV}\ (\nu\ L\ (\tau\texttt{..}E))\ (\tau\texttt{..}\nu\ L\ E)$

[WEAK_RESTR_SUM]

$\vdash \forall E\ E'\ L.\ \texttt{WEAK\_EQUIV}\ (\nu\ L\ (E + E'))\ (\nu\ L\ E + \nu\ L\ E')$

[WEAK_SUM_ASSOC_L]

$\vdash \forall E\ E'\ E''.\ \texttt{WEAK\_EQUIV}\ (E + (E' + E''))\ (E + E' + E'')$

[WEAK_SUM_ASSOC_R]

$\vdash \forall E\ E'\ E''.\ \texttt{WEAK\_EQUIV}\ (E + E' + E'')\ (E + (E' + E''))$

[WEAK_SUM_COMM]

$\vdash \forall E\ E'.\ \texttt{WEAK\_EQUIV}\ (E + E')\ (E' + E)$

[WEAK_SUM_IDEMP]

$\vdash \forall E.\ \texttt{WEAK\_EQUIV}\ (E + E)\ E$

[WEAK_SUM_IDENT_L]

$\vdash \forall E.\ \texttt{WEAK\_EQUIV}\ (\texttt{nil} + E)\ E$

[WEAK_SUM_IDENT_R]

$\vdash \forall E.\ \texttt{WEAK\_EQUIV}\ (E + \texttt{nil})\ E$

[WEAK_UNFOLDING]

$\vdash \forall X\ E.\ \texttt{WEAK\_EQUIV}\ (\texttt{rec}\ X\ E)\ (\texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X)$

9.6 ObsCongr Theory

Built: 08 Dicembre 2017
Parent Theories: WeakLaws

9.6.1 Definitions

[OBS_CONGR]

$\vdash \forall E\ E'.$
$\quad \texttt{OBS\_CONGR}\ E\ E' \iff$
$\quad \forall u.$
$\quad\quad (\forall E_1.\ E\ {-u\!\rightarrow}\ E_1 \implies \exists E_2.\ E'\ {=u\!\Rightarrow}\ E_2 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2) \land$
$\quad\quad \forall E_2.\ E'\ {-u\!\rightarrow}\ E_2 \implies \exists E_1.\ E\ {=u\!\Rightarrow}\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2$

9.6.2 Theorems

[EQUAL_IMP_OBS_CONGR]

$$\vdash \forall E\ E'.\ (E = E') \Longrightarrow \texttt{OBS\_CONGR}\ E\ E'$$

[OBS_CONGR_BY_WEAK_BISIM]

$$\begin{array}{l}
\vdash \forall\, \mathit{Wbsm}. \\
\quad\quad \texttt{WEAK\_BISIM}\ \mathit{Wbsm} \Longrightarrow \\
\quad\quad \forall E\ E'. \\
\quad\quad\quad (\forall u. \\
\quad\quad\quad\quad (\forall E_1.\ E\ -u\!\rightarrow\ E_1 \Longrightarrow \exists E_2.\ E'\ =u\!\Rightarrow\ E_2 \wedge \mathit{Wbsm}\ E_1\ E_2) \wedge \\
\quad\quad\quad\quad \forall E_2.\ E'\ -u\!\rightarrow\ E_2 \Longrightarrow \exists E_1.\ E\ =u\!\Rightarrow\ E_1 \wedge \mathit{Wbsm}\ E_1\ E_2) \Longrightarrow \\
\quad\quad\quad \texttt{OBS\_CONGR}\ E\ E'
\end{array}$$

[OBS_CONGR_EPS]

$$\begin{array}{l}
\vdash \forall E\ E'. \\
\quad\quad \texttt{OBS\_CONGR}\ E\ E' \Longrightarrow \\
\quad\quad \forall E_1.\ \texttt{EPS}\ E\ E_1 \Longrightarrow \exists E_2.\ \texttt{EPS}\ E'\ E_2 \wedge \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[OBS_CONGR_EPS']

$$\begin{array}{l}
\vdash \forall E\ E'. \\
\quad\quad \texttt{OBS\_CONGR}\ E\ E' \Longrightarrow \\
\quad\quad \forall E_2.\ \texttt{EPS}\ E'\ E_2 \Longrightarrow \exists E_1.\ \texttt{EPS}\ E\ E_1 \wedge \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[OBS_CONGR_equivalence]

$$\vdash \texttt{equivalence OBS\_CONGR}$$

[OBS_CONGR_IMP_WEAK_EQUIV]

$$\vdash \forall E\ E'.\ \texttt{OBS\_CONGR}\ E\ E' \Longrightarrow \texttt{WEAK\_EQUIV}\ E\ E'$$

[OBS_CONGR_PRESD_BY_PAR]

$$\begin{array}{l}
\vdash \forall E_1\ E_1'\ E_2\ E_2'. \\
\quad\quad \texttt{OBS\_CONGR}\ E_1\ E_1' \wedge \texttt{OBS\_CONGR}\ E_2\ E_2' \Longrightarrow \\
\quad\quad \texttt{OBS\_CONGR}\ (E_1 \parallel E_2)\ (E_1' \parallel E_2')
\end{array}$$

[OBS_CONGR_PRESD_BY_SUM]

$$\begin{array}{l}
\vdash \forall E_1\ E_1'\ E_2\ E_2'. \\
\quad\quad \texttt{OBS\_CONGR}\ E_1\ E_1' \wedge \texttt{OBS\_CONGR}\ E_2\ E_2' \Longrightarrow \\
\quad\quad \texttt{OBS\_CONGR}\ (E_1 + E_2)\ (E_1' + E_2')
\end{array}$$

[OBS_CONGR_REFL]

$$\vdash \forall E.\ \texttt{OBS\_CONGR}\ E\ E$$

[OBS_CONGR_SUBST_PAR_L]

$$\vdash \forall E\ E'.\ \texttt{OBS\_CONGR}\ E\ E' \Longrightarrow \forall E''.\ \texttt{OBS\_CONGR}\ (E'' \parallel E)\ (E'' \parallel E')$$

[OBS_CONGR_SUBST_PAR_R]
$\vdash \forall E\ E'.\ \texttt{OBS\_CONGR}\ E\ E' \implies \forall E''.\ \texttt{OBS\_CONGR}\ (E \parallel E'')\ (E' \parallel E'')$

[OBS_CONGR_SUBST_PREFIX]
$\vdash \forall E\ E'.\ \texttt{OBS\_CONGR}\ E\ E' \implies \forall u.\ \texttt{OBS\_CONGR}\ (u..E)\ (u..E')$

[OBS_CONGR_SUBST_RELAB]
$\vdash \forall E\ E'.$
$\quad \texttt{OBS\_CONGR}\ E\ E' \implies \forall rf.\ \texttt{OBS\_CONGR}\ (\texttt{relab}\ E\ rf)\ (\texttt{relab}\ E'\ rf)$

[OBS_CONGR_SUBST_RESTR]
$\vdash \forall E\ E'.\ \texttt{OBS\_CONGR}\ E\ E' \implies \forall L.\ \texttt{OBS\_CONGR}\ (\nu\ L\ E)\ (\nu\ L\ E')$

[OBS_CONGR_SUBST_SUM_L]
$\vdash \forall E\ E'.\ \texttt{OBS\_CONGR}\ E\ E' \implies \forall E''.\ \texttt{OBS\_CONGR}\ (E'' + E)\ (E'' + E')$

[OBS_CONGR_SUBST_SUM_R]
$\vdash \forall E\ E'.\ \texttt{OBS\_CONGR}\ E\ E' \implies \forall E''.\ \texttt{OBS\_CONGR}\ (E + E'')\ (E' + E'')$

[OBS_CONGR_SYM]
$\vdash \forall E\ E'.\ \texttt{OBS\_CONGR}\ E\ E' \implies \texttt{OBS\_CONGR}\ E'\ E$

[OBS_CONGR_TRANS]
$\vdash \forall E\ E'\ E''.$
$\quad \texttt{OBS\_CONGR}\ E\ E' \land \texttt{OBS\_CONGR}\ E'\ E'' \implies \texttt{OBS\_CONGR}\ E\ E''$

[OBS_CONGR_TRANS_LEFT]
$\vdash \forall E\ E'.$
$\quad \texttt{OBS\_CONGR}\ E\ E' \implies$
$\quad \forall u\ E_1.\ E\ {-u\rightarrow}\ E_1 \implies \exists E_2.\ E'\ {=u\Rightarrow}\ E_2 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2$

[OBS_CONGR_TRANS_RIGHT]
$\vdash \forall E\ E'.$
$\quad \texttt{OBS\_CONGR}\ E\ E' \implies$
$\quad \forall u\ E_2.\ E'\ {-u\rightarrow}\ E_2 \implies \exists E_1.\ E\ {=u\Rightarrow}\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2$

[OBS_CONGR_WEAK_TRANS]
$\vdash \forall E\ E'.$
$\quad \texttt{OBS\_CONGR}\ E\ E' \implies$
$\quad \forall u\ E_1.\ E\ {=u\Rightarrow}\ E_1 \implies \exists E_2.\ E'\ {=u\Rightarrow}\ E_2 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2$

[OBS_CONGR_WEAK_TRANS']
$\vdash \forall E\ E'.$
$\quad \texttt{OBS\_CONGR}\ E\ E' \implies$
$\quad \forall u\ E_2.\ E'\ {=u\Rightarrow}\ E_2 \implies \exists E_1.\ E\ {=u\Rightarrow}\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2$

[PROP6]
$\vdash \forall E\ E'.\ \texttt{WEAK\_EQUIV}\ E\ E' \implies \forall u.\ \texttt{OBS\_CONGR}\ (u..E)\ (u..E')$

[STRONG_IMP_OBS_CONGR]
$\vdash \forall E\ E'.\ \texttt{STRONG\_EQUIV}\ E\ E' \implies \texttt{OBS\_CONGR}\ E\ E'$

[WEAK_EQUIV_STABLE_IMP_CONGR]
$\vdash \forall E\ E'.$
$\quad \texttt{WEAK\_EQUIV}\ E\ E' \land \texttt{STABLE}\ E \land \texttt{STABLE}\ E' \implies \texttt{OBS\_CONGR}\ E\ E'$

9.7 ObsCongrLaws Theory

Built: 08 Dicembre 2017
Parent Theories: ObsCongr

9.7.1 Theorems

[OBS_EXPANSION_LAW]

```
⊢ ∀f n f′ m.
      (∀i. i ≤ n ⟹ Is_Prefix (f i)) ∧
      (∀j. j ≤ m ⟹ Is_Prefix (f′ j)) ⟹
      OBS_CONGR (SIGMA f n ∥ SIGMA f′ m)
        (SIGMA
           (λi. PREF_ACT (f i)..(PREF_PROC (f i) ∥ SIGMA f′ m))
           n +
         SIGMA
           (λj. PREF_ACT (f′ j)..(SIGMA f n ∥ PREF_PROC (f′ j)))
           m + ALL_SYNC f n f′ m)
```

[OBS_PAR_ASSOC]

```
⊢ ∀E E′ E″. OBS_CONGR (E ∥ E′ ∥ E″) (E ∥ (E′ ∥ E″))
```

[OBS_PAR_COMM]

```
⊢ ∀E E′. OBS_CONGR (E ∥ E′) (E′ ∥ E)
```

[OBS_PAR_IDENT_L]

```
⊢ ∀E. OBS_CONGR (nil ∥ E) E
```

[OBS_PAR_IDENT_R]

```
⊢ ∀E. OBS_CONGR (E ∥ nil) E
```

[OBS_PAR_PREF_NO_SYNCR]

```
⊢ ∀l l′.
      l ≠ COMPL l′ ⟹
      ∀E E′.
        OBS_CONGR (label l..E ∥ label l′..E′)
          (label l..(E ∥ label l′..E′) +
           label l′..(label l..E ∥ E′))
```

[OBS_PAR_PREF_SYNCR]

```
⊢ ∀l l′.
      (l = COMPL l′) ⟹
      ∀E E′.
        OBS_CONGR (label l..E ∥ label l′..E′)
          (label l..(E ∥ label l′..E′) +
           label l′..(label l..E ∥ E′) + τ..(E ∥ E′))
```

[OBS_PAR_PREF_TAU]

$$\begin{array}{l}\vdash \forall u\ E\ E'.\\ \quad \texttt{OBS\_CONGR}\ (u\texttt{..}E \parallel \tau\texttt{..}E')\ (u\texttt{..}(E \parallel \tau\texttt{..}E')\ \texttt{+}\ \tau\texttt{..}(u\texttt{..}E \parallel E'))\end{array}$$

[OBS_PAR_TAU_PREF]

$$\begin{array}{l}\vdash \forall E\ u\ E'.\\ \quad \texttt{OBS\_CONGR}\ (\tau\texttt{..}E \parallel u\texttt{..}E')\ (\tau\texttt{..}(E \parallel u\texttt{..}E')\ \texttt{+}\ u\texttt{..}(\tau\texttt{..}E \parallel E'))\end{array}$$

[OBS_PAR_TAU_TAU]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{OBS\_CONGR}\ (\tau\texttt{..}E \parallel \tau\texttt{..}E')\ (\tau\texttt{..}(E \parallel \tau\texttt{..}E')\ \texttt{+}\ \tau\texttt{..}(\tau\texttt{..}E \parallel E'))\end{array}$$

[OBS_PREF_REC_EQUIV]

$$\begin{array}{l}\vdash \forall u\ s\ v.\\ \quad \texttt{OBS\_CONGR}\ (u\texttt{..rec}\ s\ (v\texttt{..}u\texttt{..var}\ s))\ (\texttt{rec}\ s\ (u\texttt{..}v\texttt{..var}\ s))\end{array}$$

[OBS_RELAB_NIL]

$$\vdash \forall rf.\ \texttt{OBS\_CONGR}\ (\texttt{relab nil}\ rf)\ \texttt{nil}$$

[OBS_RELAB_PREFIX]

$$\begin{array}{l}\vdash \forall u\ E\ labl.\\ \quad \texttt{OBS\_CONGR}\ (\texttt{relab}\ (u\texttt{..}E)\ (\texttt{RELAB}\ labl))\\ \quad\quad (\texttt{relabel}\ (\texttt{RELAB}\ labl)\ u\texttt{..relab}\ E\ (\texttt{RELAB}\ labl))\end{array}$$

[OBS_RELAB_SUM]

$$\begin{array}{l}\vdash \forall E\ E'\ rf.\\ \quad \texttt{OBS\_CONGR}\ (\texttt{relab}\ (E\ \texttt{+}\ E')\ rf)\ (\texttt{relab}\ E\ rf\ \texttt{+}\ \texttt{relab}\ E'\ rf)\end{array}$$

[OBS_RESTR_NIL]

$$\vdash \forall L.\ \texttt{OBS\_CONGR}\ (\nu\ L\ \texttt{nil})\ \texttt{nil}$$

[OBS_RESTR_PR_LAB_NIL]

$$\begin{array}{l}\vdash \forall l\ L.\\ \quad l \in L \lor \texttt{COMPL}\ l \in L \Longrightarrow \forall E.\ \texttt{OBS\_CONGR}\ (\nu\ L\ (\texttt{label}\ l\texttt{..}E))\ \texttt{nil}\end{array}$$

[OBS_RESTR_PREFIX_LABEL]

$$\begin{array}{l}\vdash \forall l\ L.\\ \quad l \notin L \land \texttt{COMPL}\ l \notin L \Longrightarrow\\ \quad \forall E.\ \texttt{OBS\_CONGR}\ (\nu\ L\ (\texttt{label}\ l\texttt{..}E))\ (\texttt{label}\ l\texttt{..}\nu\ L\ E)\end{array}$$

[OBS_RESTR_PREFIX_TAU]

$$\vdash \forall E\ L.\ \texttt{OBS\_CONGR}\ (\nu\ L\ (\tau\texttt{..}E))\ (\tau\texttt{..}\nu\ L\ E)$$

[OBS_RESTR_SUM]

$$\vdash \forall E\ E'\ L.\ \texttt{OBS\_CONGR}\ (\nu\ L\ (E\ \texttt{+}\ E'))\ (\nu\ L\ E\ \texttt{+}\ \nu\ L\ E')$$

[OBS_SUM_ASSOC_L]

$\vdash \forall E\ E'\ E''.\ \texttt{OBS\_CONGR}\ (E\ \texttt{+}\ (E'\ \texttt{+}\ E''))\ (E\ \texttt{+}\ E'\ \texttt{+}\ E'')$

[OBS_SUM_ASSOC_R]

$\vdash \forall E\ E'\ E''.\ \texttt{OBS\_CONGR}\ (E\ \texttt{+}\ E'\ \texttt{+}\ E'')\ (E\ \texttt{+}\ (E'\ \texttt{+}\ E''))$

[OBS_SUM_COMM]

$\vdash \forall E\ E'.\ \texttt{OBS\_CONGR}\ (E\ \texttt{+}\ E')\ (E'\ \texttt{+}\ E)$

[OBS_SUM_IDEMP]

$\vdash \forall E.\ \texttt{OBS\_CONGR}\ (E\ \texttt{+}\ E)\ E$

[OBS_SUM_IDENT_L]

$\vdash \forall E.\ \texttt{OBS\_CONGR}\ (\texttt{nil}\ \texttt{+}\ E)\ E$

[OBS_SUM_IDENT_R]

$\vdash \forall E.\ \texttt{OBS\_CONGR}\ (E\ \texttt{+}\ \texttt{nil})\ E$

[OBS_UNFOLDING]

$\vdash \forall X\ E.\ \texttt{OBS\_CONGR}\ (\texttt{rec}\ X\ E)\ (\texttt{CCS\_Subst}\ E\ (\texttt{rec}\ X\ E)\ X)$

[TAU1]

$\vdash \forall u\ E.\ \texttt{OBS\_CONGR}\ (u..\tau..E)\ (u..E)$

[TAU2]

$\vdash \forall E.\ \texttt{OBS\_CONGR}\ (E\ \texttt{+}\ \tau..E)\ (\tau..E)$

[TAU3]

$\vdash \forall u\ E\ E'.\ \texttt{OBS\_CONGR}\ (u..(E\ \texttt{+}\ \tau..E')\ \texttt{+}\ u..E')\ (u..(E\ \texttt{+}\ \tau..E'))$

[WEAK_TAU1]

$\vdash \forall u\ E.\ \texttt{WEAK\_EQUIV}\ (u..\tau..E)\ (u..E)$

[WEAK_TAU2]

$\vdash \forall E.\ \texttt{WEAK\_EQUIV}\ (E\ \texttt{+}\ \tau..E)\ (\tau..E)$

[WEAK_TAU3]

$\vdash \forall u\ E\ E'.\ \texttt{WEAK\_EQUIV}\ (u..(E\ \texttt{+}\ \tau..E')\ \texttt{+}\ u..E')\ (u..(E\ \texttt{+}\ \tau..E'))$

9.8 Congruence Theory

Built: 08 Dicembre 2017
Parent Theories: ObsCongrLaws, string

9.8.1 Definitions

[CC_def]

$$\vdash \forall R.\ \texttt{CC}\ R = (\lambda g\ h.\ \forall c.\ \texttt{CONTEXT}\ c \Longrightarrow R\ (c\ g)\ (c\ h))$$

[congruence1_def]

$$\vdash \forall R.\ \texttt{congruence1}\ R \iff \texttt{equivalence}\ R \land \texttt{precongruence1}\ R$$

[congruence_def]

$$\vdash \forall R.\ \texttt{congruence}\ R \iff \texttt{equivalence}\ R \land \texttt{precongruence}\ R$$

[CONTEXT_def]

$$\begin{array}{l}
\vdash \texttt{CONTEXT} = \\
\quad (\lambda a_0. \\
\qquad \forall \mathit{CONTEXT}'. \\
\qquad\quad (\forall a_0. \\
\qquad\qquad (a_0 = (\lambda t.\ t)) \lor (\exists p.\ a_0 = (\lambda t.\ p)) \lor \\
\qquad\qquad (\exists a\ e.\ (a_0 = (\lambda t.\ a\,..\,e\ t)) \land \mathit{CONTEXT}'\ e) \lor \\
\qquad\qquad (\exists e_1\ e_2. \\
\qquad\qquad\quad (a_0 = (\lambda t.\ e_1\ t + e_2\ t)) \land \mathit{CONTEXT}'\ e_1 \land \\
\qquad\qquad\quad \mathit{CONTEXT}'\ e_2) \lor \\
\qquad\qquad (\exists e_1\ e_2. \\
\qquad\qquad\quad (a_0 = (\lambda t.\ e_1\ t \parallel e_2\ t)) \land \mathit{CONTEXT}'\ e_1 \land \\
\qquad\qquad\quad \mathit{CONTEXT}'\ e_2) \lor \\
\qquad\qquad (\exists L\ e.\ (a_0 = (\lambda t.\ \nu\ L\ (e\ t))) \land \mathit{CONTEXT}'\ e) \lor \\
\qquad\qquad (\exists \mathit{rf}\ e.\ (a_0 = (\lambda t.\ \texttt{relab}\ (e\ t)\ \mathit{rf})) \land \mathit{CONTEXT}'\ e) \Longrightarrow \\
\qquad\qquad \mathit{CONTEXT}'\ a_0) \Longrightarrow \\
\qquad\quad \mathit{CONTEXT}'\ a_0)
\end{array}$$

[GCC_def]

$$\vdash \forall R.\ \texttt{GCC}\ R = (\lambda g\ h.\ \forall c.\ \texttt{GCONTEXT}\ c \Longrightarrow R\ (c\ g)\ (c\ h))$$

[GCONTEXT_def]

$$\begin{array}{l}
\vdash \texttt{GCONTEXT} = \\
\quad (\lambda a_0. \\
\qquad \forall \mathit{GCONTEXT}'. \\
\qquad\quad (\forall a_0. \\
\qquad\qquad (a_0 = (\lambda t.\ t)) \lor (\exists p.\ a_0 = (\lambda t.\ p)) \lor \\
\qquad\qquad (\exists a\ e.\ (a_0 = (\lambda t.\ a\,..\,e\ t)) \land \mathit{GCONTEXT}'\ e) \lor \\
\qquad\qquad (\exists a_1\ a_2\ e_1\ e_2. \\
\qquad\qquad\quad (a_0 = (\lambda t.\ a_1\,..\,e_1\ t + a_2\,..\,e_2\ t)) \land \mathit{GCONTEXT}'\ e_1 \land \\
\qquad\qquad\quad \mathit{GCONTEXT}'\ e_2) \lor \\
\qquad\qquad (\exists e_1\ e_2. \\
\qquad\qquad\quad (a_0 = (\lambda t.\ e_1\ t \parallel e_2\ t)) \land \mathit{GCONTEXT}'\ e_1 \land \\
\qquad\qquad\quad \mathit{GCONTEXT}'\ e_2) \lor \\
\qquad\qquad (\exists L\ e.\ (a_0 = (\lambda t.\ \nu\ L\ (e\ t))) \land \mathit{GCONTEXT}'\ e) \lor \\
\qquad\qquad (\exists \mathit{rf}\ e.\ (a_0 = (\lambda t.\ \texttt{relab}\ (e\ t)\ \mathit{rf})) \land \mathit{GCONTEXT}'\ e) \Longrightarrow \\
\qquad\qquad \mathit{GCONTEXT}'\ a_0) \Longrightarrow \\
\qquad\quad \mathit{GCONTEXT}'\ a_0)
\end{array}$$

[GSEQ_def]

```
⊢ GSEQ =
  (λ a0.
     ∀ GSEQ'.
       (∀ a0.
          (a0 = (λ t. t)) ∨ (∃ p. a0 = (λ t. p)) ∨
          (∃ a e. (a0 = (λ t. a..e t)) ∧ GSEQ' e) ∨
          (∃ a1 a2 e1 e2.
             (a0 = (λ t. a1..e1 t + a2..e2 t)) ∧ GSEQ' e1 ∧
             GSEQ' e2) ⟹
          GSEQ' a0) ⟹
       GSEQ' a0)
```

[OH_CONTEXT_def]

```
⊢ OH_CONTEXT =
  (λ a0.
     ∀ OH_CONTEXT'.
       (∀ a0.
          (a0 = (λ t. t)) ∨
          (∃ a c. (a0 = (λ t. a..c t)) ∧ OH_CONTEXT' c) ∨
          (∃ x c. (a0 = (λ t. c t + x)) ∧ OH_CONTEXT' c) ∨
          (∃ x c. (a0 = (λ t. x + c t)) ∧ OH_CONTEXT' c) ∨
          (∃ x c. (a0 = (λ t. c t ‖ x)) ∧ OH_CONTEXT' c) ∨
          (∃ x c. (a0 = (λ t. x ‖ c t)) ∧ OH_CONTEXT' c) ∨
          (∃ L c. (a0 = (λ t. ν L (c t))) ∧ OH_CONTEXT' c) ∨
          (∃ rf c.
             (a0 = (λ t. relab (c t) rf)) ∧ OH_CONTEXT' c) ⟹
          OH_CONTEXT' a0) ⟹
       OH_CONTEXT' a0)
```

[precongruence1_def]

```
⊢ ∀ R.
    precongruence1 R ⟺
    ∀ x y ctx. GCONTEXT ctx ⟹ R x y ⟹ R (ctx x) (ctx y)
```

[precongruence_def]

```
⊢ ∀ R.
    precongruence R ⟺
    ∀ x y ctx. CONTEXT ctx ⟹ R x y ⟹ R (ctx x) (ctx y)
```

[SEQ_def]

```
⊢ SEQ =
  (λ a0.
     ∀ SEQ'.
       (∀ a0.
          (a0 = (λ t. t)) ∨ (∃ p. a0 = (λ t. p)) ∨
          (∃a e. (a0 = (λt. a..e t)) ∧ SEQ' e) ∨
          (∃e1 e2.
             (a0 = (λt. e1 t + e2 t)) ∧ SEQ' e1 ∧ SEQ' e2) ⟹
          SEQ' a0) ⟹
       SEQ' a0)
```

[SG_def]

```
 ⊢ SG =
   (λa0.
      ∀SG'.
       (∀a0.
          (∃p. a0 = (λt. p)) ∨
          (∃l e. (a0 = (λt. label l..e t)) ∧ CONTEXT e) ∨
          (∃a e. (a0 = (λt. a..e t)) ∧ SG' e) ∨
          (∃e1 e2.
             (a0 = (λt. e1 t + e2 t)) ∧ SG' e1 ∧ SG' e2) ∨
          (∃e1 e2.
             (a0 = (λt. e1 t ∥ e2 t)) ∧ SG' e1 ∧ SG' e2) ∨
          (∃L e. (a0 = (λt. ν L (e t))) ∧ SG' e) ∨
          (∃rf e. (a0 = (λt. relab (e t) rf)) ∧ SG' e) ⟹
          SG' a0) ⟹
       SG' a0)
```

[weakly_guarded1_def]

```
 ⊢ ∀E.
     weakly_guarded1 E ⟺
     ∀X. X ∈ FV E ⟹ ∀e. CONTEXT e ∧ (e (var X) = E) ⟹ WG e
```

[weakly_guarded_def]

```
 ⊢ ∀Es. weakly_guarded Es ⟺ EVERY weakly_guarded1 Es
```

[WG_def]

```
 ⊢ WG =
   (λa0.
      ∀WG'.
       (∀a0.
          (∃p. a0 = (λt. p)) ∨
          (∃a e. (a0 = (λt. a..e t)) ∧ CONTEXT e) ∨
          (∃e1 e2.
             (a0 = (λt. e1 t + e2 t)) ∧ WG' e1 ∧ WG' e2) ∨
          (∃e1 e2.
             (a0 = (λt. e1 t ∥ e2 t)) ∧ WG' e1 ∧ WG' e2) ∨
          (∃L e. (a0 = (λt. ν L (e t))) ∧ WG' e) ∨
          (∃rf e. (a0 = (λt. relab (e t) rf)) ∧ WG' e) ⟹
          WG' a0) ⟹
       WG' a0)
```

[WGS_def]

$$\begin{array}{l}
\vdash \texttt{WGS} = \\
\quad (\lambda\, a_0 . \\
\quad\quad \forall\, \mathit{WGS}' . \\
\quad\quad\quad (\forall\, a_0 . \\
\quad\quad\quad\quad (\exists\, p .\ a_0 = (\lambda\, t .\ p)) \lor \\
\quad\quad\quad\quad (\exists\, a\ e .\ (a_0 = (\lambda\, t .\ a \texttt{..} e\ t)) \land \texttt{GCONTEXT}\ e) \lor \\
\quad\quad\quad\quad (\exists\, a_1\ a_2\ e_1\ e_2 . \\
\quad\quad\quad\quad\quad (a_0 = (\lambda\, t .\ a_1 \texttt{..} e_1\ t + a_2 \texttt{..} e_2\ t)) \land \texttt{GCONTEXT}\ e_1 \land \\
\quad\quad\quad\quad\quad \texttt{GCONTEXT}\ e_2) \lor \\
\quad\quad\quad\quad (\exists\, e_1\ e_2 . \\
\quad\quad\quad\quad\quad (a_0 = (\lambda\, t .\ e_1\ t \parallel e_2\ t)) \land \mathit{WGS}'\ e_1 \land \mathit{WGS}'\ e_2) \lor \\
\quad\quad\quad\quad (\exists\, L\ e .\ (a_0 = (\lambda\, t .\ \nu\ L\ (e\ t))) \land \mathit{WGS}'\ e) \lor \\
\quad\quad\quad\quad (\exists\, \mathit{rf}\ e .\ (a_0 = (\lambda\, t .\ \texttt{relab}\ (e\ t)\ \mathit{rf})) \land \mathit{WGS}'\ e) \Longrightarrow \\
\quad\quad\quad\quad \mathit{WGS}'\ a_0) \Longrightarrow \\
\quad\quad\quad \mathit{WGS}'\ a_0)
\end{array}$$

9.8.2 Theorems

[CC_congruence]

$\vdash \forall\, R .\ \texttt{equivalence}\ R \Longrightarrow \texttt{congruence}\ (\texttt{CC}\ R)$

[CC_is_coarsest]

$\vdash \forall\, R\ R' .\ \texttt{congruence}\ R' \land R'\ \texttt{RSUBSET}\ R \Longrightarrow R'\ \texttt{RSUBSET}\ \texttt{CC}\ R$

[CC_is_coarsest']

$\vdash \forall\, R\ R' .\ \texttt{precongruence}\ R' \land R'\ \texttt{RSUBSET}\ R \Longrightarrow R'\ \texttt{RSUBSET}\ \texttt{CC}\ R$

[CC_is_finer]

$\vdash \forall\, R .\ \texttt{CC}\ R\ \texttt{RSUBSET}\ R$

[CC_precongruence]

$\vdash \forall\, R .\ \texttt{precongruence}\ (\texttt{CC}\ R)$

[CONTEXT1]

$\vdash \texttt{CONTEXT}\ (\lambda\, t .\ t)$

[CONTEXT2]

$\vdash \forall\, p .\ \texttt{CONTEXT}\ (\lambda\, t .\ p)$

[CONTEXT3]

$\vdash \forall\, a\ e .\ \texttt{CONTEXT}\ e \Longrightarrow \texttt{CONTEXT}\ (\lambda\, t .\ a \texttt{..} e\ t)$

[CONTEXT3a]

$\vdash \forall\, a .\ \texttt{CONTEXT}\ (\lambda\, t .\ a \texttt{..} t)$

[CONTEXT4]

$$\vdash \forall e_1\ e_2.\ \mathtt{CONTEXT}\ e_1 \land \mathtt{CONTEXT}\ e_2 \implies \mathtt{CONTEXT}\ (\lambda t.\ e_1\ t\ \mathtt{+}\ e_2\ t)$$

[CONTEXT5]

$$\vdash \forall e_1\ e_2.\ \mathtt{CONTEXT}\ e_1 \land \mathtt{CONTEXT}\ e_2 \implies \mathtt{CONTEXT}\ (\lambda t.\ e_1\ t \parallel e_2\ t)$$

[CONTEXT6]

$$\vdash \forall L\ e.\ \mathtt{CONTEXT}\ e \implies \mathtt{CONTEXT}\ (\lambda t.\ \nu\ L\ (e\ t))$$

[CONTEXT7]

$$\vdash \forall rf\ e.\ \mathtt{CONTEXT}\ e \implies \mathtt{CONTEXT}\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)$$

[CONTEXT_cases]

$$\begin{array}{l}
\vdash \forall a_0.\\
\quad \mathtt{CONTEXT}\ a_0 \iff\\
\quad (a_0 = (\lambda t.\ t)) \lor (\exists p.\ a_0 = (\lambda t.\ p)) \lor\\
\quad (\exists a\ e.\ (a_0 = (\lambda t.\ a..e\ t)) \land \mathtt{CONTEXT}\ e) \lor\\
\quad (\exists e_1\ e_2.\\
\qquad (a_0 = (\lambda t.\ e_1\ t\ \mathtt{+}\ e_2\ t)) \land \mathtt{CONTEXT}\ e_1 \land \mathtt{CONTEXT}\ e_2) \lor\\
\quad (\exists e_1\ e_2.\\
\qquad (a_0 = (\lambda t.\ e_1\ t \parallel e_2\ t)) \land \mathtt{CONTEXT}\ e_1 \land \mathtt{CONTEXT}\ e_2) \lor\\
\quad (\exists L\ e.\ (a_0 = (\lambda t.\ \nu\ L\ (e\ t))) \land \mathtt{CONTEXT}\ e) \lor\\
\quad \exists rf\ e.\ (a_0 = (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)) \land \mathtt{CONTEXT}\ e
\end{array}$$

[CONTEXT_combin]

$$\vdash \forall c_1\ c_2.\ \mathtt{CONTEXT}\ c_1 \land \mathtt{CONTEXT}\ c_2 \implies \mathtt{CONTEXT}\ (c_1 \circ c_2)$$

[CONTEXT_ind]

$$\begin{array}{l}
\vdash \forall \mathit{CONTEXT}'.\\
\quad \mathit{CONTEXT}'\ (\lambda t.\ t) \land (\forall p.\ \mathit{CONTEXT}'\ (\lambda t.\ p)) \land\\
\quad (\forall a\ e.\ \mathit{CONTEXT}'\ e \implies \mathit{CONTEXT}'\ (\lambda t.\ a..e\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathit{CONTEXT}'\ e_1 \land \mathit{CONTEXT}'\ e_2 \implies\\
\qquad \mathit{CONTEXT}'\ (\lambda t.\ e_1\ t\ \mathtt{+}\ e_2\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathit{CONTEXT}'\ e_1 \land \mathit{CONTEXT}'\ e_2 \implies\\
\qquad \mathit{CONTEXT}'\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\quad (\forall L\ e.\ \mathit{CONTEXT}'\ e \implies \mathit{CONTEXT}'\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad (\forall rf\ e.\ \mathit{CONTEXT}'\ e \implies \mathit{CONTEXT}'\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)) \implies\\
\quad \forall a_0.\ \mathtt{CONTEXT}\ a_0 \implies \mathit{CONTEXT}'\ a_0
\end{array}$$

[CONTEXT_rules]

$$\begin{array}{l}
\vdash \mathtt{CONTEXT}\ (\lambda t.\ t) \land (\forall p.\ \mathtt{CONTEXT}\ (\lambda t.\ p)) \land\\
\quad (\forall a\ e.\ \mathtt{CONTEXT}\ e \implies \mathtt{CONTEXT}\ (\lambda t.\ a..e\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathtt{CONTEXT}\ e_1 \land \mathtt{CONTEXT}\ e_2 \implies \mathtt{CONTEXT}\ (\lambda t.\ e_1\ t\ \mathtt{+}\ e_2\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathtt{CONTEXT}\ e_1 \land \mathtt{CONTEXT}\ e_2 \implies \mathtt{CONTEXT}\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\quad (\forall L\ e.\ \mathtt{CONTEXT}\ e \implies \mathtt{CONTEXT}\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad \forall rf\ e.\ \mathtt{CONTEXT}\ e \implies \mathtt{CONTEXT}\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)
\end{array}$$

[CONTEXT_strongind]

$$\begin{array}{l}
\vdash \forall \mathit{CONTEXT}'.\\
\qquad \mathit{CONTEXT}'\ (\lambda t.\ t) \land (\forall p.\ \mathit{CONTEXT}'\ (\lambda t.\ p)) \land\\
\qquad (\forall a\ e.\ \texttt{CONTEXT}\ e \land \mathit{CONTEXT}'\ e \implies \mathit{CONTEXT}'\ (\lambda t.\ a\texttt{..}e\ t)) \land\\
\qquad (\forall e_1\ e_2.\\
\qquad\quad \texttt{CONTEXT}\ e_1 \land \mathit{CONTEXT}'\ e_1 \land \texttt{CONTEXT}\ e_2 \land \mathit{CONTEXT}'\ e_2 \implies\\
\qquad\quad \mathit{CONTEXT}'\ (\lambda t.\ e_1\ t\ \texttt{+}\ e_2\ t)) \land\\
\qquad (\forall e_1\ e_2.\\
\qquad\quad \texttt{CONTEXT}\ e_1 \land \mathit{CONTEXT}'\ e_1 \land \texttt{CONTEXT}\ e_2 \land \mathit{CONTEXT}'\ e_2 \implies\\
\qquad\quad \mathit{CONTEXT}'\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\qquad (\forall L\ e.\ \texttt{CONTEXT}\ e \land \mathit{CONTEXT}'\ e \implies \mathit{CONTEXT}'\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\qquad (\forall \mathit{rf}\ e.\\
\qquad\quad \texttt{CONTEXT}\ e \land \mathit{CONTEXT}'\ e \implies\\
\qquad\quad \mathit{CONTEXT}'\ (\lambda t.\ \texttt{relab}\ (e\ t)\ \mathit{rf})) \implies\\
\qquad \forall a_0.\ \texttt{CONTEXT}\ a_0 \implies \mathit{CONTEXT}'\ a_0
\end{array}$$

[CONTEXT_WG_combin]

$$\vdash \forall c\ e.\ \texttt{CONTEXT}\ c \land \texttt{WG}\ e \implies \texttt{WG}\ (c \circ e)$$

[GCONTEXT1]

$$\vdash \texttt{GCONTEXT}\ (\lambda t.\ t)$$

[GCONTEXT2]

$$\vdash \forall p.\ \texttt{GCONTEXT}\ (\lambda t.\ p)$$

[GCONTEXT3]

$$\vdash \forall a\ e.\ \texttt{GCONTEXT}\ e \implies \texttt{GCONTEXT}\ (\lambda t.\ a\texttt{..}e\ t)$$

[GCONTEXT3a]

$$\vdash \forall a.\ \texttt{GCONTEXT}\ (\lambda t.\ a\texttt{..}t)$$

[GCONTEXT4]

$$\begin{array}{l}
\vdash \forall a_1\ a_2\ e_1\ e_2.\\
\qquad \texttt{GCONTEXT}\ e_1 \land \texttt{GCONTEXT}\ e_2 \implies\\
\qquad \texttt{GCONTEXT}\ (\lambda t.\ a_1\texttt{..}e_1\ t\ \texttt{+}\ a_2\texttt{..}e_2\ t)
\end{array}$$

[GCONTEXT5]

$$\begin{array}{l}
\vdash \forall e_1\ e_2.\\
\qquad \texttt{GCONTEXT}\ e_1 \land \texttt{GCONTEXT}\ e_2 \implies \texttt{GCONTEXT}\ (\lambda t.\ e_1\ t \parallel e_2\ t)
\end{array}$$

[GCONTEXT6]

$$\vdash \forall L\ e.\ \texttt{GCONTEXT}\ e \implies \texttt{GCONTEXT}\ (\lambda t.\ \nu\ L\ (e\ t))$$

[GCONTEXT7]

$$\vdash \forall \mathit{rf}\ e.\ \texttt{GCONTEXT}\ e \implies \texttt{GCONTEXT}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ \mathit{rf})$$

[GCONTEXT_cases]

$$\begin{array}{l}
\vdash \forall a_0 .\\
\qquad \mathtt{GCONTEXT}\ a_0 \iff\\
\qquad (a_0 = (\lambda t.\ t)) \lor (\exists p.\ a_0 = (\lambda t.\ p)) \lor\\
\qquad (\exists a\ e.\ (a_0 = (\lambda t.\ a..e\ t)) \land \mathtt{GCONTEXT}\ e) \lor\\
\qquad (\exists a_1\ a_2\ e_1\ e_2 .\\
\qquad\quad (a_0 = (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)) \land \mathtt{GCONTEXT}\ e_1 \land\\
\qquad\quad \mathtt{GCONTEXT}\ e_2) \lor\\
\qquad (\exists e_1\ e_2 .\\
\qquad\quad (a_0 = (\lambda t.\ e_1\ t \parallel e_2\ t)) \land \mathtt{GCONTEXT}\ e_1 \land \mathtt{GCONTEXT}\ e_2) \lor\\
\qquad (\exists L\ e.\ (a_0 = (\lambda t.\ \nu\ L\ (e\ t))) \land \mathtt{GCONTEXT}\ e) \lor\\
\qquad \exists \mathit{rf}\ e.\ (a_0 = (\lambda t.\ \mathtt{relab}\ (e\ t)\ \mathit{rf})) \land \mathtt{GCONTEXT}\ e
\end{array}$$

[GCONTEXT_combin]

$$\vdash \forall c_1\ c_2 .\ \mathtt{GCONTEXT}\ c_1 \land \mathtt{GCONTEXT}\ c_2 \implies \mathtt{GCONTEXT}\ (c_1 \circ c_2)$$

[GCONTEXT_ind]

$$\begin{array}{l}
\vdash \forall \mathit{GCONTEXT}' .\\
\qquad \mathit{GCONTEXT}'\ (\lambda t.\ t) \land (\forall p.\ \mathit{GCONTEXT}'\ (\lambda t.\ p)) \land\\
\qquad (\forall a\ e.\ \mathit{GCONTEXT}'\ e \implies \mathit{GCONTEXT}'\ (\lambda t.\ a..e\ t)) \land\\
\qquad (\forall a_1\ a_2\ e_1\ e_2 .\\
\qquad\quad \mathit{GCONTEXT}'\ e_1 \land \mathit{GCONTEXT}'\ e_2 \implies\\
\qquad\quad \mathit{GCONTEXT}'\ (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)) \land\\
\qquad (\forall e_1\ e_2 .\\
\qquad\quad \mathit{GCONTEXT}'\ e_1 \land \mathit{GCONTEXT}'\ e_2 \implies\\
\qquad\quad \mathit{GCONTEXT}'\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\qquad (\forall L\ e.\ \mathit{GCONTEXT}'\ e \implies \mathit{GCONTEXT}'\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\qquad (\forall \mathit{rf}\ e.\ \mathit{GCONTEXT}'\ e \implies \mathit{GCONTEXT}'\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ \mathit{rf})) \implies\\
\qquad \forall a_0 .\ \mathtt{GCONTEXT}\ a_0 \implies \mathit{GCONTEXT}'\ a_0
\end{array}$$

[GCONTEXT_IS_CONTEXT]

$$\vdash \forall c.\ \mathtt{GCONTEXT}\ c \implies \mathtt{CONTEXT}\ c$$

[GCONTEXT_rules]

$$\begin{array}{l}
\vdash \mathtt{GCONTEXT}\ (\lambda t.\ t) \land (\forall p.\ \mathtt{GCONTEXT}\ (\lambda t.\ p)) \land\\
\quad (\forall a\ e.\ \mathtt{GCONTEXT}\ e \implies \mathtt{GCONTEXT}\ (\lambda t.\ a..e\ t)) \land\\
\quad (\forall a_1\ a_2\ e_1\ e_2 .\\
\qquad \mathtt{GCONTEXT}\ e_1 \land \mathtt{GCONTEXT}\ e_2 \implies\\
\qquad \mathtt{GCONTEXT}\ (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)) \land\\
\quad (\forall e_1\ e_2 .\\
\qquad \mathtt{GCONTEXT}\ e_1 \land \mathtt{GCONTEXT}\ e_2 \implies \mathtt{GCONTEXT}\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\quad (\forall L\ e.\ \mathtt{GCONTEXT}\ e \implies \mathtt{GCONTEXT}\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad \forall \mathit{rf}\ e.\ \mathtt{GCONTEXT}\ e \implies \mathtt{GCONTEXT}\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ \mathit{rf})
\end{array}$$

[GCONTEXT_strongind]

$$\begin{array}{l}
\vdash \forall \mathit{GCONTEXT}'.\\
\quad\quad \mathit{GCONTEXT}'\ (\lambda t.\ t) \land (\forall p.\ \mathit{GCONTEXT}'\ (\lambda t.\ p)) \land\\
\quad\quad (\forall a\ e.\ \texttt{GCONTEXT}\ e \land \mathit{GCONTEXT}'\ e \Longrightarrow \mathit{GCONTEXT}'\ (\lambda t.\ a\texttt{..}e\ t)) \land\\
\quad\quad (\forall a_1\ a_2\ e_1\ e_2.\\
\quad\quad\quad \texttt{GCONTEXT}\ e_1 \land \mathit{GCONTEXT}'\ e_1 \land \texttt{GCONTEXT}\ e_2 \land\\
\quad\quad\quad \mathit{GCONTEXT}'\ e_2 \Longrightarrow\\
\quad\quad\quad \mathit{GCONTEXT}'\ (\lambda t.\ a_1\texttt{..}e_1\ t\ \texttt{+}\ a_2\texttt{..}e_2\ t)) \land\\
\quad\quad (\forall e_1\ e_2.\\
\quad\quad\quad \texttt{GCONTEXT}\ e_1 \land \mathit{GCONTEXT}'\ e_1 \land \texttt{GCONTEXT}\ e_2 \land\\
\quad\quad\quad \mathit{GCONTEXT}'\ e_2 \Longrightarrow\\
\quad\quad\quad \mathit{GCONTEXT}'\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\quad\quad (\forall L\ e.\\
\quad\quad\quad \texttt{GCONTEXT}\ e \land \mathit{GCONTEXT}'\ e \Longrightarrow \mathit{GCONTEXT}'\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad\quad (\forall \mathit{rf}\ e.\\
\quad\quad\quad \texttt{GCONTEXT}\ e \land \mathit{GCONTEXT}'\ e \Longrightarrow\\
\quad\quad\quad \mathit{GCONTEXT}'\ (\lambda t.\ \texttt{relab}\ (e\ t)\ \mathit{rf})) \Longrightarrow\\
\quad\quad \forall a_0.\ \texttt{GCONTEXT}\ a_0 \Longrightarrow \mathit{GCONTEXT}'\ a_0
\end{array}$$

[GCONTEXT_WGS_combin]

$$\vdash \forall c\ e.\ \texttt{GCONTEXT}\ c \land \texttt{WGS}\ e \Longrightarrow \texttt{WGS}\ (c \circ e)$$

[GSEQ1]

$$\vdash \texttt{GSEQ}\ (\lambda t.\ t)$$

[GSEQ2]

$$\vdash \forall p.\ \texttt{GSEQ}\ (\lambda t.\ p)$$

[GSEQ3]

$$\vdash \forall a\ e.\ \texttt{GSEQ}\ e \Longrightarrow \texttt{GSEQ}\ (\lambda t.\ a\texttt{..}e\ t)$$

[GSEQ3a]

$$\vdash \forall a.\ \texttt{GSEQ}\ (\lambda t.\ a\texttt{..}t)$$

[GSEQ4]

$$\begin{array}{l}
\vdash \forall a_1\ a_2\ e_1\ e_2.\\
\quad\quad \texttt{GSEQ}\ e_1 \land \texttt{GSEQ}\ e_2 \Longrightarrow \texttt{GSEQ}\ (\lambda t.\ a_1\texttt{..}e_1\ t\ \texttt{+}\ a_2\texttt{..}e_2\ t)
\end{array}$$

[GSEQ_cases]

$$\begin{array}{l}
\vdash \forall a_0.\\
\quad\quad \texttt{GSEQ}\ a_0 \iff\\
\quad\quad (a_0\ \texttt{=}\ (\lambda t.\ t)) \lor (\exists p.\ a_0\ \texttt{=}\ (\lambda t.\ p)) \lor\\
\quad\quad (\exists a\ e.\ (a_0\ \texttt{=}\ (\lambda t.\ a\texttt{..}e\ t)) \land \texttt{GSEQ}\ e) \lor\\
\quad\quad \exists a_1\ a_2\ e_1\ e_2.\\
\quad\quad\quad (a_0\ \texttt{=}\ (\lambda t.\ a_1\texttt{..}e_1\ t\ \texttt{+}\ a_2\texttt{..}e_2\ t)) \land \texttt{GSEQ}\ e_1 \land \texttt{GSEQ}\ e_2
\end{array}$$

[GSEQ_combin]

$$\vdash \forall E.\ \texttt{GSEQ}\ E \Longrightarrow \forall E'.\ \texttt{GSEQ}\ E' \Longrightarrow \texttt{GSEQ}\ (E \circ E')$$

[GSEQ_ind]

$$\begin{array}{l}
\vdash \forall \mathit{GSEQ}'.\\
\quad \mathit{GSEQ}'\ (\lambda t.\ t) \land (\forall p.\ \mathit{GSEQ}'\ (\lambda t.\ p)) \land\\
\quad (\forall a\ e.\ \mathit{GSEQ}'\ e \Longrightarrow \mathit{GSEQ}'\ (\lambda t.\ a..e\ t)) \land\\
\quad (\forall a_1\ a_2\ e_1\ e_2.\\
\qquad \mathit{GSEQ}'\ e_1 \land \mathit{GSEQ}'\ e_2 \Longrightarrow \mathit{GSEQ}'\ (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)) \Longrightarrow\\
\quad \forall a_0.\ \texttt{GSEQ}\ a_0 \Longrightarrow \mathit{GSEQ}'\ a_0
\end{array}$$

[GSEQ_IS_CONTEXT]

$$\vdash \forall e.\ \texttt{GSEQ}\ e \Longrightarrow \texttt{CONTEXT}\ e$$

[GSEQ_rules]

$$\begin{array}{l}
\vdash \texttt{GSEQ}\ (\lambda t.\ t) \land (\forall p.\ \texttt{GSEQ}\ (\lambda t.\ p)) \land\\
\quad (\forall a\ e.\ \texttt{GSEQ}\ e \Longrightarrow \texttt{GSEQ}\ (\lambda t.\ a..e\ t)) \land\\
\quad \forall a_1\ a_2\ e_1\ e_2.\\
\qquad \texttt{GSEQ}\ e_1 \land \texttt{GSEQ}\ e_2 \Longrightarrow \texttt{GSEQ}\ (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)
\end{array}$$

[GSEQ_strongind]

$$\begin{array}{l}
\vdash \forall \mathit{GSEQ}'.\\
\quad \mathit{GSEQ}'\ (\lambda t.\ t) \land (\forall p.\ \mathit{GSEQ}'\ (\lambda t.\ p)) \land\\
\quad (\forall a\ e.\ \texttt{GSEQ}\ e \land \mathit{GSEQ}'\ e \Longrightarrow \mathit{GSEQ}'\ (\lambda t.\ a..e\ t)) \land\\
\quad (\forall a_1\ a_2\ e_1\ e_2.\\
\qquad \texttt{GSEQ}\ e_1 \land \mathit{GSEQ}'\ e_1 \land \texttt{GSEQ}\ e_2 \land \mathit{GSEQ}'\ e_2 \Longrightarrow\\
\qquad \mathit{GSEQ}'\ (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)) \Longrightarrow\\
\quad \forall a_0.\ \texttt{GSEQ}\ a_0 \Longrightarrow \mathit{GSEQ}'\ a_0
\end{array}$$

[OBS_CONGR_congruence]

$$\vdash \texttt{congruence OBS\_CONGR}$$

[OBS_CONGR_SUBST_CONTEXT]

$$\vdash \forall P\ Q.\ \texttt{OBS\_CONGR}\ P\ Q \Longrightarrow \forall E.\ \texttt{CONTEXT}\ E \Longrightarrow \texttt{OBS\_CONGR}\ (E\ P)\ (E\ Q)$$

[OBS_CONGR_SUBST_SEQ]

$$\vdash \forall P\ Q.\ \texttt{OBS\_CONGR}\ P\ Q \Longrightarrow \forall E.\ \texttt{SEQ}\ E \Longrightarrow \texttt{OBS\_CONGR}\ (E\ P)\ (E\ Q)$$

[OH_CONTEXT1]

$$\vdash \texttt{OH\_CONTEXT}\ (\lambda t.\ t)$$

[OH_CONTEXT2]

$$\vdash \forall a\ c.\ \texttt{OH\_CONTEXT}\ c \Longrightarrow \texttt{OH\_CONTEXT}\ (\lambda t.\ a..c\ t)$$

[OH_CONTEXT3]

$\vdash \forall x\ \ c.\ \texttt{OH\_CONTEXT}\ c \implies \texttt{OH\_CONTEXT}\ (\lambda t.\ c\ t\ \texttt{+}\ x)$

[OH_CONTEXT4]

$\vdash \forall x\ \ c.\ \texttt{OH\_CONTEXT}\ c \implies \texttt{OH\_CONTEXT}\ (\lambda t.\ x\ \texttt{+}\ c\ t)$

[OH_CONTEXT5]

$\vdash \forall x\ \ c.\ \texttt{OH\_CONTEXT}\ c \implies \texttt{OH\_CONTEXT}\ (\lambda t.\ c\ t\ \parallel\ x)$

[OH_CONTEXT6]

$\vdash \forall x\ \ c.\ \texttt{OH\_CONTEXT}\ c \implies \texttt{OH\_CONTEXT}\ (\lambda t.\ x\ \parallel\ c\ t)$

[OH_CONTEXT7]

$\vdash \forall L\ \ c.\ \texttt{OH\_CONTEXT}\ c \implies \texttt{OH\_CONTEXT}\ (\lambda t.\ \nu\ L\ (c\ t))$

[OH_CONTEXT8]

$\vdash \forall rf\ \ c.\ \texttt{OH\_CONTEXT}\ c \implies \texttt{OH\_CONTEXT}\ (\lambda t.\ \texttt{relab}\ (c\ t)\ rf)$

[OH_CONTEXT_cases]

$$\begin{array}{l}
\vdash \forall a_0. \\
\quad \texttt{OH\_CONTEXT}\ a_0 \iff \\
\quad (a_0\ \texttt{=}\ (\lambda t.\ t)) \lor \\
\quad (\exists a\ \ c.\ (a_0\ \texttt{=}\ (\lambda t.\ a..c\ t)) \land \texttt{OH\_CONTEXT}\ c) \lor \\
\quad (\exists x\ \ c.\ (a_0\ \texttt{=}\ (\lambda t.\ c\ t\ \texttt{+}\ x)) \land \texttt{OH\_CONTEXT}\ c) \lor \\
\quad (\exists x\ \ c.\ (a_0\ \texttt{=}\ (\lambda t.\ x\ \texttt{+}\ c\ t)) \land \texttt{OH\_CONTEXT}\ c) \lor \\
\quad (\exists x\ \ c.\ (a_0\ \texttt{=}\ (\lambda t.\ c\ t\ \parallel\ x)) \land \texttt{OH\_CONTEXT}\ c) \lor \\
\quad (\exists x\ \ c.\ (a_0\ \texttt{=}\ (\lambda t.\ x\ \parallel\ c\ t)) \land \texttt{OH\_CONTEXT}\ c) \lor \\
\quad (\exists L\ \ c.\ (a_0\ \texttt{=}\ (\lambda t.\ \nu\ L\ (c\ t))) \land \texttt{OH\_CONTEXT}\ c) \lor \\
\quad \exists rf\ \ c.\ (a_0\ \texttt{=}\ (\lambda t.\ \texttt{relab}\ (c\ t)\ rf)) \land \texttt{OH\_CONTEXT}\ c
\end{array}$$

[OH_CONTEXT_combin]

$\vdash \forall c_1\ \ c_2.\ \texttt{OH\_CONTEXT}\ c_1 \land \texttt{OH\_CONTEXT}\ c_2 \implies \texttt{OH\_CONTEXT}\ (c_1 \circ c_2)$

[OH_CONTEXT_ind]

$$\begin{array}{l}
\vdash \forall \mathit{OH\_CONTEXT}'. \\
\quad \mathit{OH\_CONTEXT}'\ (\lambda t.\ t) \land \\
\quad (\forall a\ \ c.\ \mathit{OH\_CONTEXT}'\ c \implies \mathit{OH\_CONTEXT}'\ (\lambda t.\ a..c\ t)) \land \\
\quad (\forall x\ \ c.\ \mathit{OH\_CONTEXT}'\ c \implies \mathit{OH\_CONTEXT}'\ (\lambda t.\ c\ t\ \texttt{+}\ x)) \land \\
\quad (\forall x\ \ c.\ \mathit{OH\_CONTEXT}'\ c \implies \mathit{OH\_CONTEXT}'\ (\lambda t.\ x\ \texttt{+}\ c\ t)) \land \\
\quad (\forall x\ \ c.\ \mathit{OH\_CONTEXT}'\ c \implies \mathit{OH\_CONTEXT}'\ (\lambda t.\ c\ t\ \parallel\ x)) \land \\
\quad (\forall x\ \ c.\ \mathit{OH\_CONTEXT}'\ c \implies \mathit{OH\_CONTEXT}'\ (\lambda t.\ x\ \parallel\ c\ t)) \land \\
\quad (\forall L\ \ c.\ \mathit{OH\_CONTEXT}'\ c \implies \mathit{OH\_CONTEXT}'\ (\lambda t.\ \nu\ L\ (c\ t))) \land \\
\quad (\forall rf\ \ c.\ \mathit{OH\_CONTEXT}'\ c \implies \mathit{OH\_CONTEXT}'\ (\lambda t.\ \texttt{relab}\ (c\ t)\ rf)) \implies \\
\quad \forall a_0.\ \texttt{OH\_CONTEXT}\ a_0 \implies \mathit{OH\_CONTEXT}'\ a_0
\end{array}$$

[OH_CONTEXT_IS_CONTEXT]

```
⊢ ∀c. OH_CONTEXT c ⟹ CONTEXT c
```

[OH_CONTEXT_rules]

```
⊢ OH_CONTEXT (λt. t) ∧
  (∀a c. OH_CONTEXT c ⟹ OH_CONTEXT (λt. a..c t)) ∧
  (∀x c. OH_CONTEXT c ⟹ OH_CONTEXT (λt. c t + x)) ∧
  (∀x c. OH_CONTEXT c ⟹ OH_CONTEXT (λt. x + c t)) ∧
  (∀x c. OH_CONTEXT c ⟹ OH_CONTEXT (λt. c t ∥ x)) ∧
  (∀x c. OH_CONTEXT c ⟹ OH_CONTEXT (λt. x ∥ c t)) ∧
  (∀L c. OH_CONTEXT c ⟹ OH_CONTEXT (λt. ν L (c t))) ∧
  ∀rf c. OH_CONTEXT c ⟹ OH_CONTEXT (λt. relab (c t) rf)
```

[OH_CONTEXT_strongind]

```
⊢ ∀OH_CONTEXT′.
     OH_CONTEXT′ (λt. t) ∧
     (∀a c.
        OH_CONTEXT c ∧ OH_CONTEXT′ c ⟹
        OH_CONTEXT′ (λt. a..c t)) ∧
     (∀x c.
        OH_CONTEXT c ∧ OH_CONTEXT′ c ⟹
        OH_CONTEXT′ (λt. c t + x)) ∧
     (∀x c.
        OH_CONTEXT c ∧ OH_CONTEXT′ c ⟹
        OH_CONTEXT′ (λt. x + c t)) ∧
     (∀x c.
        OH_CONTEXT c ∧ OH_CONTEXT′ c ⟹
        OH_CONTEXT′ (λt. c t ∥ x)) ∧
     (∀x c.
        OH_CONTEXT c ∧ OH_CONTEXT′ c ⟹
        OH_CONTEXT′ (λt. x ∥ c t)) ∧
     (∀L c.
        OH_CONTEXT c ∧ OH_CONTEXT′ c ⟹
        OH_CONTEXT′ (λt. ν L (c t))) ∧
     (∀rf c.
        OH_CONTEXT c ∧ OH_CONTEXT′ c ⟹
        OH_CONTEXT′ (λt. relab (c t) rf)) ⟹
     ∀a0. OH_CONTEXT a0 ⟹ OH_CONTEXT′ a0
```

[SEQ1]

```
⊢ SEQ (λt. t)
```

[SEQ2]

```
⊢ ∀p. SEQ (λt. p)
```

[SEQ3]

$$\vdash \forall a\ e.\ \texttt{SEQ}\ e \implies \texttt{SEQ}\ (\lambda t.\ a..e\ t)$$

[SEQ3a]

$$\vdash \forall a.\ \texttt{SEQ}\ (\lambda t.\ a..t)$$

[SEQ4]

$$\vdash \forall e_1\ e_2.\ \texttt{SEQ}\ e_1 \wedge \texttt{SEQ}\ e_2 \implies \texttt{SEQ}\ (\lambda t.\ e_1\ t\ \texttt{+}\ e_2\ t)$$

[SEQ_cases]

$$\begin{array}{l}
\vdash \forall a_0.\\
\quad \texttt{SEQ}\ a_0 \iff\\
\quad (a_0\ \texttt{=}\ (\lambda t.\ t)) \vee (\exists p.\ a_0\ \texttt{=}\ (\lambda t.\ p)) \vee\\
\quad (\exists a\ e.\ (a_0\ \texttt{=}\ (\lambda t.\ a..e\ t)) \wedge \texttt{SEQ}\ e) \vee\\
\quad \exists e_1\ e_2.\ (a_0\ \texttt{=}\ (\lambda t.\ e_1\ t\ \texttt{+}\ e_2\ t)) \wedge \texttt{SEQ}\ e_1 \wedge \texttt{SEQ}\ e_2
\end{array}$$

[SEQ_combin]

$$\vdash \forall E.\ \texttt{SEQ}\ E \implies \forall E'.\ \texttt{SEQ}\ E' \implies \texttt{SEQ}\ (E \circ E')$$

[SEQ_ind]

$$\begin{array}{l}
\vdash \forall \mathit{SEQ}'.\\
\quad \mathit{SEQ}'\ (\lambda t.\ t) \wedge (\forall p.\ \mathit{SEQ}'\ (\lambda t.\ p)) \wedge\\
\quad (\forall a\ e.\ \mathit{SEQ}'\ e \implies \mathit{SEQ}'\ (\lambda t.\ a..e\ t)) \wedge\\
\quad (\forall e_1\ e_2.\ \mathit{SEQ}'\ e_1 \wedge \mathit{SEQ}'\ e_2 \implies \mathit{SEQ}'\ (\lambda t.\ e_1\ t\ \texttt{+}\ e_2\ t)) \implies\\
\quad \forall a_0.\ \texttt{SEQ}\ a_0 \implies \mathit{SEQ}'\ a_0
\end{array}$$

[SEQ_IS_CONTEXT]

$$\vdash \forall e.\ \texttt{SEQ}\ e \implies \texttt{CONTEXT}\ e$$

[SEQ_rules]

$$\begin{array}{l}
\vdash \texttt{SEQ}\ (\lambda t.\ t) \wedge (\forall p.\ \texttt{SEQ}\ (\lambda t.\ p)) \wedge\\
\quad (\forall a\ e.\ \texttt{SEQ}\ e \implies \texttt{SEQ}\ (\lambda t.\ a..e\ t)) \wedge\\
\quad \forall e_1\ e_2.\ \texttt{SEQ}\ e_1 \wedge \texttt{SEQ}\ e_2 \implies \texttt{SEQ}\ (\lambda t.\ e_1\ t\ \texttt{+}\ e_2\ t)
\end{array}$$

[SEQ_strongind]

$$\begin{array}{l}
\vdash \forall \mathit{SEQ}'.\\
\quad \mathit{SEQ}'\ (\lambda t.\ t) \wedge (\forall p.\ \mathit{SEQ}'\ (\lambda t.\ p)) \wedge\\
\quad (\forall a\ e.\ \texttt{SEQ}\ e \wedge \mathit{SEQ}'\ e \implies \mathit{SEQ}'\ (\lambda t.\ a..e\ t)) \wedge\\
\quad (\forall e_1\ e_2.\\
\qquad \texttt{SEQ}\ e_1 \wedge \mathit{SEQ}'\ e_1 \wedge \texttt{SEQ}\ e_2 \wedge \mathit{SEQ}'\ e_2 \implies\\
\qquad \mathit{SEQ}'\ (\lambda t.\ e_1\ t\ \texttt{+}\ e_2\ t)) \implies\\
\quad \forall a_0.\ \texttt{SEQ}\ a_0 \implies \mathit{SEQ}'\ a_0
\end{array}$$

[SG1]

$$\vdash \forall p.\ \texttt{SG}\ (\lambda t.\ p)$$

[SG10]

$\vdash \forall e\ e'.\ \texttt{SG}\ (\lambda t.\ \tau..e\ t\ \texttt{+}\ \tau..e'\ t) \implies \texttt{SG}\ e \land \texttt{SG}\ e'$

[SG11]

$\vdash \forall e\ e'\ L.\ \texttt{SG}\ (\lambda t.\ \tau..e\ t\ \texttt{+}\ \texttt{label}\ L..e'\ t) \implies \texttt{SG}\ e$

[SG11']

$\vdash \forall e\ e'\ L.\ \texttt{SG}\ (\lambda t.\ \texttt{label}\ L..e\ t\ \texttt{+}\ \tau..e'\ t) \implies \texttt{SG}\ e'$

[SG2]

$\vdash \forall l\ e.\ \texttt{CONTEXT}\ e \implies \texttt{SG}\ (\lambda t.\ \texttt{label}\ l..e\ t)$

[SG3]

$\vdash \forall a\ e.\ \texttt{SG}\ e \implies \texttt{SG}\ (\lambda t.\ a..e\ t)$

[SG4]

$\vdash \forall e_1\ e_2.\ \texttt{SG}\ e_1 \land \texttt{SG}\ e_2 \implies \texttt{SG}\ (\lambda t.\ e_1\ t\ \texttt{+}\ e_2\ t)$

[SG5]

$\vdash \forall e_1\ e_2.\ \texttt{SG}\ e_1 \land \texttt{SG}\ e_2 \implies \texttt{SG}\ (\lambda t.\ e_1\ t \parallel e_2\ t)$

[SG6]

$\vdash \forall L\ e.\ \texttt{SG}\ e \implies \texttt{SG}\ (\lambda t.\ \nu\ L\ (e\ t))$

[SG7]

$\vdash \forall rf\ e.\ \texttt{SG}\ e \implies \texttt{SG}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf)$

[SG8]

$\vdash \forall e.\ \texttt{SG}\ (\lambda t.\ \tau..e\ t) \implies \texttt{SG}\ e$

[SG9]

$\vdash \forall e\ e'.\ \texttt{SG}\ (\lambda t.\ e\ t\ \texttt{+}\ e'\ t) \implies \texttt{SG}\ e \land \texttt{SG}\ e'$

[SG_cases]

$$
\begin{aligned}
&\vdash \forall a_0.\\
&\qquad \texttt{SG}\ a_0 \iff\\
&\qquad (\exists p.\ a_0 = (\lambda t.\ p)) \lor\\
&\qquad (\exists l\ e.\ (a_0 = (\lambda t.\ \texttt{label}\ l..e\ t)) \land \texttt{CONTEXT}\ e) \lor\\
&\qquad (\exists a\ e.\ (a_0 = (\lambda t.\ a..e\ t)) \land \texttt{SG}\ e) \lor\\
&\qquad (\exists e_1\ e_2.\ (a_0 = (\lambda t.\ e_1\ t\ \texttt{+}\ e_2\ t)) \land \texttt{SG}\ e_1 \land \texttt{SG}\ e_2) \lor\\
&\qquad (\exists e_1\ e_2.\ (a_0 = (\lambda t.\ e_1\ t \parallel e_2\ t)) \land \texttt{SG}\ e_1 \land \texttt{SG}\ e_2) \lor\\
&\qquad (\exists L\ e.\ (a_0 = (\lambda t.\ \nu\ L\ (e\ t))) \land \texttt{SG}\ e) \lor\\
&\qquad \exists rf\ e.\ (a_0 = (\lambda t.\ \texttt{relab}\ (e\ t)\ rf)) \land \texttt{SG}\ e
\end{aligned}
$$

[SG_GSEQ_combin]

$$\vdash \forall E.\ \mathtt{SG}\ E \land \mathtt{GSEQ}\ E \Longrightarrow \forall H.\ \mathtt{GSEQ}\ H \Longrightarrow \mathtt{SG}\ (H \circ E) \land \mathtt{GSEQ}\ (H \circ E)$$

[SG_GSEQ_strong_induction]

$$\begin{array}{l}
\vdash \forall R.\\
\qquad (\forall p.\ R\ (\lambda t.\ p)) \land (\forall l\ e.\ \mathtt{GSEQ}\ e \Longrightarrow R\ (\lambda t.\ \mathtt{label}\ l..e\ t)) \land\\
\qquad (\forall a\ e.\ \mathtt{SG}\ e \land \mathtt{GSEQ}\ e \land R\ e \Longrightarrow R\ (\lambda t.\ a..e\ t)) \land\\
\qquad (\forall e_1\ e_2.\\
\qquad\quad \mathtt{SG}\ e_1 \land \mathtt{GSEQ}\ e_1 \land R\ e_1 \land \mathtt{SG}\ e_2 \land \mathtt{GSEQ}\ e_2 \land R\ e_2 \Longrightarrow\\
\qquad\quad R\ (\lambda t.\ \tau..e_1\ t\ \mathtt{+}\ \tau..e_2\ t)) \land\\
\qquad (\forall l_2\ e_1\ e_2.\\
\qquad\quad \mathtt{SG}\ e_1 \land \mathtt{GSEQ}\ e_1 \land R\ e_1 \land \mathtt{GSEQ}\ e_2 \Longrightarrow\\
\qquad\quad R\ (\lambda t.\ \tau..e_1\ t\ \mathtt{+}\ \mathtt{label}\ l_2..e_2\ t)) \land\\
\qquad (\forall l_1\ e_1\ e_2.\\
\qquad\quad \mathtt{GSEQ}\ e_1 \land \mathtt{SG}\ e_2 \land \mathtt{GSEQ}\ e_2 \land R\ e_2 \Longrightarrow\\
\qquad\quad R\ (\lambda t.\ \mathtt{label}\ l_1..e_1\ t\ \mathtt{+}\ \tau..e_2\ t)) \land\\
\qquad (\forall l_1\ l_2\ e_1\ e_2.\\
\qquad\quad \mathtt{GSEQ}\ e_1 \land \mathtt{GSEQ}\ e_2 \Longrightarrow\\
\qquad\quad R\ (\lambda t.\ \mathtt{label}\ l_1..e_1\ t\ \mathtt{+}\ \mathtt{label}\ l_2..e_2\ t)) \Longrightarrow\\
\qquad \forall e.\ \mathtt{SG}\ e \land \mathtt{GSEQ}\ e \Longrightarrow R\ e
\end{array}$$

[SG_IMP_WG]

$$\vdash \forall e.\ \mathtt{SG}\ e \Longrightarrow \mathtt{WG}\ e$$

[SG_ind]

$$\begin{array}{l}
\vdash \forall SG'.\\
\qquad (\forall p.\ SG'\ (\lambda t.\ p)) \land\\
\qquad (\forall l\ e.\ \mathtt{CONTEXT}\ e \Longrightarrow SG'\ (\lambda t.\ \mathtt{label}\ l..e\ t)) \land\\
\qquad (\forall a\ e.\ SG'\ e \Longrightarrow SG'\ (\lambda t.\ a..e\ t)) \land\\
\qquad (\forall e_1\ e_2.\ SG'\ e_1 \land SG'\ e_2 \Longrightarrow SG'\ (\lambda t.\ e_1\ t\ \mathtt{+}\ e_2\ t)) \land\\
\qquad (\forall e_1\ e_2.\ SG'\ e_1 \land SG'\ e_2 \Longrightarrow SG'\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\qquad (\forall L\ e.\ SG'\ e \Longrightarrow SG'\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\qquad (\forall rf\ e.\ SG'\ e \Longrightarrow SG'\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)) \Longrightarrow\\
\qquad \forall a_0.\ \mathtt{SG}\ a_0 \Longrightarrow SG'\ a_0
\end{array}$$

[SG_IS_CONTEXT]

$$\vdash \forall e.\ \mathtt{SG}\ e \Longrightarrow \mathtt{CONTEXT}\ e$$

[SG_rules]

$$\begin{array}{l}
\vdash (\forall p.\ \mathtt{SG}\ (\lambda t.\ p)) \land\\
\quad (\forall l\ e.\ \mathtt{CONTEXT}\ e \Longrightarrow \mathtt{SG}\ (\lambda t.\ \mathtt{label}\ l..e\ t)) \land\\
\quad (\forall a\ e.\ \mathtt{SG}\ e \Longrightarrow \mathtt{SG}\ (\lambda t.\ a..e\ t)) \land\\
\quad (\forall e_1\ e_2.\ \mathtt{SG}\ e_1 \land \mathtt{SG}\ e_2 \Longrightarrow \mathtt{SG}\ (\lambda t.\ e_1\ t\ \mathtt{+}\ e_2\ t)) \land\\
\quad (\forall e_1\ e_2.\ \mathtt{SG}\ e_1 \land \mathtt{SG}\ e_2 \Longrightarrow \mathtt{SG}\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\quad (\forall L\ e.\ \mathtt{SG}\ e \Longrightarrow \mathtt{SG}\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad \forall rf\ e.\ \mathtt{SG}\ e \Longrightarrow \mathtt{SG}\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)
\end{array}$$

[SG_SEQ_combin]

$$\vdash \forall E.\ \mathtt{SG}\ E \land \mathtt{SEQ}\ E \Longrightarrow \forall H.\ \mathtt{SEQ}\ H \Longrightarrow \mathtt{SG}\ (H \circ E) \land \mathtt{SEQ}\ (H \circ E)$$

[SG_SEQ_strong_induction]

$$\begin{array}{l}
\vdash \forall R.\\
\quad (\forall p.\ R\ (\lambda t.\ p)) \land (\forall l\ e.\ \mathtt{SEQ}\ e \Longrightarrow R\ (\lambda t.\ \mathtt{label}\ l\mathtt{..}e\ t)) \land\\
\quad (\forall a\ e.\ \mathtt{SG}\ e \land \mathtt{SEQ}\ e \land R\ e \Longrightarrow R\ (\lambda t.\ a\mathtt{..}e\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathtt{SG}\ e_1 \land \mathtt{SEQ}\ e_1 \land R\ e_1 \land \mathtt{SG}\ e_2 \land \mathtt{SEQ}\ e_2 \land R\ e_2 \Longrightarrow\\
\qquad R\ (\lambda t.\ e_1\ t\ \mathtt{+}\ e_2\ t)) \Longrightarrow\\
\quad \forall e.\ \mathtt{SG}\ e \land \mathtt{SEQ}\ e \Longrightarrow R\ e
\end{array}$$

[SG_strongind]

$$\begin{array}{l}
\vdash \forall SG'.\\
\quad (\forall p.\ SG'\ (\lambda t.\ p)) \land\\
\quad (\forall l\ e.\ \mathtt{CONTEXT}\ e \Longrightarrow SG'\ (\lambda t.\ \mathtt{label}\ l\mathtt{..}e\ t)) \land\\
\quad (\forall a\ e.\ \mathtt{SG}\ e \land SG'\ e \Longrightarrow SG'\ (\lambda t.\ a\mathtt{..}e\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathtt{SG}\ e_1 \land SG'\ e_1 \land \mathtt{SG}\ e_2 \land SG'\ e_2 \Longrightarrow\\
\qquad SG'\ (\lambda t.\ e_1\ t\ \mathtt{+}\ e_2\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathtt{SG}\ e_1 \land SG'\ e_1 \land \mathtt{SG}\ e_2 \land SG'\ e_2 \Longrightarrow\\
\qquad SG'\ (\lambda t.\ e_1\ t\ \|\ e_2\ t)) \land\\
\quad (\forall L\ e.\ \mathtt{SG}\ e \land SG'\ e \Longrightarrow SG'\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad (\forall rf\ e.\ \mathtt{SG}\ e \land SG'\ e \Longrightarrow SG'\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)) \Longrightarrow\\
\quad \forall a_0.\ \mathtt{SG}\ a_0 \Longrightarrow SG'\ a_0
\end{array}$$

[STRONG_EQUIV_congruence]

$$\vdash \mathtt{congruence}\ \mathtt{STRONG\_EQUIV}$$

[STRONG_EQUIV_SUBST_CONTEXT]

$$\begin{array}{l}
\vdash \forall P\ Q.\\
\quad \mathtt{STRONG\_EQUIV}\ P\ Q \Longrightarrow \forall E.\ \mathtt{CONTEXT}\ E \Longrightarrow \mathtt{STRONG\_EQUIV}\ (E\ P)\ (E\ Q)
\end{array}$$

[WEAK_EQUIV_congruence]

$$\vdash \mathtt{congruence1}\ \mathtt{WEAK\_EQUIV}$$

[WEAK_EQUIV_SUBST_GCONTEXT]

$$\begin{array}{l}
\vdash \forall P\ Q.\\
\quad \mathtt{WEAK\_EQUIV}\ P\ Q \Longrightarrow \forall E.\ \mathtt{GCONTEXT}\ E \Longrightarrow \mathtt{WEAK\_EQUIV}\ (E\ P)\ (E\ Q)
\end{array}$$

[WEAK_EQUIV_SUBST_GSEQ]

$$\vdash \forall P\ Q.\ \mathtt{WEAK\_EQUIV}\ P\ Q \Longrightarrow \forall E.\ \mathtt{GSEQ}\ E \Longrightarrow \mathtt{WEAK\_EQUIV}\ (E\ P)\ (E\ Q)$$

[WG1]

$$\vdash \forall a.\ \mathtt{WG}\ (\lambda t.\ a\mathtt{..}t)$$

[WG2]

```
⊢ ∀p. WG (λt. p)
```

[WG3]

```
⊢ ∀a e. CONTEXT e ⟹ WG (λt. a..e t)
```

[WG4]

```
⊢ ∀e1 e2. WG e1 ∧ WG e2 ⟹ WG (λt. e1 t + e2 t)
```

[WG5]

```
⊢ ∀e1 e2. WG e1 ∧ WG e2 ⟹ WG (λt. e1 t ∥ e2 t)
```

[WG6]

```
⊢ ∀L e. WG e ⟹ WG (λt. ν L (e t))
```

[WG7]

```
⊢ ∀rf e. WG e ⟹ WG (λt. relab (e t) rf)
```

[WG_cases]

```
⊢ ∀a0.
     WG a0 ⟺
     (∃p. a0 = (λt. p)) ∨
     (∃a e. (a0 = (λt. a..e t)) ∧ CONTEXT e) ∨
     (∃e1 e2. (a0 = (λt. e1 t + e2 t)) ∧ WG e1 ∧ WG e2) ∨
     (∃e1 e2. (a0 = (λt. e1 t ∥ e2 t)) ∧ WG e1 ∧ WG e2) ∨
     (∃L e. (a0 = (λt. ν L (e t))) ∧ WG e) ∨
     ∃rf e. (a0 = (λt. relab (e t) rf)) ∧ WG e
```

[WG_ind]

```
⊢ ∀WG′.
     (∀p. WG′ (λt. p)) ∧ (∀a e. CONTEXT e ⟹ WG′ (λt. a..e t)) ∧
     (∀e1 e2. WG′ e1 ∧ WG′ e2 ⟹ WG′ (λt. e1 t + e2 t)) ∧
     (∀e1 e2. WG′ e1 ∧ WG′ e2 ⟹ WG′ (λt. e1 t ∥ e2 t)) ∧
     (∀L e. WG′ e ⟹ WG′ (λt. ν L (e t))) ∧
     (∀rf e. WG′ e ⟹ WG′ (λt. relab (e t) rf)) ⟹
     ∀a0. WG a0 ⟹ WG′ a0
```

[WG_IS_CONTEXT]

```
⊢ ∀e. WG e ⟹ CONTEXT e
```

[WG_rules]

```
⊢ (∀p. WG (λt. p)) ∧ (∀a e. CONTEXT e ⟹ WG (λt. a..e t)) ∧
  (∀e1 e2. WG e1 ∧ WG e2 ⟹ WG (λt. e1 t + e2 t)) ∧
  (∀e1 e2. WG e1 ∧ WG e2 ⟹ WG (λt. e1 t ∥ e2 t)) ∧
  (∀L e. WG e ⟹ WG (λt. ν L (e t))) ∧
  ∀rf e. WG e ⟹ WG (λt. relab (e t) rf)
```

[WG_strongind]

$$\begin{array}{l}
\vdash \forall\, WG'.\\
\quad (\forall p.\ WG'\ (\lambda t.\ p)) \land (\forall a\ e.\ \mathtt{CONTEXT}\ e \implies WG'\ (\lambda t.\ a..e\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathtt{WG}\ e_1 \land WG'\ e_1 \land \mathtt{WG}\ e_2 \land WG'\ e_2 \implies\\
\qquad WG'\ (\lambda t.\ e_1\ t + e_2\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathtt{WG}\ e_1 \land WG'\ e_1 \land \mathtt{WG}\ e_2 \land WG'\ e_2 \implies\\
\qquad WG'\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\quad (\forall L\ e.\ \mathtt{WG}\ e \land WG'\ e \implies WG'\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad (\forall rf\ e.\ \mathtt{WG}\ e \land WG'\ e \implies WG'\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)) \implies\\
\quad \forall a_0.\ \mathtt{WG}\ a_0 \implies WG'\ a_0
\end{array}$$

[WGS1]

$$\vdash \forall a.\ \mathtt{WGS}\ (\lambda t.\ a..t)$$

[WGS2]

$$\vdash \forall p.\ \mathtt{WGS}\ (\lambda t.\ p)$$

[WGS3]

$$\vdash \forall a\ e.\ \mathtt{GCONTEXT}\ e \implies \mathtt{WGS}\ (\lambda t.\ a..e\ t)$$

[WGS4]

$$\begin{array}{l}
\vdash \forall a_1\ a_2\ e_1\ e_2.\\
\quad \mathtt{GCONTEXT}\ e_1 \land \mathtt{GCONTEXT}\ e_2 \implies \mathtt{WGS}\ (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)
\end{array}$$

[WGS5]

$$\vdash \forall e_1\ e_2.\ \mathtt{WGS}\ e_1 \land \mathtt{WGS}\ e_2 \implies \mathtt{WGS}\ (\lambda t.\ e_1\ t \parallel e_2\ t)$$

[WGS6]

$$\vdash \forall L\ e.\ \mathtt{WGS}\ e \implies \mathtt{WGS}\ (\lambda t.\ \nu\ L\ (e\ t))$$

[WGS7]

$$\vdash \forall rf\ e.\ \mathtt{WGS}\ e \implies \mathtt{WGS}\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)$$

[WGS_cases]

$$\begin{array}{l}
\vdash \forall a_0.\\
\quad \mathtt{WGS}\ a_0 \iff\\
\quad (\exists p.\ a_0 = (\lambda t.\ p)) \lor\\
\quad (\exists a\ e.\ (a_0 = (\lambda t.\ a..e\ t)) \land \mathtt{GCONTEXT}\ e) \lor\\
\quad (\exists a_1\ a_2\ e_1\ e_2.\\
\qquad (a_0 = (\lambda t.\ a_1..e_1\ t + a_2..e_2\ t)) \land \mathtt{GCONTEXT}\ e_1 \land\\
\qquad \mathtt{GCONTEXT}\ e_2) \lor\\
\quad (\exists e_1\ e_2.\ (a_0 = (\lambda t.\ e_1\ t \parallel e_2\ t)) \land \mathtt{WGS}\ e_1 \land \mathtt{WGS}\ e_2) \lor\\
\quad (\exists L\ e.\ (a_0 = (\lambda t.\ \nu\ L\ (e\ t))) \land \mathtt{WGS}\ e) \lor\\
\quad \exists rf\ e.\ (a_0 = (\lambda t.\ \mathtt{relab}\ (e\ t)\ rf)) \land \mathtt{WGS}\ e
\end{array}$$

[WGS_ind]

$$
\begin{array}{l}
\vdash \forall \mathit{WGS}'.\\
\quad (\forall p.\ \mathit{WGS}'\ (\lambda t.\ p)) \land\\
\quad (\forall a\ e.\ \mathtt{GCONTEXT}\ e \Longrightarrow \mathit{WGS}'\ (\lambda t.\ a\mathtt{..}e\ t)) \land\\
\quad (\forall a_1\ a_2\ e_1\ e_2.\\
\qquad \mathtt{GCONTEXT}\ e_1 \land \mathtt{GCONTEXT}\ e_2 \Longrightarrow\\
\qquad \mathit{WGS}'\ (\lambda t.\ a_1\mathtt{..}e_1\ t\ \mathtt{+}\ a_2\mathtt{..}e_2\ t)) \land\\
\quad (\forall e_1\ e_2.\ \mathit{WGS}'\ e_1 \land \mathit{WGS}'\ e_2 \Longrightarrow \mathit{WGS}'\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\quad (\forall L\ e.\ \mathit{WGS}'\ e \Longrightarrow \mathit{WGS}'\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad (\forall \mathit{rf}\ e.\ \mathit{WGS}'\ e \Longrightarrow \mathit{WGS}'\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ \mathit{rf})) \Longrightarrow\\
\quad \forall a_0.\ \mathtt{WGS}\ a_0 \Longrightarrow \mathit{WGS}'\ a_0
\end{array}
$$

[WGS_IS_CONTEXT]

$$\vdash \forall e.\ \mathtt{WGS}\ e \Longrightarrow \mathtt{CONTEXT}\ e$$

[WGS_IS_GCONTEXT]

$$\vdash \forall e.\ \mathtt{WGS}\ e \Longrightarrow \mathtt{GCONTEXT}\ e$$

[WGS_rules]

$$
\begin{array}{l}
\vdash (\forall p.\ \mathtt{WGS}\ (\lambda t.\ p)) \land (\forall a\ e.\ \mathtt{GCONTEXT}\ e \Longrightarrow \mathtt{WGS}\ (\lambda t.\ a\mathtt{..}e\ t)) \land\\
\quad (\forall a_1\ a_2\ e_1\ e_2.\\
\qquad \mathtt{GCONTEXT}\ e_1 \land \mathtt{GCONTEXT}\ e_2 \Longrightarrow\\
\qquad \mathtt{WGS}\ (\lambda t.\ a_1\mathtt{..}e_1\ t\ \mathtt{+}\ a_2\mathtt{..}e_2\ t)) \land\\
\quad (\forall e_1\ e_2.\ \mathtt{WGS}\ e_1 \land \mathtt{WGS}\ e_2 \Longrightarrow \mathtt{WGS}\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\quad (\forall L\ e.\ \mathtt{WGS}\ e \Longrightarrow \mathtt{WGS}\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad \forall \mathit{rf}\ e.\ \mathtt{WGS}\ e \Longrightarrow \mathtt{WGS}\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ \mathit{rf})
\end{array}
$$

[WGS_strongind]

$$
\begin{array}{l}
\vdash \forall \mathit{WGS}'.\\
\quad (\forall p.\ \mathit{WGS}'\ (\lambda t.\ p)) \land\\
\quad (\forall a\ e.\ \mathtt{GCONTEXT}\ e \Longrightarrow \mathit{WGS}'\ (\lambda t.\ a\mathtt{..}e\ t)) \land\\
\quad (\forall a_1\ a_2\ e_1\ e_2.\\
\qquad \mathtt{GCONTEXT}\ e_1 \land \mathtt{GCONTEXT}\ e_2 \Longrightarrow\\
\qquad \mathit{WGS}'\ (\lambda t.\ a_1\mathtt{..}e_1\ t\ \mathtt{+}\ a_2\mathtt{..}e_2\ t)) \land\\
\quad (\forall e_1\ e_2.\\
\qquad \mathtt{WGS}\ e_1 \land \mathit{WGS}'\ e_1 \land \mathtt{WGS}\ e_2 \land \mathit{WGS}'\ e_2 \Longrightarrow\\
\qquad \mathit{WGS}'\ (\lambda t.\ e_1\ t \parallel e_2\ t)) \land\\
\quad (\forall L\ e.\ \mathtt{WGS}\ e \land \mathit{WGS}'\ e \Longrightarrow \mathit{WGS}'\ (\lambda t.\ \nu\ L\ (e\ t))) \land\\
\quad (\forall \mathit{rf}\ e.\ \mathtt{WGS}\ e \land \mathit{WGS}'\ e \Longrightarrow \mathit{WGS}'\ (\lambda t.\ \mathtt{relab}\ (e\ t)\ \mathit{rf})) \Longrightarrow\\
\quad \forall a_0.\ \mathtt{WGS}\ a_0 \Longrightarrow \mathit{WGS}'\ a_0
\end{array}
$$

9.9 CoarsestCongr Theory

Built: 08 Dicembre 2017

Parent Theories: Trace, Congruence

9.9.1 Definitions

[free_action_def]

```
⊢ ∀p. free_action p ⟺ ∃a. ∀p′. ¬(p =label a⇒ p′)
```

[KLOP_def]

```
⊢ (∀a. KLOP a 0 = nil) ∧
  ∀a n. KLOP a (SUC n) = KLOP a n + label a..KLOP a n
```

[SUM_EQUIV]

```
⊢ SUM_EQUIV = (λp q. ∀r. WEAK_EQUIV (p + r) (q + r))
```

[WEAK_CONGR]

```
⊢ WEAK_CONGR = CC WEAK_EQUIV
```

9.9.2 Theorems

[COARSEST_CONGR_FINITE]

```
⊢ ∀p q.
    finite_state p ∧ finite_state q ⟹
    (OBS_CONGR p q ⟺ ∀r. WEAK_EQUIV (p + r) (q + r))
```

[COARSEST_CONGR_LR]

```
⊢ ∀p q. OBS_CONGR p q ⟹ ∀r. WEAK_EQUIV (p + r) (q + r)
```

[COARSEST_CONGR_RL]

```
⊢ ∀p q.
    free_action p ∧ free_action q ⟹
    (∀r. WEAK_EQUIV (p + r) (q + r)) ⟹
    OBS_CONGR p q
```

[COARSEST_CONGR_THM]

```
⊢ ∀p q.
    free_action p ∧ free_action q ⟹
    (OBS_CONGR p q ⟺ ∀r. WEAK_EQUIV (p + r) (q + r))
```

[DENG_LEMMA]

```
⊢ ∀p q.
    WEAK_EQUIV p q ⟹
    (∃p′. p −τ→ p′ ∧ WEAK_EQUIV p′ q) ∨
    (∃q′. q −τ→ q′ ∧ WEAK_EQUIV p q′) ∨ OBS_CONGR p q
```

[HENNESSY_LEMMA]

```
⊢ ∀p q.
    WEAK_EQUIV p q ⟺
    OBS_CONGR p q ∨ OBS_CONGR p (τ..q) ∨ OBS_CONGR (τ..p) q
```

[HENNESSY_LEMMA_LR]

```
⊢ ∀p q.
     WEAK_EQUIV p q ⟹
     OBS_CONGR p q ∨ OBS_CONGR p (τ..q) ∨ OBS_CONGR (τ..p) q
```

[HENNESSY_LEMMA_RL]

```
⊢ ∀p q.
     OBS_CONGR p q ∨ OBS_CONGR p (τ..q) ∨ OBS_CONGR (τ..p) q ⟹
     WEAK_EQUIV p q
```

[INFINITE_EXISTS_LEMMA]

```
⊢ ∀R A B.
     equivalence R ⟹
     FINITE A ∧ INFINITE B ∧
     (∀x y. x ∈ B ∧ y ∈ B ∧ x ≠ y ⟹ ¬R x y) ⟹
     ∃k. k ∈ B ∧ ∀n. n ∈ A ⟹ ¬R n k
```

[K0_NO_TRANS]

```
⊢ ∀a u E. ¬(KLOP a 0 −u→ E)
```

[KLOP_def_compute]

```
⊢ (∀a. KLOP a 0 = nil) ∧
  (∀a n.
     KLOP a (NUMERAL (BIT1 n)) =
     KLOP a (NUMERAL (BIT1 n) - 1) +
     label a..KLOP a (NUMERAL (BIT1 n) - 1)) ∧
  ∀a n.
    KLOP a (NUMERAL (BIT2 n)) =
    KLOP a (NUMERAL (BIT1 n)) +
    label a..KLOP a (NUMERAL (BIT1 n))
```

[KLOP_LEMMA_FINITE]

```
⊢ ∀p q.
     finite_state p ∧ finite_state q ⟹
     ∃k.
       STABLE k ∧ (∀p′ u. p =u⇒ p′ ⟹ ¬WEAK_EQUIV p′ k) ∧
       ∀q′ u. q =u⇒ q′ ⟹ ¬WEAK_EQUIV q′ k
```

[KLOP_ONE_ONE]

```
⊢ ∀a. ONE_ONE (KLOP a)
```

[KLOP_PROP0]

```
⊢ ∀a n. STABLE (KLOP a n)
```

[KLOP_PROP1]

```
⊢ ∀a n E. KLOP a n −label a→ E ⟺ ∃m. m < n ∧ (E = KLOP a m)
```

[KLOP_PROP1']

$\vdash \forall a\ n\ E.\ \mathtt{KLOP}\ a\ n\ \mathtt{=label}\ a\!\Rightarrow\ E \iff \exists m.\ m < n \land (E = \mathtt{KLOP}\ a\ m)$

[KLOP_PROP1_LR]

$\vdash \forall a\ n\ E.\ \mathtt{KLOP}\ a\ n\ \mathtt{-label}\ a\!\rightarrow\ E \implies \exists m.\ m < n \land (E = \mathtt{KLOP}\ a\ m)$

[KLOP_PROP1_RL]

$\vdash \forall a\ n\ E.\ (\exists m.\ m < n \land (E = \mathtt{KLOP}\ a\ m)) \implies \mathtt{KLOP}\ a\ n\ \mathtt{-label}\ a\!\rightarrow\ E$

[KLOP_PROP2]

$\vdash \forall a\ n\ m.\ m < n \implies \neg\mathtt{STRONG\_EQUIV}\ (\mathtt{KLOP}\ a\ m)\ (\mathtt{KLOP}\ a\ n)$

[KLOP_PROP2']

$\vdash \forall a\ n\ m.\ m < n \implies \neg\mathtt{WEAK\_EQUIV}\ (\mathtt{KLOP}\ a\ m)\ (\mathtt{KLOP}\ a\ n)$

[OBS_CONGR_IMP_WEAK_CONGR]

$\vdash \forall p\ q.\ \mathtt{OBS\_CONGR}\ p\ q \implies \mathtt{WEAK\_CONGR}\ p\ q$

[PROP3_COMMON]

$$\begin{array}{l}
\vdash \forall p\ q.\\
\quad\quad (\exists k.\\
\quad\quad\quad \mathtt{STABLE}\ k \land (\forall p'\ u.\ p\ \mathtt{=}u\mathtt{\Rightarrow}\ p' \implies \neg\mathtt{WEAK\_EQUIV}\ p'\ k) \land\\
\quad\quad\quad \forall q'\ u.\ q\ \mathtt{=}u\mathtt{\Rightarrow}\ q' \implies \neg\mathtt{WEAK\_EQUIV}\ q'\ k) \implies\\
\quad\quad (\forall r.\ \mathtt{WEAK\_EQUIV}\ (p + r)\ (q + r)) \implies\\
\quad\quad \mathtt{OBS\_CONGR}\ p\ q
\end{array}$$

[TAU_STRAT]

$\vdash \forall E\ E'.\ \mathtt{OBS\_CONGR}\ (E + \tau..(E' + E))\ (\tau..(E' + E))$

[WEAK_CONGR_congruence]

$\vdash \mathtt{congruence}\ \mathtt{WEAK\_CONGR}$

[WEAK_CONGR_IMP_SUM_EQUIV]

$\vdash \forall p\ q.\ \mathtt{WEAK\_CONGR}\ p\ q \implies \mathtt{SUM\_EQUIV}\ p\ q$

[WEAK_CONGR_THM]

$\vdash \mathtt{WEAK\_CONGR} = (\lambda g\ h.\ \forall c.\ \mathtt{CONTEXT}\ c \implies \mathtt{WEAK\_EQUIV}\ (c\ g)\ (c\ h))$

9.10 BisimulationUpto Theory

Built: 08 Dicembre 2017
Parent Theories: Congruence

9.10.1 Definitions

[OBS_BISIM_UPTO]

```
$\vdash \forall \mathit{Obsm}.$
    OBS_BISIM_UPTO $\mathit{Obsm} \iff$
    $\forall E\ E'.$
      $\mathit{Obsm}\ E\ E' \implies$
      $\forall u.$
        $(\forall E_1.$
           $E\ {-u\rightarrow}\ E_1 \implies$
           $\exists E_2.$
             $E'\ {=u\Rightarrow}\ E_2\ \wedge$
             (WEAK_EQUIV O $\mathit{Obsm}$ O STRONG_EQUIV) $E_1\ E_2)\ \wedge$
        $\forall E_2.$
          $E'\ {-u\rightarrow}\ E_2 \implies$
          $\exists E_1.$
            $E\ {=u\Rightarrow}\ E_1\ \wedge$ (STRONG_EQUIV O $\mathit{Obsm}$ O WEAK_EQUIV) $E_1\ E_2$
```

[STRONG_BISIM_UPTO]

```
$\vdash \forall \mathit{Bsm}.$
    STRONG_BISIM_UPTO $\mathit{Bsm} \iff$
    $\forall E\ E'.$
      $\mathit{Bsm}\ E\ E' \implies$
      $\forall u.$
        $(\forall E_1.$
           $E\ {-u\rightarrow}\ E_1 \implies$
           $\exists E_2.$
             $E'\ {-u\rightarrow}\ E_2\ \wedge$
             (STRONG_EQUIV O $\mathit{Bsm}$ O STRONG_EQUIV) $E_1\ E_2)\ \wedge$
        $\forall E_2.$
          $E'\ {-u\rightarrow}\ E_2 \implies$
          $\exists E_1.$
            $E\ {-u\rightarrow}\ E_1\ \wedge$
            (STRONG_EQUIV O $\mathit{Bsm}$ O STRONG_EQUIV) $E_1\ E_2$
```

[WEAK_BISIM_UPTO]

```
$\vdash \forall \mathit{Wbsm}.$
    WEAK_BISIM_UPTO $\mathit{Wbsm} \iff$
    $\forall E\ E'.$
      $\mathit{Wbsm}\ E\ E' \implies$
      $(\forall l.$
         $(\forall E_1.$
            $E\ {-}$label $l{\rightarrow}\ E_1 \implies$
            $\exists E_2.$
              $E'\ {=}$label $l{\Rightarrow}\ E_2\ \wedge$
              (WEAK_EQUIV O $\mathit{Wbsm}$ O STRONG_EQUIV) $E_1\ E_2)\ \wedge$
         $\forall E_2.$
```

```
            $E'$ −label $l$→ $E_2$ $\Longrightarrow$
            $\exists\, E_1$.
              $E$ =label $l$⇒ $E_1$ $\wedge$
              (STRONG_EQUIV O $Wbsm$ O WEAK_EQUIV) $E_1$ $E_2$) $\wedge$
        ($\forall\, E_1$.
           $E$ −$\tau$→ $E_1$ $\Longrightarrow$
           $\exists\, E_2$.
             EPS $E'$ $E_2$ $\wedge$
             (WEAK_EQUIV O $Wbsm$ O STRONG_EQUIV) $E_1$ $E_2$) $\wedge$
        $\forall\, E_2$.
          $E'$ −$\tau$→ $E_2$ $\Longrightarrow$
          $\exists\, E_1$.
            EPS $E$ $E_1$ $\wedge$ (STRONG_EQUIV O $Wbsm$ O WEAK_EQUIV) $E_1$ $E_2$
```

[WEAK_BISIM_UPTO_ALT]

```
 ⊢ $\forall\, Wbsm$.
     WEAK_BISIM_UPTO_ALT $Wbsm$ $\iff$
     $\forall\, E\ E'$.
       $Wbsm$ $E$ $E'$ $\Longrightarrow$
       ($\forall\, l$.
          ($\forall\, E_1$.
             $E$ =label $l$⇒ $E_1$ $\Longrightarrow$
             $\exists\, E_2$.
               $E'$ =label $l$⇒ $E_2$ $\wedge$
               (WEAK_EQUIV O $Wbsm$ O WEAK_EQUIV) $E_1$ $E_2$) $\wedge$
          $\forall\, E_2$.
            $E'$ =label $l$⇒ $E_2$ $\Longrightarrow$
            $\exists\, E_1$.
              $E$ =label $l$⇒ $E_1$ $\wedge$
              (WEAK_EQUIV O $Wbsm$ O WEAK_EQUIV) $E_1$ $E_2$) $\wedge$
       ($\forall\, E_1$.
          $E$ =$\tau$⇒ $E_1$ $\Longrightarrow$
          $\exists\, E_2$.
            EPS $E'$ $E_2$ $\wedge$
            (WEAK_EQUIV O $Wbsm$ O WEAK_EQUIV) $E_1$ $E_2$) $\wedge$
       $\forall\, E_2$.
         $E'$ =$\tau$⇒ $E_2$ $\Longrightarrow$
         $\exists\, E_1$. EPS $E$ $E_1$ $\wedge$ (WEAK_EQUIV O $Wbsm$ O WEAK_EQUIV) $E_1$ $E_2$
```

9.10.2 Theorems

[CONVERSE_OBS_BISIM_UPTO]

```
 ⊢ $\forall\, Obsm$. OBS_BISIM_UPTO $Obsm$ $\Longrightarrow$ OBS_BISIM_UPTO (relinv $Obsm$)
```

[CONVERSE_STRONG_BISIM_UPTO]

```
 ⊢ $\forall\, Wbsm$.
     STRONG_BISIM_UPTO $Wbsm$ $\Longrightarrow$ STRONG_BISIM_UPTO (relinv $Wbsm$)
```

[CONVERSE_WEAK_BISIM_UPTO]

$$\vdash \forall\, \mathit{Wbsm}.\ \texttt{WEAK\_BISIM\_UPTO}\ \mathit{Wbsm} \Longrightarrow \texttt{WEAK\_BISIM\_UPTO}\ (\texttt{relinv}\ \mathit{Wbsm})$$

[CONVERSE_WEAK_BISIM_UPTO_lemma]

$$\begin{array}{l}\vdash \forall\, \mathit{Wbsm}\ E\ E'.\\ \qquad (\texttt{WEAK\_EQUIV O relinv}\ \mathit{Wbsm}\ \texttt{O STRONG\_EQUIV})\ E\ E' \iff\\ \qquad (\texttt{STRONG\_EQUIV O}\ \mathit{Wbsm}\ \texttt{O WEAK\_EQUIV})\ E'\ E\end{array}$$

[CONVERSE_WEAK_BISIM_UPTO_lemma’]

$$\begin{array}{l}\vdash \forall\, \mathit{Wbsm}\ E\ E'.\\ \qquad (\texttt{STRONG\_EQUIV O relinv}\ \mathit{Wbsm}\ \texttt{O WEAK\_EQUIV})\ E\ E' \iff\\ \qquad (\texttt{WEAK\_EQUIV O}\ \mathit{Wbsm}\ \texttt{O STRONG\_EQUIV})\ E'\ E\end{array}$$

[IDENTITY_OBS_BISIM_UPTO]

$$\vdash \texttt{OBS\_BISIM\_UPTO (=)}$$

[IDENTITY_STRONG_BISIM_UPTO]

$$\vdash \texttt{STRONG\_BISIM\_UPTO (=)}$$

[IDENTITY_STRONG_BISIM_UPTO_lemma]

$$\vdash \forall\, E.\ (\texttt{STRONG\_EQUIV O (=) O STRONG\_EQUIV})\ E\ E$$

[IDENTITY_WEAK_BISIM_UPTO]

$$\vdash \texttt{WEAK\_BISIM\_UPTO (=)}$$

[IDENTITY_WEAK_BISIM_UPTO_lemma]

$$\vdash \forall\, E.\ (\texttt{WEAK\_EQUIV O (=) O STRONG\_EQUIV})\ E\ E$$

[IDENTITY_WEAK_BISIM_UPTO_lemma’]

$$\vdash \forall\, E.\ (\texttt{STRONG\_EQUIV O (=) O WEAK\_EQUIV})\ E\ E$$

[OBS_BISIM_UPTO_EPS]

$$\begin{array}{l}\vdash \forall\, \mathit{Obsm}.\\ \qquad \texttt{OBS\_BISIM\_UPTO}\ \mathit{Obsm} \Longrightarrow\\ \qquad \forall\, E\ E'.\\ \qquad\quad \mathit{Obsm}\ E\ E' \Longrightarrow\\ \qquad\quad \forall\, E_1.\\ \qquad\qquad \texttt{EPS}\ E\ E_1 \Longrightarrow\\ \qquad\qquad \exists\, E_2.\\ \qquad\qquad\quad \texttt{EPS}\ E'\ E_2 \land (\texttt{WEAK\_EQUIV O}\ \mathit{Obsm}\ \texttt{O STRONG\_EQUIV})\ E_1\ E_2\end{array}$$

[OBS_BISIM_UPTO_EPS']

$$
\begin{array}{l}
\vdash \forall \mathit{Obsm}.\\
\quad \texttt{OBS\_BISIM\_UPTO}\ \mathit{Obsm} \Longrightarrow\\
\quad \forall E\ E'.\\
\quad\quad \mathit{Obsm}\ E\ E' \Longrightarrow\\
\quad\quad \forall E_2.\\
\quad\quad\quad \texttt{EPS}\ E'\ E_2 \Longrightarrow\\
\quad\quad\quad \exists E_1.\\
\quad\quad\quad\quad \texttt{EPS}\ E\ E_1 \land (\texttt{STRONG\_EQUIV}\ \texttt{O}\ \mathit{Obsm}\ \texttt{O}\ \texttt{WEAK\_EQUIV})\ E_1\ E_2
\end{array}
$$

[OBS_BISIM_UPTO_THM]

$$
\vdash \forall \mathit{Obsm}.\ \texttt{OBS\_BISIM\_UPTO}\ \mathit{Obsm} \Longrightarrow \mathit{Obsm}\ \texttt{RSUBSET}\ \texttt{OBS\_CONGR}
$$

[OBS_BISIM_UPTO_TRANS]

$$
\begin{array}{l}
\vdash \forall \mathit{Obsm}.\\
\quad \texttt{OBS\_BISIM\_UPTO}\ \mathit{Obsm} \Longrightarrow\\
\quad \forall E\ E'.\\
\quad\quad \mathit{Obsm}\ E\ E' \Longrightarrow\\
\quad\quad \forall u\ E_1.\\
\quad\quad\quad E\ -u\!\rightarrow\ E_1 \Longrightarrow\\
\quad\quad\quad \exists E_2.\\
\quad\quad\quad\quad E'\ =\!u\!\Rightarrow\ E_2 \land (\texttt{WEAK\_EQUIV}\ \texttt{O}\ \mathit{Obsm}\ \texttt{O}\ \texttt{STRONG\_EQUIV})\ E_1\ E_2
\end{array}
$$

[OBS_BISIM_UPTO_TRANS']

$$
\begin{array}{l}
\vdash \forall \mathit{Obsm}.\\
\quad \texttt{OBS\_BISIM\_UPTO}\ \mathit{Obsm} \Longrightarrow\\
\quad \forall E\ E'.\\
\quad\quad \mathit{Obsm}\ E\ E' \Longrightarrow\\
\quad\quad \forall u\ E_2.\\
\quad\quad\quad E'\ -u\!\rightarrow\ E_2 \Longrightarrow\\
\quad\quad\quad \exists E_1.\\
\quad\quad\quad\quad E\ =\!u\!\Rightarrow\ E_1 \land (\texttt{STRONG\_EQUIV}\ \texttt{O}\ \mathit{Obsm}\ \texttt{O}\ \texttt{WEAK\_EQUIV})\ E_1\ E_2
\end{array}
$$

[OBS_BISIM_UPTO_WEAK_TRANS_label]

$$
\begin{array}{l}
\vdash \forall \mathit{Obsm}.\\
\quad \texttt{OBS\_BISIM\_UPTO}\ \mathit{Obsm} \Longrightarrow\\
\quad \forall E\ E'.\\
\quad\quad \mathit{Obsm}\ E\ E' \Longrightarrow\\
\quad\quad \forall l\ E_1.\\
\quad\quad\quad E\ =\!\texttt{label}\ l\!\Rightarrow\ E_1 \Longrightarrow\\
\quad\quad\quad \exists E_2.\\
\quad\quad\quad\quad E'\ =\!\texttt{label}\ l\!\Rightarrow\ E_2 \land\\
\quad\quad\quad\quad (\texttt{WEAK\_EQUIV}\ \texttt{O}\ \mathit{Obsm}\ \texttt{O}\ \texttt{STRONG\_EQUIV})\ E_1\ E_2
\end{array}
$$

[OBS_BISIM_UPTO_WEAK_TRANS_label']

```
⊢ ∀ Obsm.
      OBS_BISIM_UPTO Obsm ⟹
      ∀ E E'.
        Obsm E E' ⟹
        ∀ l E2.
          E' =label l⇒ E2 ⟹
          ∃ E1.
            E =label l⇒ E1 ∧
            (STRONG_EQUIV O Obsm O WEAK_EQUIV) E1 E2
```

[STRONG_BISIM_UPTO_LEMMA]

```
⊢ ∀ Bsm.
      STRONG_BISIM_UPTO Bsm ⟹
      STRONG_BISIM (STRONG_EQUIV O Bsm O STRONG_EQUIV)
```

[STRONG_BISIM_UPTO_THM]

```
⊢ ∀ Bsm. STRONG_BISIM_UPTO Bsm ⟹ Bsm RSUBSET STRONG_EQUIV
```

[WEAK_BISIM_UPTO_ALT_EPS]

```
⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO_ALT Wbsm ⟹
      ∀ E E'.
        Wbsm E E' ⟹
        ∀ E1.
          EPS E E1 ⟹
          ∃ E2. EPS E' E2 ∧ (WEAK_EQUIV O Wbsm O WEAK_EQUIV) E1 E2
```

[WEAK_BISIM_UPTO_ALT_EPS']

```
⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO_ALT Wbsm ⟹
      ∀ E E'.
        Wbsm E E' ⟹
        ∀ E2.
          EPS E' E2 ⟹
          ∃ E1. EPS E E1 ∧ (WEAK_EQUIV O Wbsm O WEAK_EQUIV) E1 E2
```

[WEAK_BISIM_UPTO_ALT_LEMMA]

```
⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO_ALT Wbsm ⟹
      WEAK_BISIM (WEAK_EQUIV O Wbsm O WEAK_EQUIV)
```

[WEAK_BISIM_UPTO_ALT_THM]

```
⊢ ∀ Wbsm. WEAK_BISIM_UPTO_ALT Wbsm ⟹ Wbsm RSUBSET WEAK_EQUIV
```

```
[WEAK_BISIM_UPTO_ALT_WEAK_TRANS_label]
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO_ALT Wbsm ⟹
      ∀ E E'.
        Wbsm E E' ⟹
        ∀ l E1.
          E =label l⇒ E1 ⟹
          ∃ E2.
            E' =label l⇒ E2 ∧
            (WEAK_EQUIV O Wbsm O WEAK_EQUIV) E1 E2
[WEAK_BISIM_UPTO_ALT_WEAK_TRANS_label']
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO_ALT Wbsm ⟹
      ∀ E E'.
        Wbsm E E' ⟹
        ∀ l E2.
          E' =label l⇒ E2 ⟹
          ∃ E1.
            E =label l⇒ E1 ∧
            (WEAK_EQUIV O Wbsm O WEAK_EQUIV) E1 E2
[WEAK_BISIM_UPTO_ALT_WEAK_TRANS_tau]
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO_ALT Wbsm ⟹
      ∀ E E'.
        Wbsm E E' ⟹
        ∀ E1.
          E =τ⇒ E1 ⟹
          ∃ E2. EPS E' E2 ∧ (WEAK_EQUIV O Wbsm O WEAK_EQUIV) E1 E2
[WEAK_BISIM_UPTO_ALT_WEAK_TRANS_tau']
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO_ALT Wbsm ⟹
      ∀ E E'.
        Wbsm E E' ⟹
        ∀ E2.
          E' =τ⇒ E2 ⟹
          ∃ E1. EPS E E1 ∧ (WEAK_EQUIV O Wbsm O WEAK_EQUIV) E1 E2
[WEAK_BISIM_UPTO_EPS]
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO Wbsm ⟹
      ∀ E E'.
        Wbsm E E' ⟹
        ∀ E1.
          EPS E E1 ⟹
          ∃ E2.
            EPS E' E2 ∧ (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E2
```

```
[WEAK_BISIM_UPTO_EPS']
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO Wbsm ⟹
      ∀ E E′.
        Wbsm E E′ ⟹
        ∀ E2.
          EPS E′ E2 ⟹
          ∃ E1.
            EPS E E1 ∧ (STRONG_EQUIV O Wbsm O WEAK_EQUIV) E1 E2
```

```
[WEAK_BISIM_UPTO_LEMMA]
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO Wbsm ⟹
      WEAK_BISIM (WEAK_EQUIV O Wbsm O WEAK_EQUIV)
```

```
[WEAK_BISIM_UPTO_THM]
 ⊢ ∀ Wbsm. WEAK_BISIM_UPTO Wbsm ⟹ Wbsm RSUBSET WEAK_EQUIV
```

```
[WEAK_BISIM_UPTO_TRANS_label]
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO Wbsm ⟹
      ∀ E E′.
        Wbsm E E′ ⟹
        ∀ l E1.
          E −label l→ E1 ⟹
          ∃ E2.
            E′ =label l⇒ E2 ∧
            (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E2
```

```
[WEAK_BISIM_UPTO_TRANS_label']
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO Wbsm ⟹
      ∀ E E′.
        Wbsm E E′ ⟹
        ∀ l E2.
          E′ −label l→ E2 ⟹
          ∃ E1.
            E =label l⇒ E1 ∧
            (STRONG_EQUIV O Wbsm O WEAK_EQUIV) E1 E2
```

```
[WEAK_BISIM_UPTO_TRANS_tau]
 ⊢ ∀ Wbsm.
      WEAK_BISIM_UPTO Wbsm ⟹
      ∀ E E′.
        Wbsm E E′ ⟹
        ∀ E1.
          E −τ→ E1 ⟹
          ∃ E2.
            EPS E′ E2 ∧ (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E2
```

[WEAK_BISIM_UPTO_TRANS_tau']

$$\begin{array}{l}
\vdash \forall \mathit{Wbsm}.\\
\qquad \texttt{WEAK\_BISIM\_UPTO}\ \mathit{Wbsm} \Longrightarrow\\
\qquad \forall E\ E'.\\
\qquad\quad \mathit{Wbsm}\ E\ E' \Longrightarrow\\
\qquad\quad \forall E_2.\\
\qquad\qquad E'\ {-\tau\rightarrow}\ E_2 \Longrightarrow\\
\qquad\qquad \exists E_1.\\
\qquad\qquad\quad \texttt{EPS}\ E\ E_1 \land (\texttt{STRONG\_EQUIV O}\ \mathit{Wbsm}\ \texttt{O WEAK\_EQUIV})\ E_1\ E_2
\end{array}$$

[WEAK_BISIM_UPTO_WEAK_TRANS_label]

$$\begin{array}{l}
\vdash \forall \mathit{Wbsm}.\\
\qquad \texttt{WEAK\_BISIM\_UPTO}\ \mathit{Wbsm} \Longrightarrow\\
\qquad \forall E\ E'.\\
\qquad\quad \mathit{Wbsm}\ E\ E' \Longrightarrow\\
\qquad\quad \forall l\ E_1.\\
\qquad\qquad E\ {=}\texttt{label}\ l{\Rightarrow}\ E_1 \Longrightarrow\\
\qquad\qquad \exists E_2.\\
\qquad\qquad\quad E'\ {=}\texttt{label}\ l{\Rightarrow}\ E_2 \land\\
\qquad\qquad\quad (\texttt{WEAK\_EQUIV O}\ \mathit{Wbsm}\ \texttt{O STRONG\_EQUIV})\ E_1\ E_2
\end{array}$$

[WEAK_BISIM_UPTO_WEAK_TRANS_label']

$$\begin{array}{l}
\vdash \forall \mathit{Wbsm}.\\
\qquad \texttt{WEAK\_BISIM\_UPTO}\ \mathit{Wbsm} \Longrightarrow\\
\qquad \forall E\ E'.\\
\qquad\quad \mathit{Wbsm}\ E\ E' \Longrightarrow\\
\qquad\quad \forall l\ E_2.\\
\qquad\qquad E'\ {=}\texttt{label}\ l{\Rightarrow}\ E_2 \Longrightarrow\\
\qquad\qquad \exists E_1.\\
\qquad\qquad\quad E\ {=}\texttt{label}\ l{\Rightarrow}\ E_1 \land\\
\qquad\qquad\quad (\texttt{STRONG\_EQUIV O}\ \mathit{Wbsm}\ \texttt{O WEAK\_EQUIV})\ E_1\ E_2
\end{array}$$

9.11 Trace Theory

Built: 08 Dicembre 2017
Parent Theories: WeakEQ

9.11.1 Definitions

[finite_state_def]

$$\vdash \forall p.\ \texttt{finite\_state}\ p \iff \texttt{FINITE (NODES}\ p\texttt{)}$$

[LRTC_DEF]

$$\begin{array}{l}
\vdash \forall R\ a\ l\ b.\\
\quad\quad \texttt{LRTC}\ R\ a\ l\ b \iff\\
\quad\quad \forall P.\\
\quad\quad\quad (\forall x.\ P\ x\ \texttt{[]}\ x) \land\\
\quad\quad\quad (\forall x\ h\ y\ t\ z.\ R\ x\ h\ y \land P\ y\ t\ z \Longrightarrow P\ x\ (h\texttt{::}t)\ z) \Longrightarrow\\
\quad\quad\quad P\ a\ l\ b
\end{array}$$

[NO_LABEL_def]

$$\vdash \forall L.\ \texttt{NO\_LABEL}\ L \iff \neg\exists l.\ \texttt{MEM (label}\ l\texttt{)}\ L$$

[NODES_def]

$$\vdash \forall p.\ \texttt{NODES}\ p\ \texttt{=}\ \{q\ \texttt{|}\ \texttt{Reach}\ p\ q\}$$

[Reach_def]

$$\vdash \texttt{Reach = (}\lambda E\ E'.\ \exists u.\ E\ {-u\rightarrow}\ E'\texttt{)}^*$$

[STEP_def]

$$\vdash \forall P\ n\ Q.\ \texttt{STEP}\ P\ n\ Q \iff \texttt{NRC (}\lambda E\ E'.\ \exists u.\ E\ {-u\rightarrow}\ E'\texttt{)}\ n\ P\ Q$$

[TRACE_def]

$$\vdash \texttt{TRACE = LRTC TRANS}$$

[UNIQUE_LABEL_def]

$$\begin{array}{l}
\vdash \forall u\ L.\\
\quad\quad \texttt{UNIQUE\_LABEL}\ u\ L \iff\\
\quad\quad \exists L_1\ L_2.\\
\quad\quad\quad (L_1\ \texttt{++ [}u\texttt{] ++}\ L_2\ \texttt{=}\ L) \land\\
\quad\quad\quad \neg\exists l.\ \texttt{MEM (label}\ l\texttt{)}\ L_1 \lor \texttt{MEM (label}\ l\texttt{)}\ L_2
\end{array}$$

9.11.2 Theorems

[EPS_AND_STEP]

$$\vdash \forall E\ E'.\ \texttt{EPS}\ E\ E' \Longrightarrow \exists n.\ \texttt{STEP}\ E\ n\ E'$$

[EPS_AND_TRACE]

$$\vdash \forall E\ E'.\ \texttt{EPS}\ E\ E' \iff \exists xs.\ \texttt{TRACE}\ E\ xs\ E' \land \texttt{NO\_LABEL}\ xs$$

[EPS_IN_NODES]

$$\vdash \forall p\ q.\ \texttt{EPS}\ p\ q \Longrightarrow q \in \texttt{NODES}\ p$$

[EPS_Reach]

$$\vdash \forall p\ q.\ \texttt{EPS}\ p\ q \Longrightarrow \texttt{Reach}\ p\ q$$

[EPS_TRACE]

$$\vdash \forall E\ E'.\ \mathtt{EPS}\ E\ E' \Longrightarrow \exists xs.\ \mathtt{TRACE}\ E\ xs\ E'$$

[LRTC_APPEND_CASES]

$$\begin{array}{l}\vdash \forall R\ l_1\ l_2\ x\ y.\\ \quad \mathtt{LRTC}\ R\ x\ (l_1\ \mathtt{++}\ l_2)\ y \iff \exists u.\ \mathtt{LRTC}\ R\ x\ l_1\ u \wedge \mathtt{LRTC}\ R\ u\ l_2\ y\end{array}$$

[LRTC_CASES1]

$$\begin{array}{l}\vdash \forall R\ x\ l\ y.\\ \quad \mathtt{LRTC}\ R\ x\ l\ y \iff\\ \quad \textbf{if}\ \mathtt{NULL}\ l\ \textbf{then}\ x = y\\ \quad \textbf{else}\ \exists u.\ R\ x\ (\mathtt{HD}\ l)\ u \wedge \mathtt{LRTC}\ R\ u\ (\mathtt{TL}\ l)\ y\end{array}$$

[LRTC_CASES2]

$$\begin{array}{l}\vdash \forall R\ x\ l\ y.\\ \quad \mathtt{LRTC}\ R\ x\ l\ y \iff\\ \quad \textbf{if}\ \mathtt{NULL}\ l\ \textbf{then}\ x = y\\ \quad \textbf{else}\ \exists u.\ \mathtt{LRTC}\ R\ x\ (\mathtt{FRONT}\ l)\ u \wedge R\ u\ (\mathtt{LAST}\ l)\ y\end{array}$$

[LRTC_CASES_LRTC_TWICE]

$$\begin{array}{l}\vdash \forall R\ x\ l\ y.\\ \quad \mathtt{LRTC}\ R\ x\ l\ y \iff\\ \quad \exists u\ l_1\ l_2.\ \mathtt{LRTC}\ R\ x\ l_1\ u \wedge \mathtt{LRTC}\ R\ u\ l_2\ y \wedge (l = l_1\ \mathtt{++}\ l_2)\end{array}$$

[LRTC_INDUCT]

$$\begin{array}{l}\vdash \forall R\ P.\\ \quad (\forall x.\ P\ x\ []\ x) \wedge\\ \quad (\forall x\ h\ y\ t\ z.\ R\ x\ h\ y \wedge P\ y\ t\ z \Longrightarrow P\ x\ (h::t)\ z) \Longrightarrow\\ \quad \forall x\ l\ y.\ \mathtt{LRTC}\ R\ x\ l\ y \Longrightarrow P\ x\ l\ y\end{array}$$

[LRTC_LRTC]

$$\begin{array}{l}\vdash \forall R\ x\ m\ y.\\ \quad \mathtt{LRTC}\ R\ x\ m\ y \Longrightarrow \forall n\ z.\ \mathtt{LRTC}\ R\ y\ n\ z \Longrightarrow \mathtt{LRTC}\ R\ x\ (m\ \mathtt{++}\ n)\ z\end{array}$$

[LRTC_NIL]

$$\vdash \forall R\ x\ y.\ \mathtt{LRTC}\ R\ x\ []\ y \iff (x = y)$$

[LRTC_ONE]

$$\vdash \forall R\ x\ t\ y.\ \mathtt{LRTC}\ R\ x\ [t]\ y \iff R\ x\ t\ y$$

[LRTC_REFL]

$$\vdash \forall R.\ \mathtt{LRTC}\ R\ x\ []\ x$$

[LRTC_RULES]

$$\begin{array}{l}\vdash \forall R.\\ \quad (\forall x.\ \mathtt{LRTC}\ R\ x\ \mathtt{[]}\ x)\ \land\\ \quad \forall x\ h\ y\ t\ z.\ R\ x\ h\ y \land \mathtt{LRTC}\ R\ y\ t\ z \implies \mathtt{LRTC}\ R\ x\ (h\mathtt{::}t)\ z\end{array}$$

[LRTC_SINGLE]

$$\vdash \forall R\ x\ t\ y.\ R\ x\ t\ y \implies \mathtt{LRTC}\ R\ x\ [t]\ y$$

[LRTC_STRONG_INDUCT]

$$\begin{array}{l}\vdash \forall R\ P.\\ \quad (\forall x.\ P\ x\ \mathtt{[]}\ x)\ \land\\ \quad (\forall x\ h\ y\ t\ z.\\ \qquad R\ x\ h\ y \land \mathtt{LRTC}\ R\ y\ t\ z \land P\ y\ t\ z \implies P\ x\ (h\mathtt{::}t)\ z) \implies\\ \quad \forall x\ l\ y.\ \mathtt{LRTC}\ R\ x\ l\ y \implies P\ x\ l\ y\end{array}$$

[LRTC_TRANS]

$$\begin{array}{l}\vdash \forall R\ x\ m\ y\ n\ z.\\ \quad \mathtt{LRTC}\ R\ x\ m\ y \land \mathtt{LRTC}\ R\ y\ n\ z \implies \mathtt{LRTC}\ R\ x\ (m\ \mathtt{++}\ n)\ z\end{array}$$

[MORE_NODES]

$$\vdash \forall p\ q\ q'.\ q \in \mathtt{NODES}\ p \land \mathtt{Reach}\ q\ q' \implies q' \in \mathtt{NODES}\ p$$

[NO_LABEL_cases]

$$\vdash \forall x\ xs.\ \mathtt{NO\_LABEL}\ (x\mathtt{::}xs) \iff (x = \tau) \land \mathtt{NO\_LABEL}\ xs$$

[Reach_cases1]

$$\vdash \forall x\ y.\ \mathtt{Reach}\ x\ y \iff (x = y) \lor \exists u.\ (\exists u'.\ x\ {-u'\!\rightarrow}\ u) \land \mathtt{Reach}\ u\ y$$

[Reach_cases2]

$$\vdash \forall x\ y.\ \mathtt{Reach}\ x\ y \iff (x = y) \lor \exists u.\ \mathtt{Reach}\ x\ u \land \exists u'.\ u\ {-u'\!\rightarrow}\ y$$

[Reach_ind]

$$\begin{array}{l}\vdash \forall P.\\ \quad (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ (\exists u.\ x\ {-u\!\rightarrow}\ y) \land P\ y\ z \implies P\ x\ z) \implies\\ \quad \forall x\ y.\ \mathtt{Reach}\ x\ y \implies P\ x\ y\end{array}$$

[Reach_ind_right]

$$\begin{array}{l}\vdash \forall P.\\ \quad (\forall x.\ P\ x\ x) \land (\forall x\ y\ z.\ P\ x\ y \land (\exists u.\ y\ {-u\!\rightarrow}\ z) \implies P\ x\ z) \implies\\ \quad \forall x\ y.\ \mathtt{Reach}\ x\ y \implies P\ x\ y\end{array}$$

[Reach_NODES]

$$\vdash \forall p\ q.\ \mathtt{Reach}\ p\ q \implies q \in \mathtt{NODES}\ p$$

[Reach_one]

$\vdash \forall E\ E'.\ (\exists u.\ E\ -u\rightarrow\ E') \Longrightarrow \texttt{Reach}\ E\ E'$

[Reach_self]

$\vdash \forall E.\ \texttt{Reach}\ E\ E$

[Reach_strongind]

$$\begin{aligned}
&\vdash \forall P.\\
&\qquad (\forall x.\ P\ x\ x) \land\\
&\qquad (\forall x\ y\ z.\ (\exists u.\ x\ -u\rightarrow\ y) \land \texttt{Reach}\ y\ z \land P\ y\ z \Longrightarrow P\ x\ z) \Longrightarrow\\
&\qquad \forall x\ y.\ \texttt{Reach}\ x\ y \Longrightarrow P\ x\ y
\end{aligned}$$

[Reach_strongind_right]

$$\begin{aligned}
&\vdash \forall P.\\
&\qquad (\forall x.\ P\ x\ x) \land\\
&\qquad (\forall x\ y\ z.\ P\ x\ y \land \texttt{Reach}\ x\ y \land (\exists u.\ y\ -u\rightarrow\ z) \Longrightarrow P\ x\ z) \Longrightarrow\\
&\qquad \forall x\ y.\ \texttt{Reach}\ x\ y \Longrightarrow P\ x\ y
\end{aligned}$$

[Reach_trans]

$\vdash \forall x\ y\ z.\ \texttt{Reach}\ x\ y \land \texttt{Reach}\ y\ z \Longrightarrow \texttt{Reach}\ x\ z$

[SELF_NODES]

$\vdash \forall p.\ p \in \texttt{NODES}\ p$

[STEP0]

$\vdash \forall x\ y.\ \texttt{STEP}\ x\ 0\ y \iff (x = y)$

[STEP1]

$\vdash \texttt{STEP}\ x\ 1\ y \iff \exists u.\ x\ -u\rightarrow\ y$

[STEP_ADD_E]

$\vdash \forall m\ n\ x\ z.\ \texttt{STEP}\ x\ (m + n)\ z \Longrightarrow \exists y.\ \texttt{STEP}\ x\ m\ y \land \texttt{STEP}\ y\ n\ z$

[STEP_ADD_EQN]

$\vdash \forall m\ n\ x\ z.\ \texttt{STEP}\ x\ (m + n)\ z \iff \exists y.\ \texttt{STEP}\ x\ m\ y \land \texttt{STEP}\ y\ n\ z$

[STEP_ADD_I]

$\vdash \forall m\ n\ x\ y\ z.\ \texttt{STEP}\ x\ m\ y \land \texttt{STEP}\ y\ n\ z \Longrightarrow \texttt{STEP}\ x\ (m + n)\ z$

[STEP_SUC]

$\vdash \forall n\ x\ y.\ \texttt{STEP}\ x\ (\texttt{SUC}\ n)\ y \iff \exists z.\ (\exists u.\ x\ -u\rightarrow\ z) \land \texttt{STEP}\ z\ n\ y$

[STEP_SUC_LEFT]

$\vdash \forall n\ x\ y.\ \texttt{STEP}\ x\ (\texttt{SUC}\ n)\ y \iff \exists z.\ \texttt{STEP}\ x\ n\ z \land \exists u.\ z\ -u\rightarrow\ y$

[TRACE1]

$\vdash \forall x\ t\ y.\ x\ {-t\rightarrow}\ y \Longrightarrow \texttt{TRACE}\ x\ [t]\ y$

[TRACE2]

$\vdash \forall E\ x\ E_1\ xs\ E'.\ E\ {-x\rightarrow}\ E_1 \land \texttt{TRACE}\ E_1\ xs\ E' \Longrightarrow \texttt{TRACE}\ E\ (x::xs)\ E'$

[TRACE_APPEND_cases]

$\vdash \forall l_1\ l_2\ x\ y.$
$\quad \texttt{TRACE}\ x\ (l_1 \mathbin{+\!+} l_2)\ y \iff \exists u.\ \texttt{TRACE}\ x\ l_1\ u \land \texttt{TRACE}\ u\ l_2\ y$

[TRACE_cases1]

$\vdash \forall x\ l\ y.$
$\quad \texttt{TRACE}\ x\ l\ y \iff$
$\quad \textbf{if}\ \texttt{NULL}\ l\ \textbf{then}\ x = y\ \textbf{else}\ \exists u.\ x\ {-\texttt{HD}\ l\rightarrow}\ u \land \texttt{TRACE}\ u\ (\texttt{TL}\ l)\ y$

[TRACE_cases2]

$\vdash \forall x\ l\ y.$
$\quad \texttt{TRACE}\ x\ l\ y \iff$
$\quad \textbf{if}\ \texttt{NULL}\ l\ \textbf{then}\ x = y$
$\quad \textbf{else}\ \exists u.\ \texttt{TRACE}\ x\ (\texttt{FRONT}\ l)\ u \land u\ {-\texttt{LAST}\ l\rightarrow}\ y$

[TRACE_cases_twice]

$\vdash \forall x\ l\ y.$
$\quad \texttt{TRACE}\ x\ l\ y \iff$
$\quad \exists u\ l_1\ l_2.\ \texttt{TRACE}\ x\ l_1\ u \land \texttt{TRACE}\ u\ l_2\ y \land (l = l_1 \mathbin{+\!+} l_2)$

[TRACE_ind]

$\vdash \forall P.$
$\quad (\forall x.\ P\ x\ \epsilon\ x) \land$
$\quad (\forall x\ h\ y\ t\ z.\ x\ {-h\rightarrow}\ y \land P\ y\ t\ z \Longrightarrow P\ x\ (h::t)\ z) \Longrightarrow$
$\quad \forall x\ l\ y.\ \texttt{TRACE}\ x\ l\ y \Longrightarrow P\ x\ l\ y$

[TRACE_NIL]

$\vdash \forall x\ y.\ \texttt{TRACE}\ x\ \epsilon\ y \iff (x = y)$

[TRACE_ONE]

$\vdash \forall x\ t\ y.\ \texttt{TRACE}\ x\ [t]\ y \iff x\ {-t\rightarrow}\ y$

[TRACE_REFL]

$\vdash \forall E.\ \texttt{TRACE}\ E\ \epsilon\ E$

[TRACE_rule0]

$\vdash \forall x.\ \texttt{TRACE}\ x\ \epsilon\ x$

[TRACE_rule1]

$$\vdash \forall x\ h\ y\ t\ z.\ x\ {-h\!\rightarrow}\ y \land \texttt{TRACE}\ y\ t\ z \Longrightarrow \texttt{TRACE}\ x\ (h\texttt{::}t)\ z$$

[TRACE_rules]

$$\begin{array}{l} \vdash (\forall x.\ \texttt{TRACE}\ x\ \epsilon\ x) \land \\ \quad \forall x\ h\ y\ t\ z.\ x\ {-h\!\rightarrow}\ y \land \texttt{TRACE}\ y\ t\ z \Longrightarrow \texttt{TRACE}\ x\ (h\texttt{::}t)\ z \end{array}$$

[TRACE_strongind]

$$\begin{array}{l} \vdash \forall P. \\ \qquad (\forall x.\ P\ x\ \epsilon\ x) \land \\ \qquad (\forall x\ h\ y\ t\ z. \\ \qquad\quad x\ {-h\!\rightarrow}\ y \land \texttt{TRACE}\ y\ t\ z \land P\ y\ t\ z \Longrightarrow P\ x\ (h\texttt{::}t)\ z) \Longrightarrow \\ \qquad \forall x\ l\ y.\ \texttt{TRACE}\ x\ l\ y \Longrightarrow P\ x\ l\ y \end{array}$$

[TRACE_trans]

$$\vdash \forall x\ m\ y.\ \texttt{TRACE}\ x\ m\ y \Longrightarrow \forall n\ z.\ \texttt{TRACE}\ y\ n\ z \Longrightarrow \texttt{TRACE}\ x\ (m\ \texttt{++}\ n)\ z$$

[TRACE_trans_applied]

$$\begin{array}{l} \vdash \forall xs\ ys\ E\ E_1\ E'. \\ \qquad \texttt{TRACE}\ E\ xs\ E_1 \land \texttt{TRACE}\ E_1\ ys\ E' \Longrightarrow \texttt{TRACE}\ E\ (xs\ \texttt{++}\ ys)\ E' \end{array}$$

[TRANS_IN_NODES]

$$\vdash \forall p\ q\ u.\ p\ {-u\!\rightarrow}\ q \Longrightarrow q \in \texttt{NODES}\ p$$

[UNIQUE_LABEL_cases1]

$$\begin{array}{l} \vdash \forall l\ xs. \\ \qquad \texttt{UNIQUE\_LABEL (label}\ l\texttt{)}\ (\tau\texttt{::}xs) \iff \texttt{UNIQUE\_LABEL (label}\ l\texttt{)}\ xs \end{array}$$

[UNIQUE_LABEL_cases2]

$$\begin{array}{l} \vdash \forall l\ l'\ xs. \\ \qquad \texttt{UNIQUE\_LABEL (label}\ l\texttt{) (label}\ l'\texttt{::}xs) \iff \\ \qquad (l = l') \land \texttt{NO\_LABEL}\ xs \end{array}$$

[UNIQUE_LABEL_IMP_MEM]

$$\vdash \forall u\ L.\ \texttt{UNIQUE\_LABEL}\ u\ L \Longrightarrow \texttt{MEM}\ u\ L$$

[UNIQUE_LABEL_NOT_NULL]

$$\vdash \forall u\ L.\ \texttt{UNIQUE\_LABEL}\ u\ L \Longrightarrow \neg\texttt{NULL}\ L$$

[WEAK_TRANS_AND_STEP]

$$\vdash \forall E\ u\ E'.\ E\ {=u\!\Rightarrow}\ E' \Longrightarrow \exists n.\ \texttt{STEP}\ E\ n\ E'$$

[WEAK_TRANS_AND_TRACE]

$$\begin{array}{l}
\vdash \forall E\ \ u\ \ E'.\\
\qquad E\ \ {=}u{\Rightarrow}\ \ E'\ \iff\\
\qquad \exists\, us.\\
\qquad\quad \texttt{TRACE}\ \ E\ \ us\ \ E'\ \land\ \neg\texttt{NULL}\ \ us\ \land\\
\qquad\quad \textsf{if}\ \ u\ =\ \tau\ \ \textsf{then}\ \ \texttt{NO\_LABEL}\ \ us\ \ \textsf{else}\ \ \texttt{UNIQUE\_LABEL}\ \ u\ \ us
\end{array}$$

[WEAK_TRANS_IN_NODES]

$$\vdash \forall p\ \ q\ \ u.\ \ p\ \ {=}u{\Rightarrow}\ \ q\ \Longrightarrow\ q\ \in\ \texttt{NODES}\ \ p$$

[WEAK_TRANS_Reach]

$$\vdash \forall p\ \ q\ \ u.\ \ p\ \ {=}u{\Rightarrow}\ \ q\ \Longrightarrow\ \texttt{Reach}\ \ p\ \ q$$

[WEAK_TRANS_TRACE]

$$\vdash \forall E\ \ u\ \ E'.\ \ E\ \ {=}u{\Rightarrow}\ \ E'\ \Longrightarrow\ \exists\, xs.\ \ \texttt{TRACE}\ \ E\ \ xs\ \ E'$$

9.12 Expansion Theory

Built: 08 Dicembre 2017
Parent Theories: Trace, Congruence

9.12.1 Definitions

[expands_def]

$$\begin{array}{l}
\vdash \texttt{(expands)}\ =\\
\quad (\lambda\, a_0\ \ a_1.\\
\qquad \exists\, expands'.\\
\qquad\quad expands'\ \ a_0\ \ a_1\ \land\\
\qquad\quad \forall\, a_0\ \ a_1.\\
\qquad\qquad expands'\ \ a_0\ \ a_1\ \Longrightarrow\\
\qquad\qquad (\forall\, l.\\
\qquad\qquad\quad (\forall\, E_1.\\
\qquad\qquad\qquad a_0\ \ {-}\texttt{label}\ \ l{\rightarrow}\ \ E_1\ \Longrightarrow\\
\qquad\qquad\qquad \exists\, E_2.\ \ a_1\ \ {-}\texttt{label}\ \ l{\rightarrow}\ \ E_2\ \land\ expands'\ \ E_1\ \ E_2)\ \land\\
\qquad\qquad\quad \forall\, E_2.\\
\qquad\qquad\qquad a_1\ \ {-}\texttt{label}\ \ l{\rightarrow}\ \ E_2\ \Longrightarrow\\
\qquad\qquad\qquad \exists\, E_1.\ \ a_0\ \ {=}\texttt{label}\ \ l{\Rightarrow}\ \ E_1\ \land\ expands'\ \ E_1\ \ E_2)\ \land\\
\qquad\qquad (\forall\, E_1.\\
\qquad\qquad\quad a_0\ \ {-}\tau{\rightarrow}\ \ E_1\ \Longrightarrow\\
\qquad\qquad\quad expands'\ \ E_1\ \ a_1\ \lor\ \exists\, E_2.\ \ a_1\ \ {-}\tau{\rightarrow}\ \ E_2\ \land\ expands'\ \ E_1\ \ E_2)\ \land\\
\qquad\qquad \forall\, E_2.\ \ a_1\ \ {-}\tau{\rightarrow}\ \ E_2\ \Longrightarrow\ \exists\, E_1.\ \ a_0\ \ {=}\tau{\Rightarrow}\ \ E_1\ \land\ expands'\ \ E_1\ \ E_2)
\end{array}$$

[EXPANSION]

$$
\begin{array}{l}
\vdash \forall \mathit{Exp}.\\
\quad \texttt{EXPANSION}\ \mathit{Exp} \iff\\
\quad \forall E\ E'.\\
\quad\quad \mathit{Exp}\ E\ E' \implies\\
\quad\quad (\forall l.\\
\quad\quad\quad (\forall E_1.\\
\quad\quad\quad\quad E\ -\texttt{label}\ l\rightarrow\ E_1 \implies\\
\quad\quad\quad\quad \exists E_2.\ E'\ -\texttt{label}\ l\rightarrow\ E_2 \land \mathit{Exp}\ E_1\ E_2) \land\\
\quad\quad\quad \forall E_2.\\
\quad\quad\quad\quad E'\ -\texttt{label}\ l\rightarrow\ E_2 \implies \exists E_1.\ E\ =\texttt{label}\ l\Rightarrow\ E_1 \land \mathit{Exp}\ E_1\ E_2) \land\\
\quad\quad (\forall E_1.\\
\quad\quad\quad E\ -\tau\rightarrow\ E_1 \implies \mathit{Exp}\ E_1\ E' \lor \exists E_2.\ E'\ -\tau\rightarrow\ E_2 \land \mathit{Exp}\ E_1\ E_2) \land\\
\quad\quad \forall E_2.\ E'\ -\tau\rightarrow\ E_2 \implies \exists E_1.\ E\ =\tau\Rightarrow\ E_1 \land \mathit{Exp}\ E_1\ E_2
\end{array}
$$

9.12.2 Theorems

[COMP_EXPANSION]

$$
\begin{array}{l}
\vdash \forall \mathit{Exp}_1\ \mathit{Exp}_2.\\
\quad \texttt{EXPANSION}\ \mathit{Exp}_1 \land \texttt{EXPANSION}\ \mathit{Exp}_2 \implies \texttt{EXPANSION}\ (\mathit{Exp}_2\ \texttt{O}\ \mathit{Exp}_1)
\end{array}
$$

[EQ_IMP_expands]

$$
\vdash \forall E\ E'.\ (E = E') \implies E\ \texttt{expands}\ E'
$$

[expands_AND_TRACE_label]

$$
\begin{array}{l}
\vdash \forall E\ E'.\\
\quad E\ \texttt{expands}\ E' \implies\\
\quad \forall \mathit{xs}\ l\ E_1.\\
\quad\quad \texttt{TRACE}\ E\ \mathit{xs}\ E_1 \land \texttt{UNIQUE\_LABEL}\ (\texttt{label}\ l)\ \mathit{xs} \implies\\
\quad\quad \exists \mathit{xs}'\ E_2.\\
\quad\quad\quad \texttt{TRACE}\ E'\ \mathit{xs}'\ E_2 \land E_1\ \texttt{expands}\ E_2 \land\\
\quad\quad\quad \texttt{LENGTH}\ \mathit{xs}' \leq \texttt{LENGTH}\ \mathit{xs} \land \texttt{UNIQUE\_LABEL}\ (\texttt{label}\ l)\ \mathit{xs}'
\end{array}
$$

[expands_AND_TRACE_tau]

$$
\begin{array}{l}
\vdash \forall E\ E'.\\
\quad E\ \texttt{expands}\ E' \implies\\
\quad \forall \mathit{xs}\ l\ E_1.\\
\quad\quad \texttt{TRACE}\ E\ \mathit{xs}\ E_1 \land \texttt{NO\_LABEL}\ \mathit{xs} \implies\\
\quad\quad \exists \mathit{xs}'\ E_2.\\
\quad\quad\quad \texttt{TRACE}\ E'\ \mathit{xs}'\ E_2 \land E_1\ \texttt{expands}\ E_2 \land\\
\quad\quad\quad \texttt{LENGTH}\ \mathit{xs}' \leq \texttt{LENGTH}\ \mathit{xs} \land \texttt{NO\_LABEL}\ \mathit{xs}'
\end{array}
$$

[expands_cases]

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

[expands_cases']

$$\begin{array}{l}
\vdash\ \forall E\ \ E' .\\
\qquad E\ \texttt{expands}\ E'\ \iff\\
\qquad (\forall l\ \ E_1 .\\
\qquad\quad E\ -\texttt{label}\ l\rightarrow\ E_1\ \Longrightarrow\ \exists E_2 .\ E'\ -\texttt{label}\ l\rightarrow\ E_2\ \wedge\ E_1\ \texttt{expands}\ E_2)\ \wedge\\
\qquad (\forall E_1 .\\
\qquad\quad E\ -\tau\rightarrow\ E_1\ \Longrightarrow\\
\qquad\quad E_1\ \texttt{expands}\ E'\ \vee\ \exists E_2 .\ E'\ -\tau\rightarrow\ E_2\ \wedge\ E_1\ \texttt{expands}\ E_2)\ \wedge\\
\qquad \forall u\ \ E_2 .\ E'\ -u\rightarrow\ E_2\ \Longrightarrow\ \exists E_1 .\ E\ =u\Rightarrow\ E_1\ \wedge\ E_1\ \texttt{expands}\ E_2
\end{array}$$

[expands_coind]

$$\begin{array}{l}
\vdash\ \forall \mathit{expands}' .\\
\qquad (\forall a_0\ \ a_1 .\\
\qquad\quad \mathit{expands}'\ a_0\ \ a_1\ \Longrightarrow\\
\qquad\quad (\forall l .\\
\qquad\qquad (\forall E_1 .\\
\qquad\qquad\quad a_0\ -\texttt{label}\ l\rightarrow\ E_1\ \Longrightarrow\\
\qquad\qquad\quad \exists E_2 .\ a_1\ -\texttt{label}\ l\rightarrow\ E_2\ \wedge\ \mathit{expands}'\ E_1\ \ E_2)\ \wedge\\
\qquad\qquad \forall E_2 .\\
\qquad\qquad\ \ a_1\ -\texttt{label}\ l\rightarrow\ E_2\ \Longrightarrow\\
\qquad\qquad\ \ \exists E_1 .\ a_0\ =\texttt{label}\ l\Rightarrow\ E_1\ \wedge\ \mathit{expands}'\ E_1\ \ E_2)\ \wedge\\
\qquad\quad (\forall E_1 .\\
\qquad\qquad a_0\ -\tau\rightarrow\ E_1\ \Longrightarrow\\
\qquad\qquad \mathit{expands}'\ E_1\ \ a_1\ \vee\ \exists E_2 .\ a_1\ -\tau\rightarrow\ E_2\ \wedge\ \mathit{expands}'\ E_1\ \ E_2)\ \wedge\\
\qquad\quad \forall E_2 .\ a_1\ -\tau\rightarrow\ E_2\ \Longrightarrow\ \exists E_1 .\ a_0\ =\tau\Rightarrow\ E_1\ \wedge\ \mathit{expands}'\ E_1\ \ E_2)\ \Longrightarrow\\
\qquad \forall a_0\ \ a_1 .\ \mathit{expands}'\ a_0\ \ a_1\ \Longrightarrow\ a_0\ \texttt{expands}\ a_1
\end{array}$$

[expands_EPS]

$$\begin{array}{l}
\vdash\ \forall E\ \ E' .\\
\qquad E\ \texttt{expands}\ E'\ \Longrightarrow\\
\qquad \forall E_1 .\ \texttt{EPS}\ E\ \ E_1\ \Longrightarrow\ \exists E_2 .\ \texttt{EPS}\ E'\ \ E_2\ \wedge\ E_1\ \texttt{expands}\ E_2
\end{array}$$

[expands_EPS']

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad E\ \texttt{expands}\ E' \Longrightarrow\\
\quad \forall E_2.\ \texttt{EPS}\ E'\ E_2 \Longrightarrow \exists E_1.\ \texttt{EPS}\ E\ E_1 \land E_1\ \texttt{expands}\ E_2
\end{array}$$

[expands_IMP_WEAK_EQUIV]

$$\vdash \forall P\ Q.\ P\ \texttt{expands}\ Q \Longrightarrow \texttt{WEAK\_EQUIV}\ P\ Q$$

[expands_is_EXPANSION]

$$\vdash \texttt{EXPANSION (expands)}$$

[expands_precongruence]

$$\vdash \texttt{precongruence1 (expands)}$$

[expands_PreOrder]

$$\vdash \texttt{PreOrder (expands)}$$

[expands_PRESD_BY_GUARDED_SUM]

$$\begin{array}{l}
\vdash \forall E_1\ E_1'\ E_2\ E_2'\ a_1\ a_2.\\
\quad E_1\ \texttt{expands}\ E_1' \land E_2\ \texttt{expands}\ E_2' \Longrightarrow\\
\quad a_1\texttt{..}E_1\ \texttt{+}\ a_2\texttt{..}E_2\ \texttt{expands}\ (a_1\texttt{..}E_1'\ \texttt{+}\ a_2\texttt{..}E_2')
\end{array}$$

[expands_PRESD_BY_PAR]

$$\begin{array}{l}
\vdash \forall E_1\ E_1'\ E_2\ E_2'.\\
\quad E_1\ \texttt{expands}\ E_1' \land E_2\ \texttt{expands}\ E_2' \Longrightarrow E_1 \parallel E_2\ \texttt{expands}\ E_1' \parallel E_2'
\end{array}$$

[expands_REFL]

$$\vdash \forall x.\ x\ \texttt{expands}\ x$$

[expands_reflexive]

$$\vdash \texttt{reflexive (expands)}$$

[expands_rules]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad (\forall l.\\
\quad\quad (\forall E_1.\\
\quad\quad\quad E\ -\texttt{label}\ l\!\rightarrow E_1 \Longrightarrow\\
\quad\quad\quad \exists E_2.\ E'\ -\texttt{label}\ l\!\rightarrow E_2 \land E_1\ \texttt{expands}\ E_2) \land\\
\quad\quad \forall E_2.\\
\quad\quad\quad E'\ -\texttt{label}\ l\!\rightarrow E_2 \Longrightarrow\\
\quad\quad\quad \exists E_1.\ E\ =\texttt{label}\ l\!\Rightarrow E_1 \land E_1\ \texttt{expands}\ E_2) \land\\
\quad (\forall E_1.\\
\quad\quad E\ -\tau\!\rightarrow E_1 \Longrightarrow\\
\quad\quad E_1\ \texttt{expands}\ E' \lor \exists E_2.\ E'\ -\tau\!\rightarrow E_2 \land E_1\ \texttt{expands}\ E_2) \land\\
\quad (\forall E_2.\ E'\ -\tau\!\rightarrow E_2 \Longrightarrow \exists E_1.\ E\ =\tau\!\Rightarrow E_1 \land E_1\ \texttt{expands}\ E_2) \Longrightarrow\\
\quad E\ \texttt{expands}\ E'
\end{array}$$

[expands_SUBST_GCONTEXT]

$\vdash \forall P\ Q.\ P\ \texttt{expands}\ Q \Longrightarrow \forall E.\ \texttt{GCONTEXT}\ E \Longrightarrow E\ P\ \texttt{expands}\ E\ Q$

[expands_SUBST_PREFIX]

$\vdash \forall E\ E'.\ E\ \texttt{expands}\ E' \Longrightarrow \forall u.\ u..E\ \texttt{expands}\ u..E'$

[expands_SUBST_RELAB]

$\vdash \forall E\ E'.\ E\ \texttt{expands}\ E' \Longrightarrow \forall \mathit{rf}.\ \texttt{relab}\ E\ \mathit{rf}\ \texttt{expands}\ \texttt{relab}\ E'\ \mathit{rf}$

[expands_SUBST_RESTR]

$\vdash \forall E\ E'.\ E\ \texttt{expands}\ E' \Longrightarrow \forall L.\ \nu\ L\ E\ \texttt{expands}\ \nu\ L\ E'$

[expands_thm]

$\vdash \forall P\ Q.\ P\ \texttt{expands}\ Q \iff \exists \mathit{Exp}.\ \mathit{Exp}\ P\ Q \land \texttt{EXPANSION}\ \mathit{Exp}$

[expands_TRANS]

$\vdash \forall x\ y\ z.\ x\ \texttt{expands}\ y \land y\ \texttt{expands}\ z \Longrightarrow x\ \texttt{expands}\ z$

[expands_TRANS_action']

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad E\ \texttt{expands}\ E' \Longrightarrow\\ \quad \forall u\ E_2.\ E'\ {-u\rightarrow}\ E_2 \Longrightarrow \exists E_1.\ E\ {=u\Rightarrow}\ E_1 \land E_1\ \texttt{expands}\ E_2\end{array}$$

[expands_TRANS_label]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad E\ \texttt{expands}\ E' \Longrightarrow\\ \quad \forall l\ E_1.\\ \qquad E\ {-}\texttt{label}\ l{\rightarrow}\ E_1 \Longrightarrow \exists E_2.\ E'\ {-}\texttt{label}\ l{\rightarrow}\ E_2 \land E_1\ \texttt{expands}\ E_2\end{array}$$

[expands_TRANS_label']

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad E\ \texttt{expands}\ E' \Longrightarrow\\ \quad \forall l\ E_2.\\ \qquad E'\ {-}\texttt{label}\ l{\rightarrow}\ E_2 \Longrightarrow \exists E_1.\ E\ {=}\texttt{label}\ l{\Rightarrow}\ E_1 \land E_1\ \texttt{expands}\ E_2\end{array}$$

[expands_TRANS_tau]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad E\ \texttt{expands}\ E' \Longrightarrow\\ \quad \forall E_1.\\ \qquad E\ {-\tau\rightarrow}\ E_1 \Longrightarrow E_1\ \texttt{expands}\ E' \lor \exists E_2.\ E'\ {-\tau\rightarrow}\ E_2 \land E_1\ \texttt{expands}\ E_2\end{array}$$

[expands_TRANS_tau']

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad E\ \texttt{expands}\ E' \Longrightarrow\\ \quad \forall E_2.\ E'\ {-\tau\rightarrow}\ E_2 \Longrightarrow \exists E_1.\ E\ {=\tau\Rightarrow}\ E_1 \land E_1\ \texttt{expands}\ E_2\end{array}$$

[expands_transitive]

$$\vdash \texttt{transitive (expands)}$$

[expands_WEAK_TRANS']

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{expands}\ E' \implies\\
\quad\quad \forall u\ E_2.\ E'\ =u\Rightarrow\ E_2 \implies \exists E_1.\ E\ =u\Rightarrow\ E_1 \land E_1\ \texttt{expands}\ E_2
\end{array}$$

[expands_WEAK_TRANS_label]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{expands}\ E' \implies\\
\quad\quad \forall l\ E_1.\\
\quad\quad\quad E\ =\texttt{label}\ l\Rightarrow\ E_1 \implies \exists E_2.\ E'\ =\texttt{label}\ l\Rightarrow\ E_2 \land E_1\ \texttt{expands}\ E_2
\end{array}$$

[expands_WEAK_TRANS_tau]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{expands}\ E' \implies\\
\quad\quad \forall E_1.\ E\ =\tau\Rightarrow\ E_1 \implies \exists E_2.\ \texttt{EPS}\ E'\ E_2 \land E_1\ \texttt{expands}\ E_2
\end{array}$$

[EXPANSION_ALT]

$$\begin{array}{l}
\vdash \texttt{EXPANSION}\ \mathit{Exp} \iff\\
\quad \forall E\ E'.\\
\quad\quad \mathit{Exp}\ E\ E' \implies\\
\quad\quad (\forall l\ E_1.\\
\quad\quad\quad E\ -\texttt{label}\ l\rightarrow\ E_1 \implies \exists E_2.\ E'\ -\texttt{label}\ l\rightarrow\ E_2 \land \mathit{Exp}\ E_1\ E_2) \land\\
\quad\quad (\forall E_1.\ E\ -\tau\rightarrow\ E_1 \implies \mathit{Exp}\ E_1\ E' \lor \exists E_2.\ E'\ -\tau\rightarrow\ E_2 \land \mathit{Exp}\ E_1\ E_2) \land\\
\quad\quad \forall u\ E_2.\ E'\ -u\rightarrow\ E_2 \implies \exists E_1.\ E\ =u\Rightarrow\ E_1 \land \mathit{Exp}\ E_1\ E_2
\end{array}$$

[EXPANSION_EPS]

$$\begin{array}{l}
\vdash \forall \mathit{Exp}.\\
\quad\quad \texttt{EXPANSION}\ \mathit{Exp} \implies\\
\quad\quad \forall E\ E'.\\
\quad\quad\quad \mathit{Exp}\ E\ E' \implies \forall E_1.\ \texttt{EPS}\ E\ E_1 \implies \exists E_2.\ \texttt{EPS}\ E'\ E_2 \land \mathit{Exp}\ E_1\ E_2
\end{array}$$

[EXPANSION_EPS']

$$\begin{array}{l}
\vdash \forall \mathit{Exp}.\\
\quad\quad \texttt{EXPANSION}\ \mathit{Exp} \implies\\
\quad\quad \forall E\ E'.\\
\quad\quad\quad \mathit{Exp}\ E\ E' \implies \forall E_2.\ \texttt{EPS}\ E'\ E_2 \implies \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \mathit{Exp}\ E_1\ E_2
\end{array}$$

[EXPANSION_IMP_WEAK_BISIM]

$$\vdash \forall \mathit{Exp}.\ \texttt{EXPANSION}\ \mathit{Exp} \implies \texttt{WEAK\_BISIM}\ \mathit{Exp}$$

[EXPANSION_SUBSET_expands]

$$\vdash \forall \mathit{Exp}.\ \texttt{EXPANSION}\ \mathit{Exp} \implies \mathit{Exp}\ \texttt{RSUBSET (expands)}$$

[EXPANSION_WEAK_TRANS']

$$\begin{array}{l} \vdash \forall \mathit{Exp}.\\ \quad \texttt{EXPANSION}\ \mathit{Exp} \Longrightarrow\\ \quad \forall E\ E'.\\ \qquad \mathit{Exp}\ E\ E' \Longrightarrow \forall u\ E_2.\ E' =u\Rightarrow E_2 \Longrightarrow \exists E_1.\ E =u\Rightarrow E_1 \land \mathit{Exp}\ E_1\ E_2 \end{array}$$

[IDENTITY_EXPANSION]

$\vdash$ `EXPANSION (=)`

[STRONG_BISIM_IMP_EXPANSION]

$\vdash \forall \mathit{Exp}.\ \texttt{STRONG\_BISIM}\ \mathit{Exp} \Longrightarrow \texttt{EXPANSION}\ \mathit{Exp}$

[STRONG_EQUIV_IMP_expands]

$\vdash \forall P\ Q.\ \texttt{STRONG\_EQUIV}\ P\ Q \Longrightarrow P\ \texttt{expands}\ Q$

9.13 Contraction Theory

Built: 08 Dicembre 2017
Parent Theories: Expansion

9.13.1 Definitions

[C_contracts]

$\vdash$ `C_contracts = CC (contracts)`

[CONTRACTION]

$$\begin{array}{l} \vdash \forall \mathit{Con}.\\ \quad \texttt{CONTRACTION}\ \mathit{Con} \iff\\ \quad \forall E\ E'.\\ \qquad \mathit{Con}\ E\ E' \Longrightarrow\\ \qquad (\forall l.\\ \qquad\quad (\forall E_1.\\ \qquad\qquad E\ -\texttt{label}\ l\rightarrow\ E_1 \Longrightarrow\\ \qquad\qquad \exists E_2.\ E'\ -\texttt{label}\ l\rightarrow\ E_2 \land \mathit{Con}\ E_1\ E_2) \land\\ \qquad\quad \forall E_2.\\ \qquad\qquad E'\ -\texttt{label}\ l\rightarrow\ E_2 \Longrightarrow\\ \qquad\qquad \exists E_1.\ E\ =\texttt{label}\ l\Rightarrow\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2) \land\\ \qquad (\forall E_1.\\ \qquad\quad E\ -\tau\rightarrow\ E_1 \Longrightarrow \mathit{Con}\ E_1\ E' \lor \exists E_2.\ E'\ -\tau\rightarrow\ E_2 \land \mathit{Con}\ E_1\ E_2) \land\\ \qquad \forall E_2.\ E'\ -\tau\rightarrow\ E_2 \Longrightarrow \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2 \end{array}$$

[contracts_def]

$$\begin{array}{l}
\vdash \texttt{(contracts) =} \\
\quad (\lambda\, a_0\ \ a_1\,. \\
\qquad \exists\, \mathit{contracts}'\,. \\
\qquad\quad \mathit{contracts}'\ \ a_0\ \ a_1\ \wedge \\
\qquad\quad \forall\, a_0\ \ a_1\,. \\
\qquad\qquad \mathit{contracts}'\ \ a_0\ \ a_1 \implies \\
\qquad\qquad (\forall\, l\,. \\
\qquad\qquad\quad (\forall\, E_1\,. \\
\qquad\qquad\qquad a_0\ \ -\texttt{label}\ \ l\!\rightarrow\ E_1 \implies \\
\qquad\qquad\qquad \exists\, E_2\,.\ \ a_1\ \ -\texttt{label}\ \ l\!\rightarrow\ E_2\ \wedge\ \mathit{contracts}'\ \ E_1\ \ E_2)\ \wedge \\
\qquad\qquad\quad \forall\, E_2\,. \\
\qquad\qquad\qquad a_1\ \ -\texttt{label}\ \ l\!\rightarrow\ E_2 \implies \\
\qquad\qquad\qquad \exists\, E_1\,.\ \ a_0\ \ =\texttt{label}\ \ l\!\Rightarrow\ E_1\ \wedge\ \texttt{WEAK\_EQUIV}\ \ E_1\ \ E_2)\ \wedge \\
\qquad\qquad (\forall\, E_1\,. \\
\qquad\qquad\quad a_0\ \ -\tau\!\rightarrow\ E_1 \implies \\
\qquad\qquad\quad \mathit{contracts}'\ \ E_1\ \ a_1\ \vee \\
\qquad\qquad\quad \exists\, E_2\,.\ \ a_1\ \ -\tau\!\rightarrow\ E_2\ \wedge\ \mathit{contracts}'\ \ E_1\ \ E_2)\ \wedge \\
\qquad\qquad \forall\, E_2\,.\ \ a_1\ \ -\tau\!\rightarrow\ E_2 \implies \exists\, E_1\,.\ \ \texttt{EPS}\ \ a_0\ \ E_1\ \wedge\ \texttt{WEAK\_EQUIV}\ \ E_1\ \ E_2)
\end{array}$$

[OBS_contracts]

$$\begin{array}{l}
\vdash \forall\, E\ \ E'\,. \\
\qquad \texttt{OBS\_contracts}\ \ E\ \ E' \iff \\
\qquad \forall\, u\,. \\
\qquad\quad (\forall\, E_1\,.\ \ E\ \ -u\!\rightarrow\ E_1 \implies \exists\, E_2\,.\ \ E'\ \ -u\!\rightarrow\ E_2\ \wedge\ E_1\ \ \texttt{contracts}\ \ E_2)\ \wedge \\
\qquad\quad \forall\, E_2\,.\ \ E'\ \ -u\!\rightarrow\ E_2 \implies \exists\, E_1\,.\ \ E\ \ =u\!\Rightarrow\ E_1\ \wedge\ \texttt{WEAK\_EQUIV}\ \ E_1\ \ E_2
\end{array}$$

[SUM_contracts]

$$\vdash \texttt{SUM\_contracts = }(\lambda\, p\ \ q\,.\ \ \forall\, r\,.\ \ p\ \texttt{+}\ r\ \ \texttt{contracts}\ \ q\ \texttt{+}\ r)$$

9.13.2 Theorems

[C_contracts_IMP_SUM_contracts]

$$\vdash \forall\, p\ \ q\,.\ \ \texttt{C\_contracts}\ \ p\ \ q \implies \texttt{SUM\_contracts}\ \ p\ \ q$$

[C_contracts_precongruence]

$$\vdash \texttt{precongruence C\_contracts}$$

[C_contracts_thm]

$$\vdash \texttt{C\_contracts = }(\lambda\, g\ \ h\,.\ \ \forall\, c\,.\ \ \texttt{CONTEXT}\ \ c \implies c\ \ g\ \ \texttt{contracts}\ \ c\ \ h)$$

[COARSEST_PRECONGR_THM]

$$\begin{array}{l}
\vdash \forall\, p\ \ q\,. \\
\qquad \texttt{free\_action}\ \ p\ \wedge\ \texttt{free\_action}\ \ q \implies \\
\qquad (\texttt{OBS\_contracts}\ \ p\ \ q \iff \texttt{SUM\_contracts}\ \ p\ \ q)
\end{array}$$

[COARSEST_PRECONGR_THM’]

$$\begin{aligned}
&\vdash \forall p\ q.\\
&\qquad \texttt{free\_action}\ p \land \texttt{free\_action}\ q \implies\\
&\qquad (\texttt{OBS\_contracts}\ p\ q \iff \forall r.\ p\ \texttt{+}\ r\ \texttt{contracts}\ q\ \texttt{+}\ r)
\end{aligned}$$

[COMP_CONTRACTION]

$$\begin{aligned}
&\vdash \forall \mathit{Con}_1\ \mathit{Con}_2.\\
&\qquad \texttt{CONTRACTION}\ \mathit{Con}_1 \land \texttt{CONTRACTION}\ \mathit{Con}_2 \implies\\
&\qquad \texttt{CONTRACTION}\ (\mathit{Con}_2\ \texttt{O}\ \mathit{Con}_1)
\end{aligned}$$

[CONTRACTION_EPS]

$$\begin{aligned}
&\vdash \forall \mathit{Con}.\\
&\qquad \texttt{CONTRACTION}\ \mathit{Con} \implies\\
&\qquad \forall E\ E'.\\
&\qquad\quad \mathit{Con}\ E\ E' \implies \forall E_1.\ \texttt{EPS}\ E\ E_1 \implies \exists E_2.\ \texttt{EPS}\ E'\ E_2 \land \mathit{Con}\ E_1\ E_2
\end{aligned}$$

[CONTRACTION_EPS’]

$$\begin{aligned}
&\vdash \forall \mathit{Con}.\\
&\qquad \texttt{CONTRACTION}\ \mathit{Con} \implies\\
&\qquad \forall E\ E'.\\
&\qquad\quad \mathit{Con}\ E\ E' \implies\\
&\qquad\quad \forall u\ E_2.\ \texttt{EPS}\ E'\ E_2 \implies \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{aligned}$$

[CONTRACTION_SUBSET_contracts]

$$\vdash \forall \mathit{Con}.\ \texttt{CONTRACTION}\ \mathit{Con} \implies \mathit{Con}\ \texttt{RSUBSET}\ (\texttt{contracts})$$

[CONTRACTION_WEAK_TRANS_label’]

$$\begin{aligned}
&\vdash \forall \mathit{Con}.\\
&\qquad \texttt{CONTRACTION}\ \mathit{Con} \implies\\
&\qquad \forall E\ E'.\\
&\qquad\quad \mathit{Con}\ E\ E' \implies\\
&\qquad\quad \forall l\ E_2.\\
&\qquad\qquad E'\ \texttt{=label}\ l\Rightarrow\ E_2 \implies\\
&\qquad\qquad \exists E_1.\ E\ \texttt{=label}\ l\Rightarrow\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{aligned}$$

[contracts_AND_TRACE1]

$$\begin{aligned}
&\vdash \forall E\ E'.\\
&\qquad E\ \texttt{contracts}\ E' \implies\\
&\qquad \forall \mathit{xs}\ E_1.\\
&\qquad\quad \texttt{TRACE}\ E\ \mathit{xs}\ E_1 \implies\\
&\qquad\quad \exists \mathit{xs}'\ E_2.\ \texttt{TRACE}\ E'\ \mathit{xs}'\ E_2 \land E_1\ \texttt{contracts}\ E_2
\end{aligned}$$

[contracts_AND_TRACE2]

$$
\begin{array}{l}
\vdash \forall E\ E'.\\
\qquad E\ \texttt{contracts}\ E' \Longrightarrow\\
\qquad \forall xs\ E_1.\\
\qquad\quad \texttt{TRACE}\ E\ xs\ E_1 \Longrightarrow\\
\qquad\quad \exists xs'\ E_2.\\
\qquad\qquad \texttt{TRACE}\ E'\ xs'\ E_2 \land E_1\ \texttt{contracts}\ E_2 \land\\
\qquad\qquad \texttt{LENGTH}\ xs' \leq \texttt{LENGTH}\ xs
\end{array}
$$

[contracts_AND_TRACE_label]

$$
\begin{array}{l}
\vdash \forall E\ E'.\\
\qquad E\ \texttt{contracts}\ E' \Longrightarrow\\
\qquad \forall xs\ l\ E_1.\\
\qquad\quad \texttt{TRACE}\ E\ xs\ E_1 \land \texttt{UNIQUE\_LABEL (label}\ l\texttt{)}\ xs \Longrightarrow\\
\qquad\quad \exists xs'\ E_2.\\
\qquad\qquad \texttt{TRACE}\ E'\ xs'\ E_2 \land E_1\ \texttt{contracts}\ E_2 \land\\
\qquad\qquad \texttt{LENGTH}\ xs' \leq \texttt{LENGTH}\ xs \land \texttt{UNIQUE\_LABEL (label}\ l\texttt{)}\ xs'
\end{array}
$$

[contracts_AND_TRACE_tau]

$$
\begin{array}{l}
\vdash \forall E\ E'.\\
\qquad E\ \texttt{contracts}\ E' \Longrightarrow\\
\qquad \forall xs\ E_1.\\
\qquad\quad \texttt{TRACE}\ E\ xs\ E_1 \land \texttt{NO\_LABEL}\ xs \Longrightarrow\\
\qquad\quad \exists xs'\ E_2.\\
\qquad\qquad \texttt{TRACE}\ E'\ xs'\ E_2 \land E_1\ \texttt{contracts}\ E_2 \land\\
\qquad\qquad \texttt{LENGTH}\ xs' \leq \texttt{LENGTH}\ xs \land \texttt{NO\_LABEL}\ xs'
\end{array}
$$

[contracts_cases]

$$
\begin{array}{l}
\vdash \forall a_0\ a_1.\\
\qquad a_0\ \texttt{contracts}\ a_1 \iff\\
\qquad (\forall l.\\
\qquad\qquad (\forall E_1.\\
\qquad\qquad\qquad a_0\ {-}\texttt{label}\ l{\rightarrow}\ E_1 \Longrightarrow\\
\qquad\qquad\qquad \exists E_2.\ a_1\ {-}\texttt{label}\ l{\rightarrow}\ E_2 \land E_1\ \texttt{contracts}\ E_2) \land\\
\qquad\qquad \forall E_2.\\
\qquad\qquad\quad a_1\ {-}\texttt{label}\ l{\rightarrow}\ E_2 \Longrightarrow\\
\qquad\qquad\quad \exists E_1.\ a_0\ {=}\texttt{label}\ l{\Rightarrow}\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2) \land\\
\qquad (\forall E_1.\\
\qquad\qquad a_0\ {-}\tau{\rightarrow}\ E_1 \Longrightarrow\\
\qquad\qquad E_1\ \texttt{contracts}\ a_1 \lor \exists E_2.\ a_1\ {-}\tau{\rightarrow}\ E_2 \land E_1\ \texttt{contracts}\ E_2) \land\\
\qquad \forall E_2.\ a_1\ {-}\tau{\rightarrow}\ E_2 \Longrightarrow \exists E_1.\ \texttt{EPS}\ a_0\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}
$$

[contracts_coind]

$$
\begin{array}{l}
\vdash \forall \mathit{contracts}'.\\
\qquad (\forall a_0\ a_1.\\
\qquad\qquad \mathit{contracts}'\ a_0\ a_1 \Longrightarrow
\end{array}
$$

```
    ($\forall l$.
       ($\forall E_1$.
          $a_0$ −label $l$→ $E_1$ $\Longrightarrow$
          $\exists E_2$. $a_1$ −label $l$→ $E_2$ $\land$ $contracts'$ $E_1$ $E_2$) $\land$
       $\forall E_2$.
         $a_1$ −label $l$→ $E_2$ $\Longrightarrow$
         $\exists E_1$. $a_0$ =label $l$⇒ $E_1$ $\land$ WEAK_EQUIV $E_1$ $E_2$) $\land$
    ($\forall E_1$.
       $a_0$ −$\tau$→ $E_1$ $\Longrightarrow$
       $contracts'$ $E_1$ $a_1$ $\lor$
       $\exists E_2$. $a_1$ −$\tau$→ $E_2$ $\land$ $contracts'$ $E_1$ $E_2$) $\land$
    $\forall E_2$. $a_1$ −$\tau$→ $E_2$ $\Longrightarrow$ $\exists E_1$. EPS $a_0$ $E_1$ $\land$ WEAK_EQUIV $E_1$ $E_2$) $\Longrightarrow$
  $\forall a_0$ $a_1$. $contracts'$ $a_0$ $a_1$ $\Longrightarrow$ $a_0$ contracts $a_1$
```

[contracts_EPS]

```
⊢ $\forall E$ $E'$.
    $E$ contracts $E'$ $\Longrightarrow$
    $\forall E_1$. EPS $E$ $E_1$ $\Longrightarrow$ $\exists E_2$. EPS $E'$ $E_2$ $\land$ $E_1$ contracts $E_2$
```

[contracts_EPS']

```
⊢ $\forall E$ $E'$.
    $E$ contracts $E'$ $\Longrightarrow$
    $\forall E_2$. EPS $E'$ $E_2$ $\Longrightarrow$ $\exists E_1$. EPS $E$ $E_1$ $\land$ WEAK_EQUIV $E_1$ $E_2$
```

[contracts_IMP_WEAK_EQUIV]

```
⊢ $\forall P$ $Q$. $P$ contracts $Q$ $\Longrightarrow$ WEAK_EQUIV $P$ $Q$
```

[contracts_is_CONTRACTION]

```
⊢ CONTRACTION (contracts)
```

[contracts_precongruence]

```
⊢ precongruence1 (contracts)
```

[contracts_PreOrder]

```
⊢ PreOrder (contracts)
```

[contracts_PRESD_BY_GUARDED_SUM]

```
⊢ $\forall E_1$ $E_1'$ $E_2$ $E_2'$ $a_1$ $a_2$.
    $E_1$ contracts $E_1'$ $\land$ $E_2$ contracts $E_2'$ $\Longrightarrow$
    $a_1$..$E_1$ + $a_2$..$E_2$ contracts $a_1$..$E_1'$ + $a_2$..$E_2'$
```

[contracts_PRESD_BY_PAR]

```
⊢ $\forall E_1$ $E_1'$ $E_2$ $E_2'$.
    $E_1$ contracts $E_1'$ $\land$ $E_2$ contracts $E_2'$ $\Longrightarrow$
    $E_1$ $\parallel$ $E_2$ contracts $E_1'$ $\parallel$ $E_2'$
```

[contracts_PROP6]

$\vdash \forall E\ E'.\ E\ \texttt{contracts}\ E' \Longrightarrow \forall u.\ \texttt{OBS\_contracts}\ (u\texttt{..}E)\ (u\texttt{..}E')$

[contracts_REFL]

$\vdash \forall x.\ x\ \texttt{contracts}\ x$

[contracts_reflexive]

$\vdash \texttt{reflexive (contracts)}$

[contracts_rules]

$$\begin{array}{l}
\vdash \forall E\ E'. \\
\quad (\forall l. \\
\qquad (\forall E_1. \\
\qquad\quad E\ -\texttt{label}\ l\!\rightarrow\ E_1 \Longrightarrow \\
\qquad\quad \exists E_2.\ E'\ -\texttt{label}\ l\!\rightarrow\ E_2 \wedge E_1\ \texttt{contracts}\ E_2) \wedge \\
\qquad \forall E_2. \\
\qquad\quad E'\ -\texttt{label}\ l\!\rightarrow\ E_2 \Longrightarrow \\
\qquad\quad \exists E_1.\ E\ =\texttt{label}\ l\!\Rightarrow\ E_1 \wedge \texttt{WEAK\_EQUIV}\ E_1\ E_2) \wedge \\
\quad (\forall E_1. \\
\qquad E\ -\tau\!\rightarrow\ E_1 \Longrightarrow \\
\qquad E_1\ \texttt{contracts}\ E' \vee \exists E_2.\ E'\ -\tau\!\rightarrow\ E_2 \wedge E_1\ \texttt{contracts}\ E_2) \wedge \\
\quad (\forall E_2.\ E'\ -\tau\!\rightarrow\ E_2 \Longrightarrow \exists E_1.\ \texttt{EPS}\ E\ E_1 \wedge \texttt{WEAK\_EQUIV}\ E_1\ E_2) \Longrightarrow \\
\quad E\ \texttt{contracts}\ E'
\end{array}$$

[contracts_SUBST_GCONTEXT]

$\vdash \forall P\ Q.\ P\ \texttt{contracts}\ Q \Longrightarrow \forall E.\ \texttt{GCONTEXT}\ E \Longrightarrow E\ P\ \texttt{contracts}\ E\ Q$

[contracts_SUBST_PAR_L]

$\vdash \forall E\ E'.\ E\ \texttt{contracts}\ E' \Longrightarrow \forall E''.\ E'' \parallel E\ \texttt{contracts}\ E'' \parallel E'$

[contracts_SUBST_PAR_R]

$\vdash \forall E\ E'.\ E\ \texttt{contracts}\ E' \Longrightarrow \forall E''.\ E \parallel E''\ \texttt{contracts}\ E' \parallel E''$

[contracts_SUBST_PREFIX]

$\vdash \forall E\ E'.\ E\ \texttt{contracts}\ E' \Longrightarrow \forall u.\ u\texttt{..}E\ \texttt{contracts}\ u\texttt{..}E'$

[contracts_SUBST_RELAB]

$\vdash \forall E\ E'.\ E\ \texttt{contracts}\ E' \Longrightarrow \forall \mathit{rf}.\ \texttt{relab}\ E\ \mathit{rf}\ \texttt{contracts}\ \texttt{relab}\ E'\ \mathit{rf}$

[contracts_SUBST_RESTR]

$\vdash \forall E\ E'.\ E\ \texttt{contracts}\ E' \Longrightarrow \forall L.\ \nu\ L\ E\ \texttt{contracts}\ \nu\ L\ E'$

[contracts_thm]

$\vdash \forall P\ Q.\ P\ \texttt{contracts}\ Q \iff \exists \mathit{Con}.\ \mathit{Con}\ P\ Q \wedge \texttt{CONTRACTION}\ \mathit{Con}$

[contracts_TRANS]

$$\vdash \forall x\ y\ z.\ x\ \texttt{contracts}\ y \land y\ \texttt{contracts}\ z \implies x\ \texttt{contracts}\ z$$

[contracts_TRANS_label]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{contracts}\ E' \implies\\
\quad\quad \forall l\ E_1.\\
\quad\quad\quad E\ -\texttt{label}\ l\rightarrow\ E_1 \implies \exists E_2.\ E'\ -\texttt{label}\ l\rightarrow\ E_2 \land E_1\ \texttt{contracts}\ E_2
\end{array}$$

[contracts_TRANS_label']

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{contracts}\ E' \implies\\
\quad\quad \forall l\ E_2.\\
\quad\quad\quad E'\ -\texttt{label}\ l\rightarrow\ E_2 \implies \exists E_1.\ E\ =\texttt{label}\ l\Rightarrow\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[contracts_TRANS_tau]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{contracts}\ E' \implies\\
\quad\quad \forall E_1.\\
\quad\quad\quad E\ -\tau\rightarrow\ E_1 \implies\\
\quad\quad\quad E_1\ \texttt{contracts}\ E' \lor \exists E_2.\ E'\ -\tau\rightarrow\ E_2 \land E_1\ \texttt{contracts}\ E_2
\end{array}$$

[contracts_TRANS_tau']

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{contracts}\ E' \implies\\
\quad\quad \forall E_2.\ E'\ -\tau\rightarrow\ E_2 \implies \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[contracts_transitive]

$$\vdash \texttt{transitive (contracts)}$$

[contracts_WEAK_TRANS_label]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{contracts}\ E' \implies\\
\quad\quad \forall l\ E_1.\\
\quad\quad\quad E\ =\texttt{label}\ l\Rightarrow\ E_1 \implies \exists E_2.\ E'\ =\texttt{label}\ l\Rightarrow\ E_2 \land E_1\ \texttt{contracts}\ E_2
\end{array}$$

[contracts_WEAK_TRANS_label']

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{contracts}\ E' \implies\\
\quad\quad \forall l\ E_2.\\
\quad\quad\quad E'\ =\texttt{label}\ l\Rightarrow\ E_2 \implies \exists E_1.\ E\ =\texttt{label}\ l\Rightarrow\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[contracts_WEAK_TRANS_tau]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad E\ \texttt{contracts}\ E' \implies\\
\quad\quad \forall E_1.\ E\ =\tau\Rightarrow\ E_1 \implies \exists E_2.\ \texttt{EPS}\ E'\ E_2 \land E_1\ \texttt{contracts}\ E_2
\end{array}$$

[contracts_WEAK_TRANS_tau’]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad E\ \texttt{contracts}\ E' \Longrightarrow\\
\quad \forall l\ E_2.\ E'\ {=}\tau{\Rightarrow}\ E_2 \Longrightarrow \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[expands_IMP_contracts]

$$\vdash \forall P\ Q.\ P\ \texttt{expands}\ Q \Longrightarrow P\ \texttt{contracts}\ Q$$

[EXPANSION_IMP_CONTRACTION]

$$\vdash \forall Con.\ \texttt{EXPANSION}\ Con \Longrightarrow \texttt{CONTRACTION}\ Con$$

[IDENTITY_CONTRACTION]

$$\vdash \texttt{CONTRACTION (=)}$$

[OBS_contracts_AND_TRACE_label]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\
\quad \forall xs\ l\ E_1.\\
\quad\quad \texttt{TRACE}\ E\ xs\ E_1 \land \texttt{UNIQUE\_LABEL (label}\ l\texttt{)}\ xs \Longrightarrow\\
\quad\quad \exists xs'\ E_2.\\
\quad\quad\quad \texttt{TRACE}\ E'\ xs'\ E_2 \land E_1\ \texttt{contracts}\ E_2 \land\\
\quad\quad\quad \texttt{LENGTH}\ xs' \leq \texttt{LENGTH}\ xs \land \texttt{UNIQUE\_LABEL (label}\ l\texttt{)}\ xs'
\end{array}$$

[OBS_contracts_AND_TRACE_tau]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\
\quad \forall xs\ l\ E_1.\\
\quad\quad \texttt{TRACE}\ E\ xs\ E_1 \land \texttt{NO\_LABEL}\ xs \Longrightarrow\\
\quad\quad \exists xs'\ E_2.\\
\quad\quad\quad \texttt{TRACE}\ E'\ xs'\ E_2 \land E_1\ \texttt{contracts}\ E_2 \land\\
\quad\quad\quad \texttt{LENGTH}\ xs' \leq \texttt{LENGTH}\ xs \land \texttt{NO\_LABEL}\ xs'
\end{array}$$

[OBS_contracts_BY_CONTRACTION]

$$\begin{array}{l}
\vdash \forall Con.\\
\quad \texttt{CONTRACTION}\ Con \Longrightarrow\\
\quad \forall E\ E'.\\
\quad\quad (\forall u.\\
\quad\quad\quad (\forall E_1.\ E\ {-}u{\rightarrow}\ E_1 \Longrightarrow \exists E_2.\ E'\ {-}u{\rightarrow}\ E_2 \land Con\ E_1\ E_2) \land\\
\quad\quad\quad \forall E_2.\ E'\ {-}u{\rightarrow}\ E_2 \Longrightarrow \exists E_1.\ E\ {=}u{\Rightarrow}\ E_1 \land Con\ E_1\ E_2) \Longrightarrow\\
\quad\quad \texttt{OBS\_contracts}\ E\ E'
\end{array}$$

[OBS_contracts_EPS’]

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\
\quad \forall E_2.\ \texttt{EPS}\ E'\ E_2 \Longrightarrow \exists E_1.\ \texttt{EPS}\ E\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[OBS_contracts_IMP_C_contracts]

$\vdash \forall p\ q.\ \texttt{OBS\_contracts}\ p\ q \implies \texttt{C\_contracts}\ p\ q$

[OBS_contracts_IMP_contracts]

$\vdash \forall E\ E'.\ \texttt{OBS\_contracts}\ E\ E' \implies E\ \texttt{contracts}\ E'$

[OBS_contracts_IMP_OBS_CONGR]

$\vdash \forall E\ E'.\ \texttt{OBS\_contracts}\ E\ E' \implies \texttt{OBS\_CONGR}\ E\ E'$

[OBS_contracts_IMP_SUM_contracts]

$\vdash \forall p\ q.\ \texttt{OBS\_contracts}\ p\ q \implies \texttt{SUM\_contracts}\ p\ q$

[OBS_contracts_IMP_WEAK_EQUIV]

$\vdash \forall E\ E'.\ \texttt{OBS\_contracts}\ E\ E' \implies \texttt{WEAK\_EQUIV}\ E\ E'$

[OBS_contracts_IMP_WEAK_EQUIV']

$\vdash \forall E\ E'.\ \texttt{OBS\_contracts}\ E\ E' \implies \texttt{WEAK\_EQUIV}\ E\ E'$

[OBS_contracts_precongruence]

$\vdash \texttt{precongruence OBS\_contracts}$

[OBS_contracts_PreOrder]

$\vdash \texttt{PreOrder OBS\_contracts}$

[OBS_contracts_PRESD_BY_PAR]

$$\begin{array}{l}
\vdash \forall E_1\ E_1'\ E_2\ E_2'.\\
\quad \texttt{OBS\_contracts}\ E_1\ E_1' \land \texttt{OBS\_contracts}\ E_2\ E_2' \implies\\
\quad \texttt{OBS\_contracts}\ (E_1 \parallel E_2)\ (E_1' \parallel E_2')
\end{array}$$

[OBS_contracts_PRESD_BY_SUM]

$$\begin{array}{l}
\vdash \forall E_1\ E_1'\ E_2\ E_2'.\\
\quad \texttt{OBS\_contracts}\ E_1\ E_1' \land \texttt{OBS\_contracts}\ E_2\ E_2' \implies\\
\quad \texttt{OBS\_contracts}\ (E_1 + E_2)\ (E_1' + E_2')
\end{array}$$

[OBS_contracts_REFL]

$\vdash \forall E.\ \texttt{OBS\_contracts}\ E\ E$

[OBS_contracts_SUBST_CONTEXT]

$$\begin{array}{l}
\vdash \forall P\ Q.\\
\quad \texttt{OBS\_contracts}\ P\ Q \implies\\
\quad \forall E.\ \texttt{CONTEXT}\ E \implies \texttt{OBS\_contracts}\ (E\ P)\ (E\ Q)
\end{array}$$

[OBS_contracts_SUBST_PAR_L]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\ \quad \forall E''.\ \texttt{OBS\_contracts}\ (E'' \parallel E)\ (E'' \parallel E')\end{array}$$

[OBS_contracts_SUBST_PAR_R]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\ \quad \forall E''.\ \texttt{OBS\_contracts}\ (E \parallel E'')\ (E' \parallel E'')\end{array}$$

[OBS_contracts_SUBST_PREFIX]

$$\vdash \forall E\ E'.\ \texttt{OBS\_contracts}\ E\ E' \Longrightarrow \forall u.\ \texttt{OBS\_contracts}\ (u..E)\ (u..E')$$

[OBS_contracts_SUBST_RELAB]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\ \quad \forall rf.\ \texttt{OBS\_contracts}\ (\texttt{relab}\ E\ rf)\ (\texttt{relab}\ E'\ rf)\end{array}$$

[OBS_contracts_SUBST_RESTR]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow \forall L.\ \texttt{OBS\_contracts}\ (\nu\ L\ E)\ (\nu\ L\ E')\end{array}$$

[OBS_contracts_SUBST_SUM_L]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\ \quad \forall E''.\ \texttt{OBS\_contracts}\ (E''\ \texttt{+}\ E)\ (E''\ \texttt{+}\ E')\end{array}$$

[OBS_contracts_SUBST_SUM_R]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\ \quad \forall E''.\ \texttt{OBS\_contracts}\ (E\ \texttt{+}\ E'')\ (E'\ \texttt{+}\ E'')\end{array}$$

[OBS_contracts_TRANS]

$$\begin{array}{l}\vdash \forall E\ E'\ E''.\\ \quad \texttt{OBS\_contracts}\ E\ E' \land \texttt{OBS\_contracts}\ E'\ E'' \Longrightarrow\\ \quad \texttt{OBS\_contracts}\ E\ E''\end{array}$$

[OBS_contracts_TRANS_LEFT]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\ \quad \forall u\ E_1.\ E\ -u\rightarrow\ E_1 \Longrightarrow \exists E_2.\ E'\ -u\rightarrow\ E_2 \land E_1\ \texttt{contracts}\ E_2\end{array}$$

[OBS_contracts_TRANS_RIGHT]

$$\begin{array}{l}\vdash \forall E\ E'.\\ \quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\ \quad \forall u\ E_2.\ E'\ -u\rightarrow\ E_2 \Longrightarrow \exists E_1.\ E\ =u\Rightarrow\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2\end{array}$$

[OBS_contracts_WEAK_TRANS']

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\
\quad\quad \forall u\ E_2.\ E'\ =u\Rightarrow\ E_2 \Longrightarrow \exists E_1.\ E\ =u\Rightarrow\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[OBS_contracts_WEAK_TRANS_label']

$$\begin{array}{l}
\vdash \forall E\ E'.\\
\quad\quad \texttt{OBS\_contracts}\ E\ E' \Longrightarrow\\
\quad\quad \forall l\ E_2.\\
\quad\quad\quad E'\ =\texttt{label}\ l\Rightarrow\ E_2 \Longrightarrow \exists E_1.\ E\ =\texttt{label}\ l\Rightarrow\ E_1 \land \texttt{WEAK\_EQUIV}\ E_1\ E_2
\end{array}$$

[SUM_contracts_IMP_OBS_contracts]

$$\begin{array}{l}
\vdash \forall p\ q.\\
\quad\quad \texttt{free\_action}\ p \land \texttt{free\_action}\ q \Longrightarrow\\
\quad\quad \texttt{SUM\_contracts}\ p\ q \Longrightarrow\\
\quad\quad \texttt{OBS\_contracts}\ p\ q
\end{array}$$

9.14 UniqueSolutions Theory

Built: 08 Dicembre 2017

Parent Theories: Contraction, BisimulationUpto

9.14.1 Theorems

[GSEQ_EPS_lemma]

$$\begin{array}{l}
\vdash \forall E\ P\ Q\ R\ H.\\
\quad\quad \texttt{SG}\ E \land \texttt{GSEQ}\ E \land \texttt{WEAK\_EQUIV}\ P\ (E\ P) \land \texttt{WEAK\_EQUIV}\ Q\ (E\ Q) \land\\
\quad\quad \texttt{GSEQ}\ H \land (R = (\lambda x\ y.\ \exists H.\ \texttt{GSEQ}\ H \land (x = H\ P) \land (y = H\ Q))) \Longrightarrow\\
\quad\quad (\forall P'.\\
\quad\quad\quad \texttt{EPS}\ (H\ P)\ P' \Longrightarrow\\
\quad\quad\quad \exists Q'.\\
\quad\quad\quad\quad \texttt{EPS}\ (H\ Q)\ Q' \land (\texttt{WEAK\_EQUIV}\ \texttt{O}\ R\ \texttt{O}\ \texttt{WEAK\_EQUIV})\ P'\ Q') \land\\
\quad\quad \forall Q'.\\
\quad\quad\quad \texttt{EPS}\ (H\ Q)\ Q' \Longrightarrow\\
\quad\quad\quad \exists P'.\ \texttt{EPS}\ (H\ P)\ P' \land (\texttt{WEAK\_EQUIV}\ \texttt{O}\ R\ \texttt{O}\ \texttt{WEAK\_EQUIV})\ P'\ Q'
\end{array}$$

[OBS_unfolding_lemma1]

$$\begin{array}{l}
\vdash \forall E\ C\ P.\\
\quad\quad \texttt{CONTEXT}\ E \land \texttt{CONTEXT}\ C \land \texttt{OBS\_contracts}\ P\ (E\ P) \Longrightarrow\\
\quad\quad \forall n.\ \texttt{OBS\_contracts}\ (C\ P)\ ((C \circ \texttt{FUNPOW}\ E\ n)\ P)
\end{array}$$

[OBS_unfolding_lemma2]

$$\begin{array}{l}
\vdash \forall E.\\
\quad\quad \texttt{WG}\ E \Longrightarrow\\
\quad\quad \forall P\ u\ P'.\\
\quad\quad\quad E\ P\ -u\rightarrow\ P' \Longrightarrow\\
\quad\quad\quad \exists C'.\ \texttt{CONTEXT}\ C' \land (P' = C'\ P) \land \forall Q.\ E\ Q\ -u\rightarrow\ C'\ Q
\end{array}$$

[OBS_unfolding_lemma3]

$$\begin{array}{l}
\vdash \forall C\ E.\\
\quad\quad \texttt{CONTEXT}\ C \land \texttt{WG}\ E \implies\\
\quad\quad \forall P\ x\ P'.\\
\quad\quad\quad C\ (E\ P)\ {-x\rightarrow}\ P' \implies\\
\quad\quad\quad \exists C'.\ \texttt{CONTEXT}\ C' \land (P' = C'\ P) \land \forall Q.\ C\ (E\ Q)\ {-x\rightarrow}\ C'\ Q
\end{array}$$

[OBS_unfolding_lemma4]

$$\begin{array}{l}
\vdash \forall C\ E\ n\ xs\ P'\ P.\\
\quad\quad \texttt{CONTEXT}\ C \land \texttt{WG}\ E \land \texttt{TRACE}\ ((C \circ \texttt{FUNPOW}\ E\ n)\ P)\ xs\ P' \land\\
\quad\quad \texttt{LENGTH}\ xs \leq n \implies\\
\quad\quad \exists C'.\\
\quad\quad\quad \texttt{CONTEXT}\ C' \land (P' = C'\ P) \land\\
\quad\quad\quad \forall Q.\ \texttt{TRACE}\ ((C \circ \texttt{FUNPOW}\ E\ n)\ Q)\ xs\ (C'\ Q)
\end{array}$$

[OBS_UNIQUE_SOLUTIONS]

$$\begin{array}{l}
\vdash \forall E.\\
\quad\quad \texttt{SG}\ E \land \texttt{SEQ}\ E \implies\\
\quad\quad \forall P\ Q.\ \texttt{OBS\_CONGR}\ P\ (E\ P) \land \texttt{OBS\_CONGR}\ Q\ (E\ Q) \implies \texttt{OBS\_CONGR}\ P\ Q
\end{array}$$

[OBS_UNIQUE_SOLUTIONS_LEMMA]

$$\begin{array}{l}
\vdash \forall G.\\
\quad\quad \texttt{SG}\ G \land \texttt{SEQ}\ G \implies\\
\quad\quad \forall P\ a\ P'.\\
\quad\quad\quad G\ P\ {-a\rightarrow}\ P' \implies\\
\quad\quad\quad \exists H.\\
\quad\quad\quad\quad \texttt{SEQ}\ H \land ((a = \tau) \implies \texttt{SG}\ H) \land (P' = H\ P) \land \forall Q.\ G\ Q\ {-a\rightarrow}\ H\ Q
\end{array}$$

[OBS_UNIQUE_SOLUTIONS_LEMMA_EPS]

$$\begin{array}{l}
\vdash \forall G.\\
\quad\quad \texttt{SG}\ G \land \texttt{SEQ}\ G \implies\\
\quad\quad \forall P\ P'.\\
\quad\quad\quad \texttt{EPS}\ (G\ P)\ P' \implies\\
\quad\quad\quad \exists H.\ \texttt{SG}\ H \land \texttt{SEQ}\ H \land (P' = H\ P) \land \forall Q.\ \texttt{EPS}\ (G\ Q)\ (H\ Q)
\end{array}$$

[STRONG_UNIQUE_SOLUTIONS]

$$\begin{array}{l}
\vdash \forall E.\\
\quad\quad \texttt{WG}\ E \implies\\
\quad\quad \forall P\ Q.\\
\quad\quad\quad \texttt{STRONG\_EQUIV}\ P\ (E\ P) \land \texttt{STRONG\_EQUIV}\ Q\ (E\ Q) \implies\\
\quad\quad\quad \texttt{STRONG\_EQUIV}\ P\ Q
\end{array}$$

[STRONG_UNIQUE_SOLUTIONS_LEMMA]

$$\begin{array}{l}
\vdash \forall E.\\
\quad\quad \texttt{WG}\ E \implies\\
\quad\quad \forall P\ a\ P'.\\
\quad\quad\quad E\ P\ {-a\rightarrow}\ P' \implies\\
\quad\quad\quad \exists E'.\ \texttt{CONTEXT}\ E' \land (P' = E'\ P) \land \forall Q.\ E\ Q\ {-a\rightarrow}\ E'\ Q
\end{array}$$

[unfolding_lemma1]

```
⊢ ∀E C P.
    GCONTEXT E ∧ GCONTEXT C ∧ P contracts E P ⟹
    ∀n. C P contracts (C ∘ FUNPOW E n) P
```

[unfolding_lemma1']

```
⊢ ∀E C P.
    GCONTEXT E ∧ GCONTEXT C ∧ P expands E P ⟹
    ∀n. C P expands (C ∘ FUNPOW E n) P
```

[unfolding_lemma2]

```
⊢ ∀E.
    WGS E ⟹
    ∀P u P′.
      E P −u→ P′ ⟹
      ∃C′. GCONTEXT C′ ∧ (P′ = C′ P) ∧ ∀Q. E Q −u→ C′ Q
```

[unfolding_lemma3]

```
⊢ ∀C E.
    GCONTEXT C ∧ WGS E ⟹
    ∀P x P′.
      C (E P) −x→ P′ ⟹
      ∃C′. GCONTEXT C′ ∧ (P′ = C′ P) ∧ ∀Q. C (E Q) −x→ C′ Q
```

[unfolding_lemma4]

```
⊢ ∀C E n xs P′ P.
    GCONTEXT C ∧ WGS E ∧ TRACE ((C ∘ FUNPOW E n) P) xs P′ ∧
    LENGTH xs ≤ n ⟹
    ∃C′.
      GCONTEXT C′ ∧ (P′ = C′ P) ∧
      ∀Q. TRACE ((C ∘ FUNPOW E n) Q) xs (C′ Q)
```

[UNIQUE_SOLUTIONS_OF_CONTRACTIONS]

```
⊢ ∀E.
    WGS E ⟹
    ∀P Q. P contracts E P ∧ Q contracts E Q ⟹ WEAK_EQUIV P Q
```

[UNIQUE_SOLUTIONS_OF_CONTRACTIONS_LEMMA]

```
⊢ ∀P Q.
    (∃E. WGS E ∧ P contracts E P ∧ Q contracts E Q) ⟹
    ∀C.
      GCONTEXT C ⟹
      (∀l R.
         C P =label l⇒ R ⟹
         ∃C′.
           GCONTEXT C′ ∧ R contracts C′ P ∧
```

```
          (WEAK_EQUIV O (λx y. x =label l⇒ y)) (C Q) (C′ Q)) ∧
    ∀R.
      C P =τ⇒ R ⟹
      ∃C′.
        GCONTEXT C′ ∧ R contracts C′ P ∧
        (WEAK_EQUIV O EPS) (C Q) (C′ Q)
```

[UNIQUE_SOLUTIONS_OF_EXPANSIONS]

```
⊢ ∀E.
    WGS E ⟹
    ∀P Q. P expands E P ∧ Q expands E Q ⟹ WEAK_EQUIV P Q
```

[UNIQUE_SOLUTIONS_OF_EXPANSIONS']

```
⊢ ∀E.
    WGS E ⟹
    ∀P Q. P expands E P ∧ Q expands E Q ⟹ WEAK_EQUIV P Q
```

[UNIQUE_SOLUTIONS_OF_EXPANSIONS_LEMMA]

```
⊢ ∀P Q.
    (∃E. WGS E ∧ P expands E P ∧ Q expands E Q) ⟹
    ∀C.
      GCONTEXT C ⟹
      (∀l R.
         C P =label l⇒ R ⟹
         ∃C′.
           GCONTEXT C′ ∧ R expands C′ P ∧
           (WEAK_EQUIV O (λx y. x =label l⇒ y)) (C Q) (C′ Q)) ∧
      ∀R.
        C P =τ⇒ R ⟹
        ∃C′.
          GCONTEXT C′ ∧ R expands C′ P ∧
          (WEAK_EQUIV O EPS) (C Q) (C′ Q)
```

[UNIQUE_SOLUTIONS_OF_OBS_CONTRACTIONS]

```
⊢ ∀E.
    WG E ⟹
    ∀P Q.
      OBS_contracts P (E P) ∧ OBS_contracts Q (E Q) ⟹
      WEAK_EQUIV P Q
```

[UNIQUE_SOLUTIONS_OF_OBS_CONTRACTIONS']

```
⊢ ∀E.
    WGS E ⟹
    ∀P Q.
      OBS_contracts P (E P) ∧ OBS_contracts Q (E Q) ⟹
      WEAK_EQUIV P Q
```

[UNIQUE_SOLUTIONS_OF_OBS_CONTRACTIONS_LEMMA]

```
 ⊢ ∀P Q.
     (∃E.
        WG E ∧ OBS_contracts P (E P) ∧ OBS_contracts Q (E Q)) ⟹
     ∀C.
       CONTEXT C ⟹
       (∀l R.
          C P =label l⇒ R ⟹
          ∃C′.
            CONTEXT C′ ∧ R contracts C′ P ∧
            (WEAK_EQUIV O (λx y. x =label l⇒ y)) (C Q) (C′ Q)) ∧
       ∀R.
         C P =τ⇒ R ⟹
         ∃C′.
           CONTEXT C′ ∧ R contracts C′ P ∧
           (WEAK_EQUIV O EPS) (C Q) (C′ Q)
```

[WEAK_UNIQUE_SOLUTIONS]

```
 ⊢ ∀E.
     SG E ∧ GSEQ E ⟹
     ∀P Q.
       WEAK_EQUIV P (E P) ∧ WEAK_EQUIV Q (E Q) ⟹ WEAK_EQUIV P Q
```

[WEAK_UNIQUE_SOLUTIONS_LEMMA]

```
 ⊢ ∀G.
     SG G ∧ GSEQ G ⟹
     ∀P a P′.
       G P −a→ P′ ⟹
       ∃H.
         GSEQ H ∧ ((a = τ) ⟹ SG H) ∧ (P′ = H P) ∧
         ∀Q. G Q −a→ H Q
```

[WEAK_UNIQUE_SOLUTIONS_LEMMA_EPS]

```
 ⊢ ∀G.
     SG G ∧ GSEQ G ⟹
     ∀P P′.
       EPS (G P) P′ ⟹
       ∃H. SG H ∧ GSEQ H ∧ (P′ = H P) ∧ ∀Q. EPS (G Q) (H Q)
```

Bibliography

[1] Gordon, M.: From LCF to HOL: a short history. Proof, language, and interaction (July 2000)

[2] Milner, R.: Communication and concurrency. (1989)

[3] Sangiorgi, D.: Equations, contractions, and unique solutions. In: ACM SIGPLAN Notices. Volume 50., ACM (2015) 421–432

[4] Gorrieri, R., Versari, C.: Introduction to Concurrency Theory. Transition Systems and CCS. Springer, Cham (September 2015)

[5] Keller, R.M.: Formal verification of parallel programs. Communications of the Association for Computing Machinery **19**(7) (1976) 371–384

[6] Gorrieri, R.: Process Algebras for Petri Nets. Springer (2017)

[7] Sangiorgi, D., Walker, D.: The pi-calculus: a Theory of Mobile Processes. Cambridge university press (2003)

[8] Bengtson, J.: Formalising process calculi. PhD thesis, Acta Universitatis Upsaliensis (2010)

[9] Nesi, M.: A formalization of the process algebra CCS in high order logic. Technical report, University of Cambridge, Computer Laboratory (1992)

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

[11] Nesi, M.: Formalising a Value-Passing Calculus in HOL. Formal Aspects of Computing **11**(2) (1999) 160–199

[12] Traytel, D., Popescu, A., Blanchette, J.C.: Foundational, compositional (co) datatypes for higher-order logic: Category theory applied to theorem proving. In: Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on, IEEE (2012) 596–605

[13] Gordon, M.J.: Edinburgh LCF: a mechanised logic of computation. (1979)

[14] : The HOL System LOGIC. `http://sourceforge.net/projects/hol/files/hol/kananaskis-11/kananaskis-11-logic.pdf/download` (November 2016) 1–45

[15] Melham, T.F.: Automating Recursive Type Definitions in Higher Order Logic. In: Current Trends in Hardware Verification and Automated Theorem Proving. Springer New York, New York, NY (1989) 341–386

[16] Melham, T.F.: The HOL pred_sets Library. (February 1992) 1–29

[17] Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order. Cambridge University Press (April 2002)

[18] Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press (October 2011)

[19] Melham, T.F.: A Package For Inductive Relation Definitions In HOL. In: 1991., International Workshop on the HOL Theorem Proving System and Its Applications, IEEE (1991) 350–357

[20] : The HOL System DESCRIPTION. (March 2017) 1–349

[21] van Glabbeek, R.J.: A characterisation of weak bisimulation congruence. Lecture notes in computer science **3838** (2005) 26–39

[22] Tian, C.: Further Formalization of the Process Algebra CCS in HOL4. arXiv.org, `https://arxiv.org/abs/1707.04894` (July 2017)

[23] Sangiorgi, D., Milner, R.: The problem of “Weak Bisimulation up to”. CONCUR’92 (1992)

[24] Sangiorgi, D., Milner, R.: The problem of âĂIJweak bisimulation up toâĂİ. In: CONCUR’92, Springer (1992) 32–46

[25] Arun-Kumar, S., Hennessy, M.: An efficiency preorder for processes. Acta Informatica **29**(8) (1992) 737–760

Index