Unique solutions of equations for strong equivalence

Chun Tian

Scuola di Scienze, Università di Bologna
`chun.tian@studio.unibo.it`
Numero di matricola: 0000735539

Abstract. This short paper contains a formal proof of Milner's "unique solution of equations" theorem for strong bisimulation ($\sim$), with the formal proofs of some basic results for strong bisimulation upto $\sim$.

1 Introduction

This work is a prelimitary thesis work, to try to formalize the "unique solution of equations" theorem for strong bisimulation ($\sim$) (Lemma 3.13 and Proposition 14 in Milner's book [1], page 101–103), using HOL theorem prover (HOL4).

The goal is to check if the author is capable to formalize this kind of CCS theorems, in which CCS expressions containing variables were needed, before working on the real thesis topic [2].

Fortunately, based on the author's previous CCS formalization projects [3] [4], this work mentione in this paper was quickly done in two days, including the understanding of informal proofs in Milner's book. All needed results for strong equivalence were already proved in previous work, the only exception is some basic results for "strong equivalence upto $\sim$".

The formal proof in this paper is written in a single Standard ML file (`UniqueSolutionsScript.sml`) (about 820 lines), in which the lemma and proposition of the "unique solution of equations for $\sim$" is so-far the longest single theorems the author ever proved in HOL4.

2 CCS expressions and weakly guarded expressions

For simplicity purposes we only focus on single-variable cases 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 [5], page 181–183). But to formally represent CCS expressions with multiple variables, there may involves new datatypes and many small theorems for recursively replacing those variables in CCS expressions. There're multiple solutions for such kind of formalization, all has equal expression power.

However, if we consider only one variable, it's straightforward to treat expressions like $E(X)$ as a λ-function taking any CCS process and returns another. In this way, no new datatypes were introduced, and actually the single variable X never appears alone in any formal proof, thus no need to represent it as a dedicated object.

In our previous work (the 2nd part), to formlize the theory of congruence for CCS, we ever defined the concept of "one-hole context" based on λ-calculus:

Definition 1. *(Semantic context of CCS) The semantic context (or one-hole context) of CCS is a function $C[\cdot]$ of type "$(\alpha,\ \beta)$ `CCS -> ` $(\alpha,\ \beta)$ `CCS`" recursively defined by following rules:*

$\texttt{CONTEXT}\ (\lambda\, t.\ t)$
$\texttt{CONTEXT}\ c \Rightarrow \texttt{CONTEXT}\ (\lambda\, t.\ a\texttt{..}c\ t)$
$\texttt{CONTEXT}\ c \Rightarrow \texttt{CONTEXT}\ (\lambda\, t.\ c\ t\ \texttt{+}\ x)$
$\texttt{CONTEXT}\ c \Rightarrow \texttt{CONTEXT}\ (\lambda\, t.\ x\ \texttt{+}\ c\ t)$
$\texttt{CONTEXT}\ c \Rightarrow \texttt{CONTEXT}\ (\lambda\, t.\ c\ t\ \texttt{||}\ x)$
$\texttt{CONTEXT}\ c \Rightarrow \texttt{CONTEXT}\ (\lambda\, t.\ x\ \texttt{||}\ c\ t)$
$\texttt{CONTEXT}\ c \Rightarrow \texttt{CONTEXT}\ (\lambda\, t.\ \nu\ L\ (c\ t))$
$\texttt{CONTEXT}\ c \Rightarrow \texttt{CONTEXT}\ (\lambda\, t.\ \texttt{relab}\ (c\ t)\ rf)$

By repeatedly applying above rules, one can imagine that, a "hole" in any CCS term at any depth, can become a λ-function, and by calling the function with another CCS term, the hold is filled by that term.

Based on the idea of semantic contexts, a CCS expression E containing at most one variable X can be seen as a semantic context containing multiple (or zero) holes, which can be defined recursively:

Definition 2. *(Expressions, or multi-hole context)*

$$
\begin{array}{l}
\texttt{EXPR}\ (\lambda t.\ t) \\
\texttt{EXPR}\ (\lambda t.\ p) \\
\texttt{EXPR}\ e \Rightarrow \texttt{EXPR}\ (\lambda t.\ a..e\ t) \\
\texttt{EXPR}\ e_1 \wedge \texttt{EXPR}\ e_2 \Rightarrow \texttt{EXPR}\ (\lambda t.\ e_1\ t + e_2\ t) \\
\texttt{EXPR}\ e_1 \wedge \texttt{EXPR}\ e_2 \Rightarrow \texttt{EXPR}\ (\lambda t.\ e_1\ t\ ||\ e_2\ t) \\
\texttt{EXPR}\ e \Rightarrow \texttt{EXPR}\ (\lambda t.\ \nu\ L\ (e\ t)) \\
\texttt{EXPR}\ e \Rightarrow \texttt{EXPR}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf)
\end{array}
$$

Noticed 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 guardness (or sequential property) as a predicate of CCS expressions, simply because there's no way to recursively check the internal strucuture of λ-functions, as such functions were basically black-boxes once defined. But an obvious workaround solution is to define it independently:

Definition 3. *(Weakly guarded expressions)*

$$
\begin{array}{l}
\texttt{WG}\ (\lambda t.\ a..t) \\
\texttt{WG}\ (\lambda t.\ p) \\
\texttt{WG}\ e \Rightarrow \texttt{WG}\ (\lambda t.\ a..e\ t) \\
\texttt{WG}\ e_1 \wedge \texttt{WG}\ e_2 \Rightarrow \texttt{WG}\ (\lambda t.\ e_1\ t + e_2\ t) \\
\texttt{WG}\ e_1 \wedge \texttt{WG}\ e_2 \Rightarrow \texttt{WG}\ (\lambda t.\ e_1\ t\ ||\ e_2\ t) \\
\texttt{WG}\ e \Rightarrow \texttt{WG}\ (\lambda t.\ \nu\ L\ (e\ t)) \\
\texttt{WG}\ e \Rightarrow \texttt{WG}\ (\lambda t.\ \texttt{relab}\ (e\ t)\ rf)
\end{array}
$$

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-hold context), we have proved their relationships by induction on their structures:

Proposition 1. *1. One-hole contexts are also expressions:*

$$\vdash \texttt{CONTEXT}\ c \Rightarrow \texttt{EXPR}\ c$$

2. Weakly guarded expressions are expreesions:

$$\vdash \texttt{WG}\ e \Rightarrow \texttt{EXPR}\ e$$

Noticed that, the first result is never needed in the rest of proof, but the second one is heavily used.

One last 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 changed to more serious representation of CCS expressions, and adding supports for constants doesn't introduce extra difficulties in the related proofs.

3 Strong bisimulation up to $\sim$

We have defined the concept of "strong bisimulation upto $\sim$" in previous project, but never proved any results about this new concept:

Definition 4. *(Strong bisimulation up to $\sim$)*

```
⊢ 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
```

The following non-trivial lemma is proved (following exactly the same steps in Milner's book). It establishes the relationship between "strong bisimulation upto $\sim$" and "strong bisimulation":

Lemma 1. *If $\mathcal{S}$ is a strong bisimulation up to $\sim$, then $\sim \mathcal{S} \sim$ is a strong bisimulation:*

```
⊢ STRONG_BISIM_UPTO Bsm ⇒
  STRONG_BISIM (STRONG_EQUIV ∘ᵣ Bsm ∘ᵣ STRONG_EQUIV)
```

Based on above lemma, we can then easily prove the following proposition:

Proposition 2. *If $\mathcal{S}$ is a strong bisimulation up to $\sim$, then $\mathcal{S} \subseteq \sim$:*

```
⊢ STRONG_BISIM_UPTO Bsm ⇒ Bsm ⊆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 Unique solution of equations

Based on all above settings, 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 2. *(Lemma 3.13 of [1]) If the variable X are weakly guarded in E, and $E\{P/X\} \xrightarrow{\alpha} P'$, then P' takes the form $E'\{P/X\}$ (for some expression E'), and moreover, for any Q, $E\{Q/X\} \xrightarrow{\alpha} E'\{Q/X\}$:*

```
⊢ WG E ⇒
  ∀ P a P'.
    E P --a-> P' ⇒
    ∃ E'. EXPR 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 1. *(Proposition 3.14 of [1]) Let the expression E contains at most the variable X, and let X be weakly guarded in E, then*

$$\text{If } P \sim E\{P/X\} \text{ and } Q \sim E\{Q/X\} \text{ then } P \sim Q. \tag{1}$$

```
⊢ 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 tacticals, to make the proof not too long in lines).

5 Conclusions

The most difficult part in this work (and the further), is not at those incredible long proof steps in the final target theroem, but the finding of satisfied representations for CCS expressions (with one or many variables), and the related devices to handle these expressions. Usually, having successfully *stated* a theorem with all its needed concepts correctly defined, is already 50% of success in the whole work. And the correctness of these definitions can only be step-by-step verified during the proof of their textbook properties.

Generally speaking, formalizations in process algebras like CCS is not quite hard, at least much easier than most mathematics theories. This is mostly because we are working in a *closed* theorem proving environments in which all needed theorems were created by ourselves and the total number of them were quite small (if comparing to the countless results about real numbers and integers), thus find correct theorems to use is never a problem.

Through this prelimitary work, the author hopes to convince Prof. Sangiorgi that, this whole topic is interested to the author (the student), and the student has ability to finish the potential thesis work in reasonable time, or in even less time.

Grammar issues and typos in this paper were not carefully checked. Apologies for potential reading issues.

References

1. Milner, R.: Communication and concurrency. Prentice Hall (1989)
2. Sangiorgi, D.: Equations, contractions, and unique solutions. ACM SIGPLAN Notices (2015)
3. Tian, C.: A Formalization of the Process Algebra CCS in HOL4. arXiv.org, `http://arxiv.org/abs/1705.07313v2` (May 2017)
4. Tian, C.: Further Formalization of the Process Algebra CCS in HOL4. arXiv.org, `http://arxiv.org/abs/1707.04894v2` (July 2017)
5. Gorrieri, R., Versari, C.: Introduction to Concurrency Theory. Transition Systems and CCS. Springer, Cham (September 2015)

Appendix: proof scripts

A digital version of original proof scripts can be viewed and downloaded from the following GitHub address:

`https://github.com/binghe/informatica-public/tree/master/pre-thesis`

```
(*
 * Copyright 2016-2017  University of Bologna   (Author: Chun Tian)
 *)

open HolKernel Parse boolLib bossLib;

open pred_setTheory relationTheory pairTheory sumTheory listTheory;
open prim_recTheory arithmeticTheory combinTheory;

open CCSLib CCSTheory CCSSyntax CCSConv;
open StrongEQTheory StrongEQLib StrongLawsTheory StrongLawsConv;
open WeakEQTheory WeakEQLib WeakLawsTheory WeakLawsConv;
open ObsCongrTheory ObsCongrLib ObsCongrLawsTheory ObsCongrConv;
open CoarsestCongrTheory;

(* this file contains the theorems of "unique solutions of equations" in CCS *)
val _ = new_theory "UniqueSolutions";

(******************************************************************************)
```

```
(*                                                                            *)
(*                       Strong Bisimulation Upto ~                           *)
(*                                                                            *)
(******************************************************************************)

(* Define the strong bisimulation relation up to STRONG_EQUIV *)
val STRONG_BISIM_UPTO = new_definition (
   "STRONG_BISIM_UPTO",
  ``STRONG_BISIM_UPTO (Bsm :('a, 'b) simulation) =
       (!E E'.
          Bsm E E' ==>
          (!u.
           (!E1. TRANS E u E1 ==>
                 ?E2. TRANS E' u E2 /\ (STRONG_EQUIV O Bsm O STRONG_EQUIV) E1 E2) /\
           (!E2. TRANS E' u E2 ==>
                 ?E1. TRANS E u E1 /\ (STRONG_EQUIV O Bsm O STRONG_EQUIV) E1 E2)))``);

val STRONG_BISIM_UPTO_LEMMA = store_thm (
   "STRONG_BISIM_UPTO_LEMMA",
  ``!Bsm. STRONG_BISIM_UPTO Bsm ==> STRONG_BISIM (STRONG_EQUIV O Bsm O STRONG_EQUIV)``,
    GEN_TAC
 >> REWRITE_TAC [STRONG_BISIM, O_DEF]
 >> REPEAT STRIP_TAC (* 2 sub-goals here *)
 >| [ (* goal 1 (of 2) *)
      Q.PAT_X_ASSUM `STRONG_EQUIV E y'`
        (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\
      POP_ASSUM (STRIP_ASSUME_TAC o (Q.SPEC `u`)) \\
      POP_ASSUM K_TAC \\
      RES_TAC \\
      Q.PAT_X_ASSUM `STRONG_BISIM_UPTO Bsm`
        (STRIP_ASSUME_TAC o (REWRITE_RULE [STRONG_BISIM_UPTO])) \\
      RES_TAC \\
      NTAC 4 (POP_ASSUM K_TAC) \\
      POP_ASSUM (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF])) \\
      Q.PAT_X_ASSUM `STRONG_EQUIV y E'`
        (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\
      POP_ASSUM (STRIP_ASSUME_TAC o (Q.SPEC `u`)) \\
      POP_ASSUM K_TAC \\
      POP_ASSUM (STRIP_ASSUME_TAC o (fn th => MATCH_MP th (ASSUME ``TRANS y u E2``))) \\
(***
   E     ~    y'     Bsm     y     ~   E'
   |        /               \        |
   u       u                 u       u
   |      /                   \      |
   E1 ~ E2 ~ y''' Bsm y'' ~ E2' ~ E2''
 ***)
      `STRONG_EQUIV E1 y'''` by PROVE_TAC [STRONG_EQUIV_TRANS] \\
      `STRONG_EQUIV y'' E2''` by PROVE_TAC [STRONG_EQUIV_TRANS] \\
      Q.EXISTS_TAC `E2''` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y''` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y'''` >> ASM_REWRITE_TAC [],
      (* goal 2 (of 2) *)
      Q.PAT_X_ASSUM `STRONG_EQUIV y E'`
        (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\
      POP_ASSUM (STRIP_ASSUME_TAC o (Q.SPEC `u`)) \\
      Q.PAT_X_ASSUM `!E1. TRANS y u E1 ==> P` K_TAC \\
      RES_TAC \\
      Q.PAT_X_ASSUM `STRONG_BISIM_UPTO Bsm`
```

```
        (STRIP_ASSUME_TAC o (REWRITE_RULE [STRONG_BISIM_UPTO])) \\
      RES_TAC \\
      NTAC 2 (POP_ASSUM K_TAC) \\
      POP_ASSUM (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF])) \\
      Q.PAT_X_ASSUM `STRONG_EQUIV E y'`
        (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\
      POP_ASSUM (STRIP_ASSUME_TAC o (Q.SPEC `u`)) \\
      Q.PAT_X_ASSUM `!E1. TRANS E u E1 ==> P` K_TAC \\
      POP_ASSUM (STRIP_ASSUME_TAC o (fn th => MATCH_MP th (ASSUME ``TRANS y' u E1'``))) \\
(***
   E     ~      y'     Bsm     y    ~    E'
   |           /                \        |
   u          u                  u       u
   |         /                    \      |
   E1'' ~ E1' ~ y''' Bsm y'' ~ E1 ~ E2
 ***)
      `STRONG_EQUIV E1'' y'''` by PROVE_TAC [STRONG_EQUIV_TRANS] \\
      `STRONG_EQUIV y'' E2` by PROVE_TAC [STRONG_EQUIV_TRANS] \\
      Q.EXISTS_TAC `E1''` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y''` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y'''` >> ASM_REWRITE_TAC [] ]);

val STRONG_BISIM_SUBSET_EQUIV = store_thm ((* NEW *)
   "STRONG_BISIM_SUBSET_EQUIV", ``!Bsm. STRONG_BISIM Bsm ==> Bsm RSUBSET STRONG_EQUIV``,
    PROVE_TAC [RSUBSET, STRONG_EQUIV]);

val STRONG_BISIM_UPTO_EQUIV = store_thm (
   "STRONG_BISIM_UPTO_EQUIV",
  ``!Bsm. STRONG_BISIM_UPTO Bsm ==> Bsm RSUBSET STRONG_EQUIV``,
    REPEAT STRIP_TAC
 >> IMP_RES_TAC STRONG_BISIM_UPTO_LEMMA
 >> IMP_RES_TAC STRONG_BISIM_SUBSET_EQUIV
 >> Suff `Bsm RSUBSET (STRONG_EQUIV O Bsm O STRONG_EQUIV)`
 >- ( DISCH_TAC \\
      Know `transitive ((RSUBSET) :('a, 'b) simulation -> ('a, 'b) simulation -> bool)`
      >- PROVE_TAC [RSUBSET_WeakOrder, WeakOrder] \\
      RW_TAC std_ss [transitive_def] >> RES_TAC )
 >> KILL_TAC
 >> REWRITE_TAC [RSUBSET, O_DEF]
 >> REPEAT STRIP_TAC
 >> `STRONG_EQUIV x x /\ STRONG_EQUIV y y` by PROVE_TAC [STRONG_EQUIV_REFL]
 >> Q.EXISTS_TAC `y` >> ASM_REWRITE_TAC []
 >> Q.EXISTS_TAC `x` >> ASM_REWRITE_TAC []
 (* Hence, to prove P ~ Q, we only have to find a strong bisimulation up to ~
    which contains (P, Q) *));

(******************************************************************************)
(*                                                                            *)
(*                      CCS Equations (X = E[X])                              *)
(*                                                                            *)
(******************************************************************************)

(* One-hole context is already defined in CoarsestCongrTheory *)
val (CONTEXT_rules, CONTEXT_ind, CONTEXT_cases) = Hol_reln `
    (                        CONTEXT (\x. x)) /\                    (* CONTEXT1 *)
    (!a c.  CONTEXT c ==> CONTEXT (\t. prefix a (c t))) /\          (* CONTEXT2 *)
    (!x c.  CONTEXT c ==> CONTEXT (\t. sum (c t) x)) /\             (* CONTEXT3 *)
    (!x c.  CONTEXT c ==> CONTEXT (\t. sum x (c t))) /\             (* CONTEXT4 *)
```

```
    (!x c.  CONTEXT c ==> CONTEXT (\t. par (c t) x)) /\          (* CONTEXT5 *)
    (!x c.  CONTEXT c ==> CONTEXT (\t. par x (c t))) /\          (* CONTEXT6 *)
    (!L c.  CONTEXT c ==> CONTEXT (\t. restr L (c t))) /\        (* CONTEXT7 *)
    (!rf c. CONTEXT c ==> CONTEXT (\t. relab (c t) rf)) `;       (* CONTEXT8 *)

(* Weakly guarded (WG) expressions *)
val (WG_rules, WG_ind, WG_cases) = Hol_reln `
    (!a.                           WG (\t. prefix a t)) /\         (* WG1 *)
    (!p.                           WG (\t. p)) /\                  (* WG2 *)
    (!a e.   WG e              ==> WG (\t. prefix a (e t))) /\     (* WG3 *)
    (!e1 e2. WG e1 /\ WG e2    ==> WG (\t. sum (e1 t) (e2 t))) /\ (* WG4 *)
    (!e1 e2. WG e1 /\ WG e2    ==> WG (\t. par (e1 t) (e2 t))) /\ (* WG5 *)
    (!L e.   WG e              ==> WG (\t. restr L (e t))) /\      (* WG6 *)
    (!rf e.  WG e              ==> WG (\t. relab (e t) rf)) `;     (* WG7 *)

val [WG1, WG2, WG3, WG4, WG5, WG6, WG7] =
    map save_thm
        (combine (["WG1", "WG2", "WG3", "WG4", "WG5", "WG6", "WG7"], CONJUNCTS WG_rules));

(* Expressions = multi-hole (or non-hole) contexts. *)
val (EXPR_rules, EXPR_ind, EXPR_cases) = Hol_reln `
    (                                  EXPR (\t. t)) /\                  (* EXPR1 *)
    (!p.                               EXPR (\t. p)) /\                  (* EXPR2 *)
    (!a e.   EXPR e                ==> EXPR (\t. prefix a (e t))) /\     (* EXPR3 *)
    (!e1 e2. EXPR e1 /\ EXPR e2 ==> EXPR (\t. sum (e1 t) (e2 t))) /\ (* EXPR4 *)
    (!e1 e2. EXPR e1 /\ EXPR e2 ==> EXPR (\t. par (e1 t) (e2 t))) /\ (* EXPR5 *)
    (!L e.   EXPR e                ==> EXPR (\t. restr L (e t))) /\      (* EXPR6 *)
    (!rf e.  EXPR e                ==> EXPR (\t. relab (e t) rf)) `;     (* EXPR7 *)

val [EXPR1, EXPR2, EXPR3, EXPR4, EXPR5, EXPR6, EXPR7] =
    map save_thm
        (combine (["EXPR1", "EXPR2", "EXPR3", "EXPR4", "EXPR5", "EXPR6", "EXPR7"],
                  CONJUNCTS EXPR_rules));

val EXPR3a = store_thm ("EXPR3a",
  ``!a :'b Action. EXPR (\t. prefix a t)``,
    ASSUME_TAC EXPR1
 >> IMP_RES_TAC EXPR3
 >> POP_ASSUM MP_TAC
 >> BETA_TAC >> REWRITE_TAC []);

(* One-hole contexts are also expressions *)
val CONTEXT_IS_EXPR = store_thm (
   "CONTEXT_IS_EXPR", ``!c. CONTEXT c ==> EXPR c``,
    Induct_on `CONTEXT`
 >> rpt STRIP_TAC (* 8 sub-goals here *)
 >| [ (* goal 1 (of 8) *)
      REWRITE_TAC [EXPR1],
      (* goal 2 (of 8) *)
      MATCH_MP_TAC EXPR3 >> ASM_REWRITE_TAC [],
      (* goal 3 (of 8) *)
      `EXPR (\y. x)` by REWRITE_TAC [EXPR2] \\
      Know `EXPR (\t. c t + (\y. x) t)`
      >- ( MATCH_MP_TAC EXPR4 >> ASM_REWRITE_TAC [] ) \\
      BETA_TAC >> REWRITE_TAC [],
      (* goal 4 (of 8) *)
      `EXPR (\y. x)` by REWRITE_TAC [EXPR2] \\
      Know `EXPR (\t. (\y. x) t + c t)`
      >- ( MATCH_MP_TAC EXPR4 >> ASM_REWRITE_TAC [] ) \\
      BETA_TAC >> REWRITE_TAC [],
      (* goal 5 (of 8) *)
```

```
      `EXPR (\y. x)` by REWRITE_TAC [EXPR2] \\
      Know `EXPR (\t. c t || (\y. x) t)`
      >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
      BETA_TAC >> REWRITE_TAC [],
      (* goal 6 (of 8) *)
      `EXPR (\y. x)` by REWRITE_TAC [EXPR2] \\
      Know `EXPR (\t. (\y. x) t || c t)`
      >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
      BETA_TAC >> REWRITE_TAC [],
      (* goal 7 (of 8) *)
      MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [],
      (* goal 8 (of 8) *)
      MATCH_MP_TAC EXPR7 >> ASM_REWRITE_TAC [] ]);

(* Weakly guarded expressions are also expressions *)
val WG_IS_EXPR = store_thm (
   "WG_IS_EXPR", ``!e. WG e ==> EXPR e``,
    Induct_on `WG`
 >> rpt STRIP_TAC (* 7 sub-goals here *)
 >- REWRITE_TAC [EXPR3a]
 >- REWRITE_TAC [EXPR2]
 >| [ MATCH_MP_TAC EXPR3,
      MATCH_MP_TAC EXPR4,
      MATCH_MP_TAC EXPR5,
      MATCH_MP_TAC EXPR6,
      MATCH_MP_TAC EXPR7 ]
 >> ASM_REWRITE_TAC []);

(******************************************************************************)
(*
*)
(*                   Unique solutions theorem for ~
*)
(*
*)
(******************************************************************************)

(* Lemma 4.13 (single variable version):
   If the variable X is weakly guarded in E, and E{P/X} --a-> P', then P' takes the form
   E'{P/X} (for some expression E'), and moreover, for any Q, E{Q/X} --a-> E'{Q/X}.
 *)
val STRONG_UNIQUE_SOLUTIONS_LEMMA = store_thm (
   "STRONG_UNIQUE_SOLUTIONS_LEMMA",
  ``!E. WG E ==> !P a P'. TRANS (E P) a P' ==>
                ?E'. EXPR E' /\ (P' = E' P) /\ !Q. TRANS (E Q) a (E' Q)``,
    Induct_on `WG` >> BETA_TAC
 >> COUNT_TAC (rpt STRIP_TAC) (* 7 sub-goals here *)
 >| [ (* goal 1 (of 7) *)
      IMP_RES_TAC TRANS_PREFIX \\
      ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `\x. x` \\
      ASM_REWRITE_TAC [EXPR1] \\
      BETA_TAC >> ASM_REWRITE_TAC [PREFIX],
      (* goal 2 (of 7) *)
      POP_ASSUM (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [TRANS_cases])) \\
      Q.EXISTS_TAC `\t. P'` >> BETA_TAC \\
      ASM_REWRITE_TAC [EXPR2] >| (* 10 sub-goals here *)
      [ REWRITE_TAC [PREFIX],
        MATCH_MP_TAC SUM1 >> ASM_REWRITE_TAC [],
        MATCH_MP_TAC SUM2 >> ASM_REWRITE_TAC [],
        MATCH_MP_TAC PAR1 >> ASM_REWRITE_TAC [],
```

```
      MATCH_MP_TAC PAR2 >> ASM_REWRITE_TAC [],
      MATCH_MP_TAC PAR3 >> Q.EXISTS_TAC `l` >> ASM_REWRITE_TAC [],
      MATCH_MP_TAC RESTR >> Q.EXISTS_TAC `l` >> FULL_SIMP_TAC std_ss [],
      MATCH_MP_TAC RESTR >> Q.EXISTS_TAC `l` >> FULL_SIMP_TAC std_ss [],
      MATCH_MP_TAC RELABELING >> ASM_REWRITE_TAC [],
      MATCH_MP_TAC REC >> ASM_REWRITE_TAC [] ],
    (* goal 3 (of 7) *)
    IMP_RES_TAC TRANS_PREFIX \\
    ASM_REWRITE_TAC [] \\
    Q.EXISTS_TAC `E` >> REWRITE_TAC [] \\
    CONJ_TAC >- IMP_RES_TAC WG_IS_EXPR \\
    REWRITE_TAC [PREFIX],
    (* goal 4 (of 7) *)
    IMP_RES_TAC TRANS_SUM >| (* 2 sub-goals here *)
    [ (* goal 4.1 (of 2) *)
      RES_TAC \\
      Q.EXISTS_TAC `E''` >> ASM_REWRITE_TAC [] \\
      GEN_TAC >> MATCH_MP_TAC SUM1 >> ASM_REWRITE_TAC [],
      (* goal 4.2 (of 2) *)
      RES_TAC \\
      Q.EXISTS_TAC `E''` >> ASM_REWRITE_TAC [] \\
      GEN_TAC >> MATCH_MP_TAC SUM2 >> ASM_REWRITE_TAC [] ],
    (* goal 5 (of 7) *)
    IMP_RES_TAC TRANS_PAR >| (* 3 sub-goals here *)
    [ (* goal 5.1 (of 3) *)
      RES_TAC >> FULL_SIMP_TAC std_ss [] \\
      Q.EXISTS_TAC `\t. par (E'' t) (E' t)` \\
      BETA_TAC >> REWRITE_TAC [] \\
      CONJ_TAC >| (* 2 sub-goals here *)
      [ (* goal 5.1.1 (of 2) *)
        MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] \\
        IMP_RES_TAC WG_IS_EXPR,
        (* goal 5.1.2 (of 2) *)
        GEN_TAC >> MATCH_MP_TAC PAR1 >> ASM_REWRITE_TAC [] ],
      (* goal 5.2 (of 3) *)
      RES_TAC >> FULL_SIMP_TAC std_ss [] \\
      Q.EXISTS_TAC `\t. par (E t) (E'' t)` \\
      BETA_TAC >> REWRITE_TAC [] \\
      CONJ_TAC >| (* 2 sub-goals here *)
      [ (* goal 5.2.1 (of 2) *)
        MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] \\
        IMP_RES_TAC WG_IS_EXPR,
        (* goal 5.2.2 (of 2) *)
        GEN_TAC >> MATCH_MP_TAC PAR2 >> ASM_REWRITE_TAC [] ],
      (* goal 5.3 (of 3) *)
      RES_TAC >> FULL_SIMP_TAC std_ss [] \\
      Q.EXISTS_TAC `\t. par (E''' t) (E'' t)` \\
      BETA_TAC >> REWRITE_TAC [] \\
      CONJ_TAC >| (* 2 sub-goals here *)
      [ (* goal 5.3.1 (of 2) *)
        MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [],
        (* goal 5.3.2 (of 2) *)
        GEN_TAC >> MATCH_MP_TAC PAR3 \\
        Q.EXISTS_TAC `l` >> ASM_REWRITE_TAC [] ] ],
    (* goal 6 (of 7) *)
    IMP_RES_TAC TRANS_RESTR >| (* 2 sub-goals here *)
    [ (* goal 6.1 (of 2) *)
      RES_TAC >> FULL_SIMP_TAC std_ss [] \\
      Q.EXISTS_TAC `\t. restr L (E' t)` >> BETA_TAC >> REWRITE_TAC [] \\
      CONJ_TAC >- ( MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ) \\
      GEN_TAC >> MATCH_MP_TAC RESTR \\
```

```
        FULL_SIMP_TAC std_ss [] \\
        PROVE_TAC [],
        (* goal 6.2 (of 2) *)
        RES_TAC >> FULL_SIMP_TAC std_ss [] \\
        Q.EXISTS_TAC `\t. restr L (E' t)` >> BETA_TAC >> REWRITE_TAC [] \\
        CONJ_TAC >- ( MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ) \\
        GEN_TAC >> MATCH_MP_TAC RESTR \\
        Q.EXISTS_TAC `l` >> FULL_SIMP_TAC std_ss [Action_distinct_label] \\
        PROVE_TAC [] ],
      (* goal 7 (of 7) *)
      IMP_RES_TAC TRANS_RELAB \\
      RES_TAC >> FULL_SIMP_TAC std_ss [] \\
      Q.EXISTS_TAC `\t. relab (E' t) rf` >> BETA_TAC >> REWRITE_TAC [] \\
      CONJ_TAC >- ( MATCH_MP_TAC EXPR7 >> ASM_REWRITE_TAC [] ) \\
      GEN_TAC >> MATCH_MP_TAC RELABELING \\
      ASM_REWRITE_TAC [] ]);

(* Proposition 3.14 (2):
   Let the expression E contains at most the variable X, and let X be weakly guarded in
   each E. Then
       If P ~ E{P/X} and Q ~ E{Q/X} then P ~ Q.
 *)
val STRONG_UNIQUE_SOLUTIONS = store_thm (
   "STRONG_UNIQUE_SOLUTIONS",
  ``!E. WG E ==>
        !P Q. STRONG_EQUIV P (E P) /\ STRONG_EQUIV Q (E Q) ==> STRONG_EQUIV P Q``,
    rpt STRIP_TAC
 >> irule (REWRITE_RULE [RSUBSET] STRONG_BISIM_UPTO_EQUIV)
 >> Q.EXISTS_TAC `\x y. (x = y) \/ (?G. EXPR G /\ (x = G P) /\ (y = G Q))`
 >> BETA_TAC >> REVERSE CONJ_TAC
 >- ( DISJ2_TAC >> Q.EXISTS_TAC `\x. x` >> BETA_TAC \\
      KILL_TAC >> RW_TAC std_ss [EXPR1] )
 >> REWRITE_TAC [STRONG_BISIM_UPTO]
 >> Q.X_GEN_TAC `P'`
 >> Q.X_GEN_TAC `Q'`
 >> BETA_TAC >> STRIP_TAC (* 2 sub-goals here *)
 >- ( ASM_REWRITE_TAC [] >> rpt STRIP_TAC >| (* 2 sub-goals here *)
      [ (* goal 1 (of 2) *)
        Q.EXISTS_TAC `E1` >> ASM_REWRITE_TAC [] \\
        REWRITE_TAC [O_DEF] \\
        Q.EXISTS_TAC `E1` >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\
        Q.EXISTS_TAC `E1` >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\
        BETA_TAC >> DISJ1_TAC >> RW_TAC std_ss [],
        (* goal 2 (of 2) *)
        Q.EXISTS_TAC `E2` >> ASM_REWRITE_TAC [] \\
        REWRITE_TAC [O_DEF] \\
        Q.EXISTS_TAC `E2` >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\
        Q.EXISTS_TAC `E2` >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\
        BETA_TAC >> DISJ1_TAC >> RW_TAC std_ss [] ] )
 >> ASM_REWRITE_TAC []
 >> NTAC 2 (POP_ASSUM K_TAC)
 >> POP_ASSUM MP_TAC
 >> Q.SPEC_TAC (`G`, `G`)
 >> COUNT_TAC (Induct_on `EXPR` >> BETA_TAC >> rpt STRIP_TAC) (* 14 sub-goals here *)
 >| [ (* goal 1 (of 14) *)
      Q.PAT_X_ASSUM `STRONG_EQUIV P (E P)`
        (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\
      RES_TAC \\
      IMP_RES_TAC STRONG_UNIQUE_SOLUTIONS_LEMMA \\ (* lemma used here *)
      FULL_SIMP_TAC std_ss [] \\
      POP_ASSUM (ASSUME_TAC o (Q.SPEC `Q`)) \\
```

```
      Q.PAT_X_ASSUM `STRONG_EQUIV Q (E Q)`
        (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\
      RES_TAC \\
      Q.EXISTS_TAC `E1'` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [O_DEF] \\
      `STRONG_EQUIV (E' Q) E1'` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      Q.EXISTS_TAC `E' Q` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `E' P` >> ASM_REWRITE_TAC [] \\
      BETA_TAC >> DISJ2_TAC \\
      Q.EXISTS_TAC `E'` >> ASM_REWRITE_TAC [],
      (* goal 2 (of 14) *)
      Q.PAT_X_ASSUM `STRONG_EQUIV Q (E Q)`
        (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\
      RES_TAC \\
      IMP_RES_TAC STRONG_UNIQUE_SOLUTIONS_LEMMA \\ (* lemma used here *)
      FULL_SIMP_TAC std_ss [] \\
      POP_ASSUM (ASSUME_TAC o (Q.SPEC `P`)) \\
      Q.PAT_X_ASSUM `STRONG_EQUIV P (E P)`
        (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\
      RES_TAC \\
      Q.EXISTS_TAC `E1` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [O_DEF] \\
      `STRONG_EQUIV (E' P) E1` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      `STRONG_EQUIV (E' Q) E2` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      Q.EXISTS_TAC `E' Q` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `E' P` >> ASM_REWRITE_TAC [] \\
      BETA_TAC >> DISJ2_TAC \\
      Q.EXISTS_TAC `E'` >> ASM_REWRITE_TAC [],

      (* goal 3 (of 14) *)
      Q.EXISTS_TAC `E1` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [O_DEF] \\
      Q.EXISTS_TAC `E1` >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\
      Q.EXISTS_TAC `E1` >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\
      BETA_TAC >> DISJ1_TAC >> RW_TAC std_ss [],
      (* goal 4 (of 14) *)
      Q.EXISTS_TAC `E2` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [O_DEF] \\
      Q.EXISTS_TAC `E2` >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\
      Q.EXISTS_TAC `E2` >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\
      BETA_TAC >> DISJ1_TAC >> RW_TAC std_ss [],

      (* goal 5 (of 14) *)
      IMP_RES_TAC TRANS_PREFIX \\
      FULL_SIMP_TAC std_ss [] \\
      NTAC 2 (POP_ASSUM K_TAC) \\
      Q.EXISTS_TAC `G Q` >> REWRITE_TAC [PREFIX] \\
      REWRITE_TAC [O_DEF] >> BETA_TAC \\
      Q.EXISTS_TAC `G Q` >> REWRITE_TAC [STRONG_EQUIV_REFL] \\
      Q.EXISTS_TAC `G P` >> REWRITE_TAC [STRONG_EQUIV_REFL] \\
      DISJ2_TAC >> Q.EXISTS_TAC `G` >> ASM_REWRITE_TAC [],
      (* goal 6 (of 14) *)
      IMP_RES_TAC TRANS_PREFIX \\
      FULL_SIMP_TAC std_ss [] \\
      NTAC 2 (POP_ASSUM K_TAC) \\
      Q.EXISTS_TAC `G P` >> REWRITE_TAC [PREFIX] \\
      REWRITE_TAC [O_DEF] >> BETA_TAC \\
      Q.EXISTS_TAC `G Q` >> REWRITE_TAC [STRONG_EQUIV_REFL] \\
      Q.EXISTS_TAC `G P` >> REWRITE_TAC [STRONG_EQUIV_REFL] \\
      DISJ2_TAC >> Q.EXISTS_TAC `G` >> ASM_REWRITE_TAC [],

```

```
    (* goal 7 (of 14) *)
    IMP_RES_TAC TRANS_SUM >| (* 2 sub-goals here *)
    [ (* goal 7.1 (of 2) *)
      RES_TAC \\
      Q.EXISTS_TAC `E2` \\
      CONJ_TAC >- ( MATCH_MP_TAC SUM1 >> ASM_REWRITE_TAC [] ) \\
      ASM_REWRITE_TAC [],
      (* goal 7.2 (of 2) *)
      RES_TAC \\
      Q.EXISTS_TAC `E2` \\
      CONJ_TAC >- ( MATCH_MP_TAC SUM2 >> ASM_REWRITE_TAC [] ) \\
      ASM_REWRITE_TAC [] ],
    (* goal 8 (of 14) *)
    IMP_RES_TAC TRANS_SUM >| (* 2 sub-goals here *)
    [ (* goal 8.1 (of 2) *)
      RES_TAC \\
      Q.EXISTS_TAC `E1` \\
      CONJ_TAC >- ( MATCH_MP_TAC SUM1 >> ASM_REWRITE_TAC [] ) \\
      ASM_REWRITE_TAC [],
      (* goal 8.2 (of 2) *)
      RES_TAC \\
      Q.EXISTS_TAC `E1` \\
      CONJ_TAC >- ( MATCH_MP_TAC SUM2 >> ASM_REWRITE_TAC [] ) \\
      ASM_REWRITE_TAC [] ],

    (* goal 9 (of 14) *)
    IMP_RES_TAC TRANS_PAR >> FULL_SIMP_TAC std_ss [] >| (* 3 sub-goals here *)
    [ (* goal 9.1 (of 3) *)
      Q.PAT_X_ASSUM `E1 = X` K_TAC \\
      RES_TAC \\
      Q.EXISTS_TAC `E2 || G' Q` \\
      CONJ_TAC >- ( MATCH_MP_TAC PAR1 >> ASM_REWRITE_TAC [] ) \\
      POP_ASSUM MP_TAC \\
      REWRITE_TAC [O_DEF] >> BETA_TAC \\
      RW_TAC std_ss [] >| (* 2 sub-goals here *)
      [ (* goal 9.1.1 (of 2) *)
        Q.EXISTS_TAC `y || G' Q` \\
        REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                              ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
        Q.EXISTS_TAC `y || G' P` \\
        CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                      ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
        DISJ2_TAC \\
        Q.EXISTS_TAC `\t. y || G' t` >> BETA_TAC >> REWRITE_TAC [] \\
        `EXPR (\z. y)` by REWRITE_TAC [EXPR2] \\
        Know `EXPR (\t. (\z. y) t || G' t)`
        >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
        BETA_TAC >> REWRITE_TAC [],
        (* goal 9.1.2 (of 2) *)
        Q.EXISTS_TAC `G'' Q || G' Q` \\
        REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                              ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
        Q.EXISTS_TAC `G'' P || G' P` \\
        CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                      ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
        DISJ2_TAC \\
        Q.EXISTS_TAC `\t. G'' t || G' t` >> BETA_TAC >> REWRITE_TAC [] \\
        MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ],
      (* goal 9.2 (of 3) *)
      Q.PAT_X_ASSUM `E1 = X` K_TAC \\
      RES_TAC \\
```

```
Q.EXISTS_TAC `G Q || E2` \\
CONJ_TAC >- ( MATCH_MP_TAC PAR2 >> ASM_REWRITE_TAC [] ) \\
POP_ASSUM MP_TAC \\
REWRITE_TAC [O_DEF] >> BETA_TAC \\
RW_TAC std_ss [] >| (* 2 sub-goals here *)
[ (* goal 9.2.1 (of 2) *)
  Q.EXISTS_TAC `G Q || y` \\
  REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                        ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
  Q.EXISTS_TAC `G P || y` \\
  CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
  DISJ2_TAC \\
  Q.EXISTS_TAC `\t. G t || y` >> BETA_TAC >> REWRITE_TAC [] \\
  `EXPR (\z. y)` by REWRITE_TAC [EXPR2] \\
  Know `EXPR (\t. G t || (\z. y) t)`
  >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
  BETA_TAC >> REWRITE_TAC [],
  (* goal 9.2.2 (of 2) *)
  Q.EXISTS_TAC `G Q || G'' Q` \\
  REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                        ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
  Q.EXISTS_TAC `G P || G'' P` \\
  CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
  DISJ2_TAC \\
  Q.EXISTS_TAC `\t. G t || G'' t` >> BETA_TAC >> REWRITE_TAC [] \\
  MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ],
(* goal 9.3 (of 3) *)
Q.PAT_X_ASSUM `E1 = X` K_TAC \\
Q.PAT_X_ASSUM `u = tau` K_TAC \\
RES_TAC \\
Q.EXISTS_TAC `E2'' || E2'` \\
CONJ_TAC >- ( MATCH_MP_TAC PAR3 >> Q.EXISTS_TAC `l` >> ASM_REWRITE_TAC [] ) \\
Q.PAT_X_ASSUM `X E2 E2'` MP_TAC \\
Q.PAT_X_ASSUM `X E1' E2''` MP_TAC \\
REWRITE_TAC [O_DEF] >> BETA_TAC \\
RW_TAC std_ss [] >| (* 4 sub-goals here *)
[ (* goal 9.3.1 (of 4) *)
  Q.EXISTS_TAC `y || y''` \\
  REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC
  Q.EXISTS_TAC `y || y''` \\
  CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\
  DISJ1_TAC >> REWRITE_TAC [],
  (* goal 9.3.2 (of 4) *)
  Q.EXISTS_TAC `y || G'' Q` \\
  REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC
  Q.EXISTS_TAC `y || G'' P` \\
  CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\
  DISJ2_TAC \\
  Q.EXISTS_TAC `\t. y || G'' t` >> BETA_TAC >> REWRITE_TAC [] \\
  `EXPR (\z. y)` by REWRITE_TAC [EXPR2] \\
  Know `EXPR (\t. (\z. y) t || G'' t)`
  >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
  BETA_TAC >> REWRITE_TAC [],
  (* goal 9.3.3 (of 4) *)
  Q.EXISTS_TAC `G'' Q || y''` \\
  REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC
  Q.EXISTS_TAC `G'' P || y''` \\
  CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\
  DISJ2_TAC \\
```

```
      Q.EXISTS_TAC `\t. G'' t || y''` >> BETA_TAC >> REWRITE_TAC [] \\
      `EXPR (\z. y'')` by REWRITE_TAC [EXPR2] \\
      Know `EXPR (\t. G'' t || (\z. y'') t)`
      >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
      BETA_TAC >> REWRITE_TAC [],
      (* goal 9.3.4 (of 4) *)
      Q.EXISTS_TAC `G'' Q || G''' Q` \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC
      Q.EXISTS_TAC `G'' P || G''' P` \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\
      DISJ2_TAC \\
      Q.EXISTS_TAC `\t. G'' t || G''' t` >> BETA_TAC >> REWRITE_TAC [] \\
      MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ] ],
  (* goal 10 (of 14) *)
  IMP_RES_TAC TRANS_PAR >> FULL_SIMP_TAC std_ss [] >| (* 3 sub-goals here *)
  [ (* goal 10.1 (of 3) *)
    Q.PAT_X_ASSUM `E2 = X` K_TAC \\
    RES_TAC \\
    Q.EXISTS_TAC `E1' || G' P` \\
    CONJ_TAC >- ( MATCH_MP_TAC PAR1 >> ASM_REWRITE_TAC [] ) \\
    POP_ASSUM MP_TAC \\
    REWRITE_TAC [O_DEF] >> BETA_TAC \\
    RW_TAC std_ss [] >| (* 2 sub-goals here *)
    [ (* goal 10.1.1 (of 2) *)
      Q.EXISTS_TAC `y || G' Q` \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                            ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
      Q.EXISTS_TAC `y || G' P` \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                    ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
      DISJ2_TAC \\
      Q.EXISTS_TAC `\t. y || G' t` >> BETA_TAC >> REWRITE_TAC [] \\
      `EXPR (\z. y)` by REWRITE_TAC [EXPR2] \\
      Know `EXPR (\t. (\z. y) t || G' t)`
      >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
      BETA_TAC >> REWRITE_TAC [],
      (* goal 10.1.2 (of 2) *)
      Q.EXISTS_TAC `G'' Q || G' Q` \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                            ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
      Q.EXISTS_TAC `G'' P || G' P` \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                    ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
      DISJ2_TAC \\
      Q.EXISTS_TAC `\t. G'' t || G' t` >> BETA_TAC >> REWRITE_TAC [] \\
      MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ],
    (* goal 10.2 (of 3) *)
    Q.PAT_X_ASSUM `E2 = X` K_TAC \\
    RES_TAC \\
    Q.EXISTS_TAC `G P || E1'` \\
    CONJ_TAC >- ( MATCH_MP_TAC PAR2 >> ASM_REWRITE_TAC [] ) \\
    POP_ASSUM MP_TAC \\
    REWRITE_TAC [O_DEF] >> BETA_TAC \\
    RW_TAC std_ss [] >| (* 2 sub-goals here *)
    [ (* goal 10.2.1 (of 2) *)
      Q.EXISTS_TAC `G Q || y` \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                            ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
      Q.EXISTS_TAC `G P || y` \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                    ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
```

```
      DISJ2_TAC \\
      Q.EXISTS_TAC `\t. G t || y` >> BETA_TAC >> REWRITE_TAC [] \\
      `EXPR (\z. y)` by REWRITE_TAC [EXPR2] \\
      Know `EXPR (\t. G t || (\z. y) t)`
      >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
      BETA_TAC >> REWRITE_TAC [],
      (* goal 10.2.2 (of 2) *)
      Q.EXISTS_TAC `G Q || G'' Q` \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                            ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
      Q.EXISTS_TAC `G P || G'' P` \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\
                    ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\
      DISJ2_TAC \\
      Q.EXISTS_TAC `\t. G t || G'' t` >> BETA_TAC >> REWRITE_TAC [] \\
      MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ],
    (* goal 10.3 (of 3) *)
    Q.PAT_X_ASSUM `E2 = X` K_TAC \\
    Q.PAT_X_ASSUM `u = tau` K_TAC \\
    RES_TAC \\
    Q.EXISTS_TAC `E1'' || E1'` \\
    CONJ_TAC >- ( MATCH_MP_TAC PAR3 >> Q.EXISTS_TAC `l` >> ASM_REWRITE_TAC [] ) \\
    Q.PAT_X_ASSUM `X E1'' E1` MP_TAC \\
    Q.PAT_X_ASSUM `X E1' E2'` MP_TAC \\
    REWRITE_TAC [O_DEF] >> BETA_TAC \\
    RW_TAC std_ss [] >| (* 4 sub-goals here *)
    [ (* goal 10.3.1 (of 4) *)
      Q.EXISTS_TAC `y'' || y` \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC
      Q.EXISTS_TAC `y'' || y` \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\
      DISJ1_TAC >> REWRITE_TAC [],
      (* goal 10.3.2 (of 4) *)
      Q.EXISTS_TAC `G'' Q || y` \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC
      Q.EXISTS_TAC `G'' P || y` \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\
      DISJ2_TAC \\
      Q.EXISTS_TAC `\t. G'' t || y` >> BETA_TAC >> REWRITE_TAC [] \\
      `EXPR (\z. y)` by REWRITE_TAC [EXPR2] \\
      Know `EXPR (\t. G'' t || (\z. y) t)`
      >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
      BETA_TAC >> REWRITE_TAC [],
      (* goal 10.3.3 (of 4) *)
      Q.EXISTS_TAC `y'' || G'' Q` \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC
      Q.EXISTS_TAC `y'' || G'' P` \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\
      DISJ2_TAC \\
      Q.EXISTS_TAC `\t. y'' || G'' t` >> BETA_TAC >> REWRITE_TAC [] \\
      `EXPR (\z. y'')` by REWRITE_TAC [EXPR2] \\
      Know `EXPR (\t. (\z. y'') t || G'' t)`
      >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\
      BETA_TAC >> REWRITE_TAC [],
      (* goal 10.3.4 (of 4) *)
      Q.EXISTS_TAC `G''' Q || G'' Q` \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC
      Q.EXISTS_TAC `G''' P || G'' P` \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\
      DISJ2_TAC \\
      Q.EXISTS_TAC `\t. G''' t || G'' t` >> BETA_TAC >> REWRITE_TAC [] \\
```

```
            MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ] ],

        (* goal 11 (of 14) *)
        IMP_RES_TAC TRANS_RESTR >> FULL_SIMP_TAC std_ss [] >| (* 2 sub-goals here *)
        [ (* goal 11.1 (of 2) *)
          RES_TAC \\
          Q.EXISTS_TAC `restr L E2` \\
          CONJ_TAC >- ( MATCH_MP_TAC RESTR >> FULL_SIMP_TAC std_ss [] ) \\
          POP_ASSUM MP_TAC \\
          REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)
          [ (* goal 11.1.1 (of 2) *)
            Q.EXISTS_TAC `restr L y` \\
            REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
            Q.EXISTS_TAC `restr L y` \\
            CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\
            DISJ1_TAC >> REWRITE_TAC [],
            (* goal 11.1.2 (of 2) *)
            Q.EXISTS_TAC `restr L (G' Q)` \\
            REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
            Q.EXISTS_TAC `restr L (G' P)` \\
            CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\
            DISJ2_TAC \\
            Q.EXISTS_TAC `\t. restr L (G' t)` >> BETA_TAC >> REWRITE_TAC [] \\
            MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ],
          (* goal 11.2 (of 2) *)
          RES_TAC \\
          Q.EXISTS_TAC `restr L E2` \\
          CONJ_TAC >- ( MATCH_MP_TAC RESTR >> Q.EXISTS_TAC `l` >> ASM_REWRITE_TAC [] ) \\
          POP_ASSUM MP_TAC \\
          REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)
          [ (* goal 11.2.1 (of 2) *)
            Q.EXISTS_TAC `restr L y` \\
            REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
            Q.EXISTS_TAC `restr L y` \\
            CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\
            DISJ1_TAC >> REWRITE_TAC [],
            (* goal 11.2.2 (of 2) *)
            Q.EXISTS_TAC `restr L (G' Q)` \\
            REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
            Q.EXISTS_TAC `restr L (G' P)` \\
            CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\
            DISJ2_TAC \\
            Q.EXISTS_TAC `\t. restr L (G' t)` >> BETA_TAC >> REWRITE_TAC [] \\
            MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ] ],
        (* goal 12 (of 14) *)
        IMP_RES_TAC TRANS_RESTR >> FULL_SIMP_TAC std_ss [] >| (* 2 sub-goals here *)
        [ (* goal 12.1 (of 2) *)
          RES_TAC \\
          Q.EXISTS_TAC `restr L E1` \\
          CONJ_TAC >- ( MATCH_MP_TAC RESTR >> FULL_SIMP_TAC std_ss [] ) \\
          POP_ASSUM MP_TAC \\
          REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)
          [ (* goal 12.1.1 (of 2) *)
            Q.EXISTS_TAC `restr L y` \\
            REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
            Q.EXISTS_TAC `restr L y` \\
            CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\
            DISJ1_TAC >> REWRITE_TAC [],
            (* goal 12.1.2 (of 2) *)
            Q.EXISTS_TAC `restr L (G' Q)` \\
            REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
```

```
      Q.EXISTS_TAC ‘restr L (G’ P)‘ \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\
      DISJ2_TAC \\
      Q.EXISTS_TAC ‘\t. restr L (G’ t)‘ >> BETA_TAC >> REWRITE_TAC [] \\
      MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ],
    (* goal 12.2 (of 2) *)
    RES_TAC \\
    Q.EXISTS_TAC ‘restr L E1‘ \\
    CONJ_TAC >- ( MATCH_MP_TAC RESTR >> Q.EXISTS_TAC ‘l‘ >> ASM_REWRITE_TAC [] ) \\
    POP_ASSUM MP_TAC \\
    REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)
    [ (* goal 12.2.1 (of 2) *)
      Q.EXISTS_TAC ‘restr L y‘ \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
      Q.EXISTS_TAC ‘restr L y‘ \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\
      DISJ1_TAC >> REWRITE_TAC [],
      (* goal 12.2.2 (of 2) *)
      Q.EXISTS_TAC ‘restr L (G’ Q)‘ \\
      REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
      Q.EXISTS_TAC ‘restr L (G’ P)‘ \\
      CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\
      DISJ2_TAC \\
      Q.EXISTS_TAC ‘\t. restr L (G’ t)‘ >> BETA_TAC >> REWRITE_TAC [] \\
      MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ] ],

  (* goal 13 (of 14) *)
  IMP_RES_TAC TRANS_RELAB \\
  FULL_SIMP_TAC std_ss [] \\
  RES_TAC \\
  Q.EXISTS_TAC ‘relab E2 rf‘ \\
  CONJ_TAC >- ( MATCH_MP_TAC RELABELING >> ASM_REWRITE_TAC [] ) \\
  POP_ASSUM MP_TAC \\
  REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)
  [ (* goal 13.1 (of 2) *)
    Q.EXISTS_TAC ‘relab y rf‘ \\
    REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC []
    Q.EXISTS_TAC ‘relab y rf‘ \\
    CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\
    DISJ1_TAC >> REWRITE_TAC [],
    (* goal 13.2 (of 2) *)
    Q.EXISTS_TAC ‘relab (G’ Q) rf‘ \\
    REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC []
    Q.EXISTS_TAC ‘relab (G’ P) rf‘ \\
    CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\
    DISJ2_TAC \\
    Q.EXISTS_TAC ‘\t. relab (G’ t) rf‘ >> BETA_TAC >> REWRITE_TAC [] \\
    MATCH_MP_TAC EXPR7 >> ASM_REWRITE_TAC [] ],
  (* goal 14 (of 14) *)
  IMP_RES_TAC TRANS_RELAB \\
  FULL_SIMP_TAC std_ss [] \\
  RES_TAC \\
  Q.EXISTS_TAC ‘relab E1 rf‘ \\
  CONJ_TAC >- ( MATCH_MP_TAC RELABELING >> ASM_REWRITE_TAC [] ) \\
  POP_ASSUM MP_TAC \\
  REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)
  [ (* goal 14.1 (of 2) *)
    Q.EXISTS_TAC ‘relab y rf‘ \\
    REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC []
    Q.EXISTS_TAC ‘relab y rf‘ \\
    CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\
```

```
        DISJ1_TAC >> REWRITE_TAC [],
        (* goal 14.2 (of 2) *)
        Q.EXISTS_TAC `relab (G' Q) rf` \\
        REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC []
        Q.EXISTS_TAC `relab (G' P) rf` \\
        CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\
        DISJ2_TAC \\
        Q.EXISTS_TAC `\t. relab (G' t) rf` >> BETA_TAC >> REWRITE_TAC [] \\
        MATCH_MP_TAC EXPR7 >> ASM_REWRITE_TAC [] ] ]);

(** Bibliography:
 *
 * Milner, R.: Communication and concurrency. (1989).
 * Sangiorgi, D.: Equations, contractions, and unique solutions. ACM SIGPLAN Notices. (201
 *)

val _ = export_theory ();
val _ = DB.html_theory "UniqueSolutions";

(* last updated: Aug 3, 2017 *)
```