

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 preliminary 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 mentioned 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 formalize 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 “ $(\alpha, \beta) \text{ CCS} \rightarrow (\alpha, \beta) \text{ CCS}$ ” recursively defined by following rules:

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

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*)

```

EXPR ( $\lambda t. t$ )
EXPR ( $\lambda t. p$ )
EXPR  $e \Rightarrow$  EXPR ( $\lambda t. a..e t$ )
EXPR  $e_1 \wedge$  EXPR  $e_2 \Rightarrow$  EXPR ( $\lambda t. e_1 t + e_2 t$ )
EXPR  $e_1 \wedge$  EXPR  $e_2 \Rightarrow$  EXPR ( $\lambda t. e_1 t || e_2 t$ )
EXPR  $e \Rightarrow$  EXPR ( $\lambda t. \nu L (e t)$ )
EXPR  $e \Rightarrow$  EXPR ( $\lambda t. \text{relab} (e t) rf$ )

```

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 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. But an obvious workaround solution is to define it independently:

Definition 3. (*Weakly guarded expressions*)

```

WG ( $\lambda t. a..t$ )
WG ( $\lambda t. p$ )
WG  $e \Rightarrow$  WG ( $\lambda t. a..e t$ )
WG  $e_1 \wedge$  WG  $e_2 \Rightarrow$  WG ( $\lambda t. e_1 t + e_2 t$ )
WG  $e_1 \wedge$  WG  $e_2 \Rightarrow$  WG ( $\lambda t. e_1 t || e_2 t$ )
WG  $e \Rightarrow$  WG ( $\lambda t. \nu L (e t)$ )
WG  $e \Rightarrow$  WG ( $\lambda t. \text{relab} (e t) rf$ )

```

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

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

2. *Weakly guarded expressions are expressions:*

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

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*)

$$\begin{aligned}
& \vdash \text{STRONG_BISIM_UPTO } Bsm \iff \\
& \quad \forall E \ E'. \\
& \quad \quad Bsm \ E \ E' \Rightarrow \\
& \quad \quad \forall u. \\
& \quad \quad \quad (\forall E_1. \\
& \quad \quad \quad \quad E \ --u-> E_1 \Rightarrow \\
& \quad \quad \quad \quad \exists E_2. \\
& \quad \quad \quad \quad \quad E' \ --u-> E_2 \wedge \\
& \quad \quad \quad \quad \quad (\text{STRONG_EQUIV} \circ_r Bsm \circ_r \text{STRONG_EQUIV}) \ E_1 \ E_2) \wedge \\
& \quad \quad \forall E_2. \\
& \quad \quad \quad E' \ --u-> E_2 \Rightarrow \\
& \quad \quad \quad \exists E_1. \\
& \quad \quad \quad \quad E \ --u-> E_1 \wedge \\
& \quad \quad \quad \quad (\text{STRONG_EQUIV} \circ_r Bsm \circ_r \text{STRONG_EQUIV}) \ E_1 \ E_2
\end{aligned}$$

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:*

$$\begin{aligned}
& \vdash \text{STRONG_BISIM_UPTO } Bsm \Rightarrow \\
& \quad \text{STRONG_BISIM } (\text{STRONG_EQUIV} \circ_r Bsm \circ_r \text{STRONG_EQUIV})
\end{aligned}$$

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 $\mathcal{S} \subseteq \sim$:*

$$\vdash \text{STRONG_BISIM_UPTO } Bsm \Rightarrow Bsm \subseteq_r \text{STRONG_EQUIV}$$

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 , $E\{Q/X\} \xrightarrow{\alpha} E'\{Q/X\}$:*

$$\begin{aligned}
& \vdash \text{WG } E \Rightarrow \\
& \quad \forall P \ a \ P'. \\
& \quad \quad E \ P \ --a-> P' \Rightarrow \\
& \quad \quad \exists E'. \text{EXPR } E' \wedge P' = E' \ P \wedge \forall Q. E \ Q \ --a-> E' \ Q
\end{aligned}$$

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*

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

$$\vdash \text{WG } E \Rightarrow \forall P \ Q. P \sim E \ P \wedge Q \sim E \ Q \Rightarrow P \sim Q$$

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 theorem, 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 preliminary 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>

```
1  (*
2   * Copyright 2016-2017 University of Bologna (Author: Chun Tian)
3   *)
4
5  open HolKernel Parse boolLib bossLib;
6
7  open pred_setTheory relationTheory pairTheory sumTheory listTheory;
8  open prim_recTheory arithmeticTheory combinTheory;
9
10 open CCSLib CCSTheory CCSSyntax CCSConv;
11 open StrongEQTheory StrongEQLib StrongLawsTheory StrongLawsConv;
12 open WeakEQTheory WeakEQLib WeakLawsTheory WeakLawsConv;
13 open ObsCongrTheory ObsCongrLib ObsCongrLawsTheory ObsCongrConv;
14 open CoarsestCongrTheory;
15
16 (* this file contains the theorems of "unique solutions of equations" in CCS *)
17 val _ = new_theory "UniqueSolutions";
18
19 (*****)
```

```

20  (*)
21  (*)
22  (*)
23  (*****)
24
25  (* Define the strong bisimulation relation up to STRONG_EQUIV *)
26  val STRONG_BISIM_UPTO = new_definition (
27    "STRONG_BISIM_UPTO",
28    'STRONG_BISIM_UPTO (Bsm :('a, 'b) simulation) =
29      (!E E'.
30        Bsm E E' ==>
31        (!u.
32          (!E1. TRANS E u E1 ==>
33            ?E2. TRANS E' u E2 /\ (STRONG_EQUIV 0 Bsm 0 STRONG_EQUIV) E1 E2) /\
34            (!E2. TRANS E' u E2 ==>
35              ?E1. TRANS E u E1 /\ (STRONG_EQUIV 0 Bsm 0 STRONG_EQUIV) E1 E2)))'');
36
37  val STRONG_BISIM_UPTO_LEMMA = store_thm (
38    "STRONG_BISIM_UPTO_LEMMA",
39    '(!Bsm. STRONG_BISIM_UPTO Bsm ==> STRONG_BISIM (STRONG_EQUIV 0 Bsm 0 STRONG_EQUIV))',
40    GEN_TAC
41  >> REWRITE_TAC [STRONG_BISIM, O_DEF]
42  >> REPEAT STRIP_TAC (* 2 sub-goals here *)
43  >| [ (* goal 1 (of 2) *)
44      Q.PAT_X_ASSUM 'STRONG_EQUIV E y',
45      (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \
46      POP_ASSUM (STRIP_ASSUME_TAC o (Q.SPEC 'u')) \
47      POP_ASSUM K_TAC \
48      RES_TAC \
49      Q.PAT_X_ASSUM 'STRONG_BISIM_UPTO Bsm',
50      (STRIP_ASSUME_TAC o (REWRITE_RULE [STRONG_BISIM_UPTO])) \
51      RES_TAC \
52      NTAC 4 (POP_ASSUM K_TAC) \
53      POP_ASSUM (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF])) \
54      Q.PAT_X_ASSUM 'STRONG_EQUIV y E',
55      (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \
56      POP_ASSUM (STRIP_ASSUME_TAC o (Q.SPEC 'u')) \
57      POP_ASSUM K_TAC \
58      POP_ASSUM (STRIP_ASSUME_TAC o (fn th => MATCH_MP th (ASSUME 'TRANS y u E2')))) \
59  (**
60      E      ~      y'      Bsm      y      ~      E'
61      /      /      \      /
62      u      u      u      u
63      /      /      \      /
64      E1 ~ E2 ~ y''' Bsm y'' ~ E2' ~ E2''
65  ***)
66      'STRONG_EQUIV E1 y''' by PROVE_TAC [STRONG_EQUIV_TRANS] \
67      'STRONG_EQUIV y'' E2''' by PROVE_TAC [STRONG_EQUIV_TRANS] \
68      Q.EXISTS_TAC 'E2''' >> ASM_REWRITE_TAC [] \
69      Q.EXISTS_TAC 'y''' >> ASM_REWRITE_TAC [] \
70      Q.EXISTS_TAC 'y''' >> ASM_REWRITE_TAC [],
71    (* goal 2 (of 2) *)
72    Q.PAT_X_ASSUM 'STRONG_EQUIV y E',
73    (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \
74    POP_ASSUM (STRIP_ASSUME_TAC o (Q.SPEC 'u')) \
75    Q.PAT_X_ASSUM '!E1. TRANS y u E1 ==> P' K_TAC \
76    RES_TAC \
77    Q.PAT_X_ASSUM 'STRONG_BISIM_UPTO Bsm'

```

```

78      (STRIP_ASSUME_TAC o (REWRITE_RULE [STRONG_BISIM_UPTO])) \\  

79      RES_TAC \\  

80      NTAC 2 (POP_ASSUM K_TAC) \\  

81      POP_ASSUM (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF])) \\  

82      Q.PAT_X_ASSUM 'STRONG_EQUIV E y'  

83      (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\  

84      POP_ASSUM (STRIP_ASSUME_TAC o (Q.SPEC 'u')) \\  

85      Q.PAT_X_ASSUM '!E1. TRANS E u E1 ==> P' K_TAC \\  

86      POP_ASSUM (STRIP_ASSUME_TAC o (fn th => MATCH_MP th (ASSUME 'TRANS y' u E1'')))) \\  

87  (**  

88      E      ~      y'      Bsm      y      ~      E'  

89      /      /      \      /  

90      u      u      u      u  

91      /      /      \      /  

92      E1'' ~ E1' ~ y'' Bsm y'' ~ E1 ~ E2  

93  ***)  

94      'STRONG_EQUIV E1'' y'' by PROVE_TAC [STRONG_EQUIV_TRANS] \\  

95      'STRONG_EQUIV y'' E2' by PROVE_TAC [STRONG_EQUIV_TRANS] \\  

96      Q.EXISTS_TAC 'E1'' >> ASM_REWRITE_TAC [] \\  

97      Q.EXISTS_TAC 'y'' >> ASM_REWRITE_TAC [] \\  

98      Q.EXISTS_TAC 'y'' >> ASM_REWRITE_TAC [] ]];  

99  

100  val STRONG_BISIM_SUBSET_EQUIV = store_thm ((* NEW *)  

101      "STRONG_BISIM_SUBSET_EQUIV", '!Bsm. STRONG_BISIM Bsm ==> Bsm RSUBSET STRONG_EQUIV',  

102      PROVE_TAC [RSUBSET, STRONG_EQUIV]);  

103  

104  val STRONG_BISIM_UPTO_EQUIV = store_thm (  

105      "STRONG_BISIM_UPTO_EQUIV",  

106      '!Bsm. STRONG_BISIM_UPTO Bsm ==> Bsm RSUBSET STRONG_EQUIV',  

107      REPEAT STRIP_TAC  

108      >> IMP_RES_TAC STRONG_BISIM_UPTO_LEMMA  

109      >> IMP_RES_TAC STRONG_BISIM_SUBSET_EQUIV  

110      >> Suff 'Bsm RSUBSET (STRONG_EQUIV O Bsm O STRONG_EQUIV)'  

111      >- ( DISCH_TAC \\  

112          Know 'transitive ((RSUBSET) :('a, 'b) simulation -> ('a, 'b) simulation -> bool)'  

113          >- PROVE_TAC [RSUBSET_WeakOrder, WeakOrder] \\  

114          RW_TAC std_ss [transitive_def] >> RES_TAC )  

115      >> KILL_TAC  

116      >> REWRITE_TAC [RSUBSET, O_DEF]  

117      >> REPEAT STRIP_TAC  

118      >> 'STRONG_EQUIV x x /\ STRONG_EQUIV y y' by PROVE_TAC [STRONG_EQUIV_REFL]  

119      >> Q.EXISTS_TAC 'y' >> ASM_REWRITE_TAC []  

120      >> Q.EXISTS_TAC 'x' >> ASM_REWRITE_TAC []  

121      (* Hence, to prove P ~ Q, we only have to find a strong bisimulation up to ~  

122         which contains (P, Q) *));  

123  

124  (*****  

125  (*  

126  *)  

127  (*  

128  *)  

129  (*****  

130  (* One-hole context is already defined in CoarsestCongrTheory *)  

131  val (CONTEXT_rules, CONTEXT_ind, CONTEXT_cases) = Hol_reln '  

132      (  

133          CONTEXT (\x. x)) /\ (* CONTEXT1 *)  

134          (!a c. CONTEXT c ==> CONTEXT (\t. prefix a (c t))) /\ (* CONTEXT2 *)  

135          (!x c. CONTEXT c ==> CONTEXT (\t. sum (c t) x)) /\ (* CONTEXT3 *)  

136          (!x c. CONTEXT c ==> CONTEXT (\t. sum x (c t))) /\ (* CONTEXT4 *)

```

```

136      (!x c. CONTEXT c ==> CONTEXT (\t. par (c t) x)) /\          (* CONTEXT5 *)
137      (!x c. CONTEXT c ==> CONTEXT (\t. par x (c t))) /\          (* CONTEXT6 *)
138      (!L c. CONTEXT c ==> CONTEXT (\t. restr L (c t))) /\        (* CONTEXT7 *)
139      (!rf c. CONTEXT c ==> CONTEXT (\t. relab (c t) rf)) ‘;      (* CONTEXT8 *)
140
141  (* Weakly guarded (WG) expressions *)
142  val (WG_rules, WG_ind, WG_cases) = Hol_reln ‘
143      (!a.          WG (\t. prefix a t)) /\          (* WG1 *)
144      (!p.          WG (\t. p)) /\          (* WG2 *)
145      (!a e. WG e    ==> WG (\t. prefix a (e t))) /\  (* WG3 *)
146      (!e1 e2. WG e1 /\ WG e2 ==> WG (\t. sum (e1 t) (e2 t))) /\ (* WG4 *)
147      (!e1 e2. WG e1 /\ WG e2 ==> WG (\t. par (e1 t) (e2 t))) /\ (* WG5 *)
148      (!L e. WG e    ==> WG (\t. restr L (e t))) /\  (* WG6 *)
149      (!rf e. WG e    ==> WG (\t. relab (e t) rf)) ‘;  (* WG7 *)
150
151  val [WG1, WG2, WG3, WG4, WG5, WG6, WG7] =
152      map save_thm
153          (combine (["WG1", "WG2", "WG3", "WG4", "WG5", "WG6", "WG7"], CONJUNCTS WG_rules));
154
155  (* Expressions = multi-hole (or non-hole) contexts. *)
156  val (EXPR_rules, EXPR_ind, EXPR_cases) = Hol_reln ‘
157      (          EXPR (\t. t)) /\          (* EXPR1 *)
158      (!p.      EXPR (\t. p)) /\          (* EXPR2 *)
159      (!a e.    EXPR e    ==> EXPR (\t. prefix a (e t))) /\  (* EXPR3 *)
160      (!e1 e2. EXPR e1 /\ EXPR e2 ==> EXPR (\t. sum (e1 t) (e2 t))) /\ (* EXPR4 *)
161      (!e1 e2. EXPR e1 /\ EXPR e2 ==> EXPR (\t. par (e1 t) (e2 t))) /\ (* EXPR5 *)
162      (!L e.    EXPR e    ==> EXPR (\t. restr L (e t))) /\  (* EXPR6 *)
163      (!rf e.    EXPR e    ==> EXPR (\t. relab (e t) rf)) ‘;  (* EXPR7 *)
164
165  val [EXPR1, EXPR2, EXPR3, EXPR4, EXPR5, EXPR6, EXPR7] =
166      map save_thm
167          (combine (["EXPR1", "EXPR2", "EXPR3", "EXPR4", "EXPR5", "EXPR6", "EXPR7"],
168                  CONJUNCTS EXPR_rules));
169
170  val EXPR3a = store_thm ("EXPR3a",
171      ‘!a :’b Action. EXPR (\t. prefix a t)‘,
172      ASSUME_TAC EXPR1
173  >> IMP_RES_TAC EXPR3
174  >> POP_ASSUM MP_TAC
175  >> BETA_TAC >> REWRITE_TAC []);
176
177  (* One-hole contexts are also expressions *)
178  val CONTEXT_IS_EXPR = store_thm (
179      "CONTEXT_IS_EXPR", ‘!c. CONTEXT c ==> EXPR c‘,
180      Induct_on ‘CONTEXT‘
181  >> rpt STRIP_TAC (* 8 sub-goals here *)
182  >| [ (* goal 1 (of 8) *)
183      REWRITE_TAC [EXPR1],
184      (* goal 2 (of 8) *)
185      MATCH_MP_TAC EXPR3 >> ASM_REWRITE_TAC [],
186      (* goal 3 (of 8) *)
187      ‘EXPR (\y. x)‘ by REWRITE_TAC [EXPR2] \\\
188      Know ‘EXPR (\t. c t + (\y. x) t)‘
189  >- ( MATCH_MP_TAC EXPR4 >> ASM_REWRITE_TAC [] ) \\\
190      BETA_TAC >> REWRITE_TAC [],
191      (* goal 4 (of 8) *)
192      ‘EXPR (\y. x)‘ by REWRITE_TAC [EXPR2] \\\
193      Know ‘EXPR (\t. (\y. x) t + c t)‘
194  >- ( MATCH_MP_TAC EXPR4 >> ASM_REWRITE_TAC [] ) \\\
195      BETA_TAC >> REWRITE_TAC [],
196      (* goal 5 (of 8) *)

```

```

197     'EXPR (\y. x)' by REWRITE_TAC [EXPR2] \\  

198     Know 'EXPR (\t. c t || (\y. x) t)'\  

199     >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\  

200     BETA_TAC >> REWRITE_TAC [],  

201     (* goal 6 (of 8) *)  

202     'EXPR (\y. x)' by REWRITE_TAC [EXPR2] \\  

203     Know 'EXPR (\t. (\y. x) t || c t)'\  

204     >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\  

205     BETA_TAC >> REWRITE_TAC [],  

206     (* goal 7 (of 8) *)  

207     MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [],  

208     (* goal 8 (of 8) *)  

209     MATCH_MP_TAC EXPR7 >> ASM_REWRITE_TAC [] ]);  

210  

211 (* Weakly guarded expressions are also expressions *)  

212 val WG_IS_EXPR = store_thm (  

213     "WG_IS_EXPR", '!'e. WG e ==> EXPR e',  

214     Induct_on 'WG'  

215     >> rpt STRIP_TAC (* 7 sub-goals here *)  

216     >- REWRITE_TAC [EXPR3a]  

217     >- REWRITE_TAC [EXPR2]  

218     >| [ MATCH_MP_TAC EXPR3,  

219         MATCH_MP_TAC EXPR4,  

220         MATCH_MP_TAC EXPR5,  

221         MATCH_MP_TAC EXPR6,  

222         MATCH_MP_TAC EXPR7 ]  

223     >> ASM_REWRITE_TAC []);  

224  

225 (*****)  

226 (*  

227 *)  

228 (*  

229 *)  

230  

231 (* Lemma 4.13 (single variable version):  

232     If the variable X is weakly guarded in E, and  $E\{P/X\} \dashv\vdash P'$ , then  $P'$  takes the form  

233      $E'\{P/X\}$  (for some expression  $E'$ ), and moreover, for any  $Q$ ,  $E\{Q/X\} \dashv\vdash E'\{Q/X\}$ .  

234 *)  

235 val STRONG_UNIQUE_SOLUTIONS_LEMMA = store_thm (  

236     "STRONG_UNIQUE_SOLUTIONS_LEMMA",  

237     '!'E. WG E ==> !P a P'. TRANS (E P) a P' ==>  

238         ?E'. EXPR E' /\ (P' = E' P) /\ !Q. TRANS (E Q) a (E' Q)',  

239     Induct_on 'WG' >> BETA_TAC  

240     >> COUNT_TAC (rpt STRIP_TAC) (* 7 sub-goals here *)  

241     >| [ (* goal 1 (of 7) *)  

242         IMP_RES_TAC TRANS_PREFIX \\  

243         ASM_REWRITE_TAC [] \\  

244         Q.EXISTS_TAC '\x. x' \\  

245         ASM_REWRITE_TAC [EXPR1] \\  

246         BETA_TAC >> ASM_REWRITE_TAC [PREFIX],  

247         (* goal 2 (of 7) *)  

248         POP_ASSUM (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [TRANS_cases])) \\  

249         Q.EXISTS_TAC '\t. P' >> BETA_TAC \\  

250         ASM_REWRITE_TAC [EXPR2] >| (* 10 sub-goals here *)  

251         [ REWRITE_TAC [PREFIX],  

252           MATCH_MP_TAC SUM1 >> ASM_REWRITE_TAC [],  

253           MATCH_MP_TAC SUM2 >> ASM_REWRITE_TAC [],  

254           MATCH_MP_TAC PAR1 >> ASM_REWRITE_TAC [],

```



```

255     MATCH_MP_TAC PAR2 >> ASM_REWRITE_TAC [],
256     MATCH_MP_TAC PAR3 >> Q.EXISTS_TAC '1' >> ASM_REWRITE_TAC [],
257     MATCH_MP_TAC RESTR >> Q.EXISTS_TAC '1' >> FULL_SIMP_TAC std_ss [],
258     MATCH_MP_TAC RESTR >> Q.EXISTS_TAC '1' >> FULL_SIMP_TAC std_ss [],
259     MATCH_MP_TAC RELABELING >> ASM_REWRITE_TAC [],
260     MATCH_MP_TAC REC >> ASM_REWRITE_TAC [] ],
261   (* goal 3 (of 7) *)
262   IMP_RES_TAC TRANS_PREFIX \\
263   ASM_REWRITE_TAC [] \\
264   Q.EXISTS_TAC 'E' >> REWRITE_TAC [] \\
265   CONJ_TAC >- IMP_RES_TAC WG_IS_EXPR \\
266   REWRITE_TAC [PREFIX],
267   (* goal 4 (of 7) *)
268   IMP_RES_TAC TRANS_SUM >| (* 2 sub-goals here *)
269   [ (* goal 4.1 (of 2) *)
270     RES_TAC \\
271     Q.EXISTS_TAC 'E'' >> ASM_REWRITE_TAC [] \\
272     GEN_TAC >> MATCH_MP_TAC SUM1 >> ASM_REWRITE_TAC [],
273     (* goal 4.2 (of 2) *)
274     RES_TAC \\
275     Q.EXISTS_TAC 'E'' >> ASM_REWRITE_TAC [] \\
276     GEN_TAC >> MATCH_MP_TAC SUM2 >> ASM_REWRITE_TAC [] ],
277   (* goal 5 (of 7) *)
278   IMP_RES_TAC TRANS_PAR >| (* 3 sub-goals here *)
279   [ (* goal 5.1 (of 3) *)
280     RES_TAC >> FULL_SIMP_TAC std_ss [] \\
281     Q.EXISTS_TAC '\t. par (E'' t) (E' t)' \\
282     BETA_TAC >> REWRITE_TAC [] \\
283     CONJ_TAC >| (* 2 sub-goals here *)
284     [ (* goal 5.1.1 (of 2) *)
285       MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] \\
286       IMP_RES_TAC WG_IS_EXPR,
287       (* goal 5.1.2 (of 2) *)
288       GEN_TAC >> MATCH_MP_TAC PAR1 >> ASM_REWRITE_TAC [] ],
289     (* goal 5.2 (of 3) *)
290     RES_TAC >> FULL_SIMP_TAC std_ss [] \\
291     Q.EXISTS_TAC '\t. par (E t) (E'' t)' \\
292     BETA_TAC >> REWRITE_TAC [] \\
293     CONJ_TAC >| (* 2 sub-goals here *)
294     [ (* goal 5.2.1 (of 2) *)
295       MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] \\
296       IMP_RES_TAC WG_IS_EXPR,
297       (* goal 5.2.2 (of 2) *)
298       GEN_TAC >> MATCH_MP_TAC PAR2 >> ASM_REWRITE_TAC [] ],
299     (* goal 5.3 (of 3) *)
300     RES_TAC >> FULL_SIMP_TAC std_ss [] \\
301     Q.EXISTS_TAC '\t. par (E'' t) (E'' t)' \\
302     BETA_TAC >> REWRITE_TAC [] \\
303     CONJ_TAC >| (* 2 sub-goals here *)
304     [ (* goal 5.3.1 (of 2) *)
305       MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [],
306       (* goal 5.3.2 (of 2) *)
307       GEN_TAC >> MATCH_MP_TAC PAR3 \\
308       Q.EXISTS_TAC '1' >> ASM_REWRITE_TAC [] ] ],
309   (* goal 6 (of 7) *)
310   IMP_RES_TAC TRANS_RESTR >| (* 2 sub-goals here *)
311   [ (* goal 6.1 (of 2) *)
312     RES_TAC >> FULL_SIMP_TAC std_ss [] \\
313     Q.EXISTS_TAC '\t. restr L (E' t)' >> BETA_TAC >> REWRITE_TAC [] \\
314     CONJ_TAC >- ( MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ) \\
315     GEN_TAC >> MATCH_MP_TAC RESTR \\

```

```

316     FULL_SIMP_TAC std_ss [] \\  

317     PROVE_TAC [],  

318     (* goal 6.2 (of 2) *)  

319     RES_TAC >> FULL_SIMP_TAC std_ss [] \\  

320     Q.EXISTS_TAC '\t. restr L (E' t)' >> BETA_TAC >> REWRITE_TAC [] \\  

321     CONJ_TAC >- ( MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ) \\  

322     GEN_TAC >> MATCH_MP_TAC RESTR \\  

323     Q.EXISTS_TAC '1' >> FULL_SIMP_TAC std_ss [Action_distinct_label] \\  

324     PROVE_TAC [] ],  

325     (* goal 7 (of 7) *)  

326     IMP_RES_TAC TRANS_RELAB \\  

327     RES_TAC >> FULL_SIMP_TAC std_ss [] \\  

328     Q.EXISTS_TAC '\t. relab (E' t) rf' >> BETA_TAC >> REWRITE_TAC [] \\  

329     CONJ_TAC >- ( MATCH_MP_TAC EXPR7 >> ASM_REWRITE_TAC [] ) \\  

330     GEN_TAC >> MATCH_MP_TAC RELABELING \\  

331     ASM_REWRITE_TAC [] ]);  

332  

333 (* Proposition 3.14 (2):  

334    Let the expression E contains at most the variable X, and let X be weakly guarded in  

335    each E. Then  

336    If  $P \sim E\{P/X\}$  and  $Q \sim E\{Q/X\}$  then  $P \sim Q$ .  

337 *)  

338 val STRONG_UNIQUE_SOLUTIONS = store_thm (  

339     "STRONG_UNIQUE_SOLUTIONS",  

340     '!E. WG E ==>  

341         !P Q. STRONG_EQUIV P (E P) /\ STRONG_EQUIV Q (E Q) ==> STRONG_EQUIV P Q',  

342     rpt STRIP_TAC  

343     >> irule (REWRITE_RULE [RSUBSET] STRONG_BISIM_UPTO_EQUIV)  

344     >> Q.EXISTS_TAC '\x y. (x = y) /\ (?G. EXPR G /\ (x = G P) /\ (y = G Q))'  

345     >> BETA_TAC >> REVERSE CONJ_TAC  

346     >- ( DISJ2_TAC >> Q.EXISTS_TAC '\x. x' >> BETA_TAC \\  

347         KILL_TAC >> RW_TAC std_ss [EXPR1] )  

348     >> REWRITE_TAC [STRONG_BISIM_UPTO]  

349     >> Q.X_GEN_TAC 'P'  

350     >> Q.X_GEN_TAC 'Q'  

351     >> BETA_TAC >> STRIP_TAC (* 2 sub-goals here *)  

352     >- ( ASM_REWRITE_TAC [] >> rpt STRIP_TAC >| (* 2 sub-goals here *)  

353         [ (* goal 1 (of 2) *)  

354             Q.EXISTS_TAC 'E1' >> ASM_REWRITE_TAC [] \\  

355             REWRITE_TAC [O_DEF] \\  

356             Q.EXISTS_TAC 'E1' >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\  

357             Q.EXISTS_TAC 'E1' >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\  

358             BETA_TAC >> DISJ1_TAC >> RW_TAC std_ss [],  

359             (* goal 2 (of 2) *)  

360             Q.EXISTS_TAC 'E2' >> ASM_REWRITE_TAC [] \\  

361             REWRITE_TAC [O_DEF] \\  

362             Q.EXISTS_TAC 'E2' >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\  

363             Q.EXISTS_TAC 'E2' >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\  

364             BETA_TAC >> DISJ1_TAC >> RW_TAC std_ss [] ] )  

365     >> ASM_REWRITE_TAC []  

366     >> NTAC 2 (POP_ASSUM K_TAC)  

367     >> POP_ASSUM MP_TAC  

368     >> Q.SPEC_TAC ('G', 'G')  

369     >> COUNT_TAC (Induct_on 'EXPR' >> BETA_TAC >> rpt STRIP_TAC) (* 14 sub-goals here *)  

370     >| [ (* goal 1 (of 14) *)  

371         Q.PAT_X_ASSUM 'STRONG_EQUIV P (E P)'  

372         (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\  

373         RES_TAC \\  

374         IMP_RES_TAC STRONG_UNIQUE_SOLUTIONS_LEMMA \\  

375         FULL_SIMP_TAC std_ss [] \\  

376         POP_ASSUM (ASSUME_TAC o (Q.SPEC 'Q')) \\  


```

```

377 Q.PAT_X_ASSUM 'STRONG_EQUIV Q (E Q)'
378 (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\\
379 RES_TAC \\\
380 Q.EXISTS_TAC 'E1' >> ASM_REWRITE_TAC [] \\\
381 REWRITE_TAC [O_DEF] \\\
382 'STRONG_EQUIV (E' Q) E1' by PROVE_TAC [STRONG_EQUIV_SYM] \\\
383 Q.EXISTS_TAC 'E' Q' >> ASM_REWRITE_TAC [] \\\
384 Q.EXISTS_TAC 'E' P' >> ASM_REWRITE_TAC [] \\\
385 BETA_TAC >> DISJ2_TAC \\\
386 Q.EXISTS_TAC 'E' >> ASM_REWRITE_TAC [],
387 (* goal 2 (of 14) *)
388 Q.PAT_X_ASSUM 'STRONG_EQUIV Q (E Q)'
389 (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\\
390 RES_TAC \\\
391 IMP_RES_TAC STRONG_UNIQUE_SOLUTIONS_LEMMA \\\ (* lemma used here *)
392 FULL_SIMP_TAC std_ss [] \\\
393 POP_ASSUM (ASSUME_TAC o (Q.SPEC 'P')) \\\
394 Q.PAT_X_ASSUM 'STRONG_EQUIV P (E P)'
395 (STRIP_ASSUME_TAC o (ONCE_REWRITE_RULE [PROPERTY_STAR])) \\\
396 RES_TAC \\\
397 Q.EXISTS_TAC 'E1' >> ASM_REWRITE_TAC [] \\\
398 REWRITE_TAC [O_DEF] \\\
399 'STRONG_EQUIV (E' P) E1' by PROVE_TAC [STRONG_EQUIV_SYM] \\\
400 'STRONG_EQUIV (E' Q) E2' by PROVE_TAC [STRONG_EQUIV_SYM] \\\
401 Q.EXISTS_TAC 'E' Q' >> ASM_REWRITE_TAC [] \\\
402 Q.EXISTS_TAC 'E' P' >> ASM_REWRITE_TAC [] \\\
403 BETA_TAC >> DISJ2_TAC \\\
404 Q.EXISTS_TAC 'E' >> ASM_REWRITE_TAC [],
405
406 (* goal 3 (of 14) *)
407 Q.EXISTS_TAC 'E1' >> ASM_REWRITE_TAC [] \\\
408 REWRITE_TAC [O_DEF] \\\
409 Q.EXISTS_TAC 'E1' >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\\
410 Q.EXISTS_TAC 'E1' >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\\
411 BETA_TAC >> DISJ1_TAC >> RW_TAC std_ss [],
412 (* goal 4 (of 14) *)
413 Q.EXISTS_TAC 'E2' >> ASM_REWRITE_TAC [] \\\
414 REWRITE_TAC [O_DEF] \\\
415 Q.EXISTS_TAC 'E2' >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\\
416 Q.EXISTS_TAC 'E2' >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL] \\\
417 BETA_TAC >> DISJ1_TAC >> RW_TAC std_ss [],
418
419 (* goal 5 (of 14) *)
420 IMP_RES_TAC TRANS_PREFIX \\\
421 FULL_SIMP_TAC std_ss [] \\\
422 NTAC 2 (POP_ASSUM K_TAC) \\\
423 Q.EXISTS_TAC 'G Q' >> REWRITE_TAC [PREFIX] \\\
424 REWRITE_TAC [O_DEF] >> BETA_TAC \\\
425 Q.EXISTS_TAC 'G Q' >> REWRITE_TAC [STRONG_EQUIV_REFL] \\\
426 Q.EXISTS_TAC 'G P' >> REWRITE_TAC [STRONG_EQUIV_REFL] \\\
427 DISJ2_TAC >> Q.EXISTS_TAC 'G' >> ASM_REWRITE_TAC [],
428 (* goal 6 (of 14) *)
429 IMP_RES_TAC TRANS_PREFIX \\\
430 FULL_SIMP_TAC std_ss [] \\\
431 NTAC 2 (POP_ASSUM K_TAC) \\\
432 Q.EXISTS_TAC 'G P' >> REWRITE_TAC [PREFIX] \\\
433 REWRITE_TAC [O_DEF] >> BETA_TAC \\\
434 Q.EXISTS_TAC 'G Q' >> REWRITE_TAC [STRONG_EQUIV_REFL] \\\
435 Q.EXISTS_TAC 'G P' >> REWRITE_TAC [STRONG_EQUIV_REFL] \\\
436 DISJ2_TAC >> Q.EXISTS_TAC 'G' >> ASM_REWRITE_TAC [],
437

```

```

438 (* goal 7 (of 14) *)
439 IMP_RES_TAC TRANS_SUM >| (* 2 sub-goals here *)
440 [ (* goal 7.1 (of 2) *)
441   RES_TAC \\  

442   Q.EXISTS_TAC 'E2' \\  

443   CONJ_TAC >- ( MATCH_MP_TAC SUM1 >> ASM_REWRITE_TAC [] ) \\  

444   ASM_REWRITE_TAC [],
445   (* goal 7.2 (of 2) *)
446   RES_TAC \\  

447   Q.EXISTS_TAC 'E2' \\  

448   CONJ_TAC >- ( MATCH_MP_TAC SUM2 >> ASM_REWRITE_TAC [] ) \\  

449   ASM_REWRITE_TAC [] ],
450 (* goal 8 (of 14) *)
451 IMP_RES_TAC TRANS_SUM >| (* 2 sub-goals here *)
452 [ (* goal 8.1 (of 2) *)
453   RES_TAC \\  

454   Q.EXISTS_TAC 'E1' \\  

455   CONJ_TAC >- ( MATCH_MP_TAC SUM1 >> ASM_REWRITE_TAC [] ) \\  

456   ASM_REWRITE_TAC [],
457   (* goal 8.2 (of 2) *)
458   RES_TAC \\  

459   Q.EXISTS_TAC 'E1' \\  

460   CONJ_TAC >- ( MATCH_MP_TAC SUM2 >> ASM_REWRITE_TAC [] ) \\  

461   ASM_REWRITE_TAC [] ],
462
463 (* goal 9 (of 14) *)
464 IMP_RES_TAC TRANS_PAR >> FULL_SIMP_TAC std_ss [] >| (* 3 sub-goals here *)
465 [ (* goal 9.1 (of 3) *)
466   Q.PAT_X_ASSUM 'E1 = X' K_TAC \\  

467   RES_TAC \\  

468   Q.EXISTS_TAC 'E2 || G' Q' \\  

469   CONJ_TAC >- ( MATCH_MP_TAC PAR1 >> ASM_REWRITE_TAC [] ) \\  

470   POP_ASSUM MP_TAC \\  

471   REWRITE_TAC [O_DEF] >> BETA_TAC \\  

472   RW_TAC std_ss [] >| (* 2 sub-goals here *)
473   [ (* goal 9.1.1 (of 2) *)
474     Q.EXISTS_TAC 'y || G' Q' \\  

475     REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_PAR \\  

476                           ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

477     Q.EXISTS_TAC 'y || G' P' \\  

478     CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_PAR \\  

479                   ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

480     DISJ2_TAC \\  

481     Q.EXISTS_TAC '\t. y || G' t' >> BETA_TAC >> REWRITE_TAC [] \\  

482     'EXPR (\z. y)' by REWRITE_TAC [EXPR2] \\  

483     Know 'EXPR (\t. (\z. y) t || G' t)'
484     >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\  

485     BETA_TAC >> REWRITE_TAC [],
486     (* goal 9.1.2 (of 2) *)
487     Q.EXISTS_TAC 'G'' Q || G' Q' \\  

488     REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_PAR \\  

489                           ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

490     Q.EXISTS_TAC 'G'' P || G' P' \\  

491     CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_PAR \\  

492                   ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

493     DISJ2_TAC \\  

494     Q.EXISTS_TAC '\t. G'' t || G' t' >> BETA_TAC >> REWRITE_TAC [] \\  

495     MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ],
496   (* goal 9.2 (of 3) *)
497   Q.PAT_X_ASSUM 'E1 = X' K_TAC \\  

498   RES_TAC \\  


```

```

499 Q.EXISTS_TAC 'G Q || E2' \\  

500 CONJ_TAC >- ( MATCH_MP_TAC PAR2 >> ASM_REWRITE_TAC [] ) \\  

501 POP_ASSUM MP_TAC \\  

502 REWRITE_TAC [O_DEF] >> BETA_TAC \\  

503 RW_TAC std_ss [] >| (* 2 sub-goals here *)  

504 [ (* goal 9.2.1 (of 2) *)  

505   Q.EXISTS_TAC 'G Q || y' \\  

506   REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\  

507                         ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

508   Q.EXISTS_TAC 'G P || y' \\  

509   CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\  

510               ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

511   DISJ2_TAC \\  

512   Q.EXISTS_TAC '\t. G t || y' >> BETA_TAC >> REWRITE_TAC [] \\  

513   'EXPR (\z. y)' by REWRITE_TAC [EXPR2] \\  

514   Know 'EXPR (\t. G t || (\z. y) t)'  

515   >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\  

516   BETA_TAC >> REWRITE_TAC [],  

517   (* goal 9.2.2 (of 2) *)  

518   Q.EXISTS_TAC 'G Q || G'' Q' \\  

519   REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\  

520                         ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

521   Q.EXISTS_TAC 'G P || G'' P' \\  

522   CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR \\  

523               ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

524   DISJ2_TAC \\  

525   Q.EXISTS_TAC '\t. G t || G'' t' >> BETA_TAC >> REWRITE_TAC [] \\  

526   MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ],  

527   (* goal 9.3 (of 3) *)  

528   Q.PAT_X_ASSUM 'E1 = X' K_TAC \\  

529   Q.PAT_X_ASSUM 'u = tau' K_TAC \\  

530   RES_TAC \\  

531   Q.EXISTS_TAC 'E2'' || E2'' \\  

532   CONJ_TAC >- ( MATCH_MP_TAC PAR3 >> Q.EXISTS_TAC 'l' >> ASM_REWRITE_TAC [] ) \\  

533   Q.PAT_X_ASSUM 'X E2 E2'' MP_TAC \\  

534   Q.PAT_X_ASSUM 'X E1' E2'' MP_TAC \\  

535   REWRITE_TAC [O_DEF] >> BETA_TAC \\  

536   RW_TAC std_ss [] >| (* 4 sub-goals here *)  

537   [ (* goal 9.3.1 (of 4) *)  

538     Q.EXISTS_TAC 'y || y'' \\  

539     REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC  

540                           Q.EXISTS_TAC 'y || y'' \\  

541                           CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\  

542                           DISJ1_TAC >> REWRITE_TAC [],  

543                           (* goal 9.3.2 (of 4) *)  

544                           Q.EXISTS_TAC 'y || G'' Q' \\  

545                           REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC  

546                           Q.EXISTS_TAC 'y || G'' P' \\  

547                           CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\  

548                           DISJ2_TAC \\  

549                           Q.EXISTS_TAC '\t. y || G'' t' >> BETA_TAC >> REWRITE_TAC [] \\  

550                           'EXPR (\z. y)' by REWRITE_TAC [EXPR2] \\  

551                           Know 'EXPR (\t. (\z. y) t || G'' t)'  

552                           >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\  

553                           BETA_TAC >> REWRITE_TAC [],  

554                           (* goal 9.3.3 (of 4) *)  

555                           Q.EXISTS_TAC 'G'' Q || y'' \\  

556                           REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC  

557                           Q.EXISTS_TAC 'G'' P || y'' \\  

558                           CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESD_BY_PAR >> ASM_REWRITE_TAC [] ) \\  

559                           DISJ2_TAC \\  


```

```

560 Q.EXISTS_TAC '\t. G'' t || y'' >> BETA_TAC >> REWRITE_TAC [] \\  

561 'EXPR (\z. y')' by REWRITE_TAC [EXPR2] \\  

562 Know 'EXPR (\t. G'' t || (\z. y')) t)' \\  

563 >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\  

564 BETA_TAC >> REWRITE_TAC [], \\  

565 (* goal 9.3.4 (of 4) *) \\  

566 Q.EXISTS_TAC 'G'' Q || G'' Q' \\  

567 REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC \\  

568 Q.EXISTS_TAC 'G'' P || G'' P' \\  

569 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC [] ) \\  

570 DISJ2_TAC \\  

571 Q.EXISTS_TAC '\t. G'' t || G'' t' >> BETA_TAC >> REWRITE_TAC [] \\  

572 MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ], \\  

573 (* goal 10 (of 14) *) \\  

574 IMP_RES_TAC TRANS_PAR >> FULL_SIMP_TAC std_ss [] >| (* 3 sub-goals here *) \\  

575 [ (* goal 10.1 (of 3) *) \\  

576 Q.PAT_X_ASSUM 'E2 = X' K_TAC \\  

577 RES_TAC \\  

578 Q.EXISTS_TAC 'E1' || G' P' \\  

579 CONJ_TAC >- ( MATCH_MP_TAC PAR1 >> ASM_REWRITE_TAC [] ) \\  

580 POP_ASSUM MP_TAC \\  

581 REWRITE_TAC [O_DEF] >> BETA_TAC \\  

582 RW_TAC std_ss [] >| (* 2 sub-goals here *) \\  

583 [ (* goal 10.1.1 (of 2) *) \\  

584 Q.EXISTS_TAC 'y || G' Q' \\  

585 REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR \\  

586 ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

587 Q.EXISTS_TAC 'y || G' P' \\  

588 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR \\  

589 ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

590 DISJ2_TAC \\  

591 Q.EXISTS_TAC '\t. y || G' t' >> BETA_TAC >> REWRITE_TAC [] \\  

592 'EXPR (\z. y)' by REWRITE_TAC [EXPR2] \\  

593 Know 'EXPR (\t. (\z. y) t || G' t)' \\  

594 >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\  

595 BETA_TAC >> REWRITE_TAC [], \\  

596 (* goal 10.1.2 (of 2) *) \\  

597 Q.EXISTS_TAC 'G'' Q || G' Q' \\  

598 REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR \\  

599 ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

600 Q.EXISTS_TAC 'G'' P || G' P' \\  

601 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR \\  

602 ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

603 DISJ2_TAC \\  

604 Q.EXISTS_TAC '\t. G'' t || G' t' >> BETA_TAC >> REWRITE_TAC [] \\  

605 MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ], \\  

606 (* goal 10.2 (of 3) *) \\  

607 Q.PAT_X_ASSUM 'E2 = X' K_TAC \\  

608 RES_TAC \\  

609 Q.EXISTS_TAC 'G P || E1'' \\  

610 CONJ_TAC >- ( MATCH_MP_TAC PAR2 >> ASM_REWRITE_TAC [] ) \\  

611 POP_ASSUM MP_TAC \\  

612 REWRITE_TAC [O_DEF] >> BETA_TAC \\  

613 RW_TAC std_ss [] >| (* 2 sub-goals here *) \\  

614 [ (* goal 10.2.1 (of 2) *) \\  

615 Q.EXISTS_TAC 'G Q || y' \\  

616 REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR \\  

617 ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  

618 Q.EXISTS_TAC 'G P || y' \\  

619 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR \\  

620 ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\  


```

```

621     DISJ2_TAC \\\
622     Q.EXISTS_TAC '\t. G t || y' >> BETA_TAC >> REWRITE_TAC [] \\\
623     'EXPR (\z. y)' by REWRITE_TAC [EXPR2] \\\
624     Know 'EXPR (\t. G t || (\z. y) t)'
625     >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\\
626     BETA_TAC >> REWRITE_TAC [],
627     (* goal 10.2.2 (of 2) *)
628     Q.EXISTS_TAC 'G Q || G'' Q' \\\
629     REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR \\\
630                           ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\\
631     Q.EXISTS_TAC 'G P || G'' P' \\\
632     CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR \\\
633                   ASM_REWRITE_TAC [STRONG_EQUIV_REFL] ) \\\
634     DISJ2_TAC \\\
635     Q.EXISTS_TAC '\t. G t || G'' t' >> BETA_TAC >> REWRITE_TAC [] \\\
636     MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ],
637     (* goal 10.3 (of 3) *)
638     Q.PAT_X_ASSUM 'E2 = X' K_TAC \\\
639     Q.PAT_X_ASSUM 'u = tau' K_TAC \\\
640     RES_TAC \\\
641     Q.EXISTS_TAC 'E1'' || E1'' \\\
642     CONJ_TAC >- ( MATCH_MP_TAC PAR3 >> Q.EXISTS_TAC 'l' >> ASM_REWRITE_TAC [] ) \\\
643     Q.PAT_X_ASSUM 'X E1'' E1' MP_TAC \\\
644     Q.PAT_X_ASSUM 'X E1' E2'' MP_TAC \\\
645     REWRITE_TAC [O_DEF] >> BETA_TAC \\\
646     RW_TAC std_ss [] >| (* 4 sub-goals here *)
647     [ (* goal 10.3.1 (of 4) *)
648       Q.EXISTS_TAC 'y'' || y' \\\
649       REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC
650                             Q.EXISTS_TAC 'y'' || y' \\\
651                             CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC [] ) \\\
652                             DISJ1_TAC >> REWRITE_TAC [],
653                             (* goal 10.3.2 (of 4) *)
654                             Q.EXISTS_TAC 'G'' Q || y' \\\
655                             REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC
656                             Q.EXISTS_TAC 'G'' P || y' \\\
657                             CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC [] ) \\\
658                             DISJ2_TAC \\\
659                             Q.EXISTS_TAC '\t. G'' t || y' >> BETA_TAC >> REWRITE_TAC [] \\\
660                             'EXPR (\z. y)' by REWRITE_TAC [EXPR2] \\\
661                             Know 'EXPR (\t. G'' t || (\z. y) t)'
662                             >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\\
663                             BETA_TAC >> REWRITE_TAC [],
664                             (* goal 10.3.3 (of 4) *)
665                             Q.EXISTS_TAC 'y'' || G'' Q' \\\
666                             REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC
667                             Q.EXISTS_TAC 'y'' || G'' P' \\\
668                             CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC [] ) \\\
669                             DISJ2_TAC \\\
670                             Q.EXISTS_TAC '\t. y'' || G'' t' >> BETA_TAC >> REWRITE_TAC [] \\\
671                             'EXPR (\z. y'')' by REWRITE_TAC [EXPR2] \\\
672                             Know 'EXPR (\t. (\z. y'') t || G'' t)'
673                             >- ( MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ) \\\
674                             BETA_TAC >> REWRITE_TAC [],
675                             (* goal 10.3.4 (of 4) *)
676                             Q.EXISTS_TAC 'G''' Q || G'' Q' \\\
677                             REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC
678                             Q.EXISTS_TAC 'G''' P || G'' P' \\\
679                             CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_PRESBY_BY_PAR >> ASM_REWRITE_TAC [] ) \\\
680                             DISJ2_TAC \\\
681                             Q.EXISTS_TAC '\t. G''' t || G'' t' >> BETA_TAC >> REWRITE_TAC [] \\\

```

```

682 MATCH_MP_TAC EXPR5 >> ASM_REWRITE_TAC [] ] ],
683
684 (* goal 11 (of 14) *)
685 IMP_RES_TAC TRANS_RESTR >> FULL_SIMP_TAC std_ss [] >| (* 2 sub-goals here *)
686 [ (* goal 11.1 (of 2) *)
687   RES_TAC \\  

688   Q.EXISTS_TAC 'restr L E2' \\  

689   CONJ_TAC >- ( MATCH_MP_TAC RESTR >> FULL_SIMP_TAC std_ss [] ) \\  

690   POP_ASSUM MP_TAC \\  

691   REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)
692   [ (* goal 11.1.1 (of 2) *)
693     Q.EXISTS_TAC 'restr L y' \\  

694     REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
695     Q.EXISTS_TAC 'restr L y' \\  

696     CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

697     DISJ1_TAC >> REWRITE_TAC [],
698     (* goal 11.1.2 (of 2) *)
699     Q.EXISTS_TAC 'restr L (G' Q)' \\  

700     REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
701     Q.EXISTS_TAC 'restr L (G' P)' \\  

702     CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

703     DISJ2_TAC \\  

704     Q.EXISTS_TAC '\t. restr L (G' t)' >> BETA_TAC >> REWRITE_TAC [] \\  

705     MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ],
706   (* goal 11.2 (of 2) *)
707   RES_TAC \\  

708   Q.EXISTS_TAC 'restr L E2' \\  

709   CONJ_TAC >- ( MATCH_MP_TAC RESTR >> Q.EXISTS_TAC '1' >> ASM_REWRITE_TAC [] ) \\  

710   POP_ASSUM MP_TAC \\  

711   REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)
712   [ (* goal 11.2.1 (of 2) *)
713     Q.EXISTS_TAC 'restr L y' \\  

714     REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
715     Q.EXISTS_TAC 'restr L y' \\  

716     CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

717     DISJ1_TAC >> REWRITE_TAC [],
718     (* goal 11.2.2 (of 2) *)
719     Q.EXISTS_TAC 'restr L (G' Q)' \\  

720     REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
721     Q.EXISTS_TAC 'restr L (G' P)' \\  

722     CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

723     DISJ2_TAC \\  

724     Q.EXISTS_TAC '\t. restr L (G' t)' >> BETA_TAC >> REWRITE_TAC [] \\  

725     MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ] ],
726   (* goal 12 (of 14) *)
727   IMP_RES_TAC TRANS_RESTR >> FULL_SIMP_TAC std_ss [] >| (* 2 sub-goals here *)
728   [ (* goal 12.1 (of 2) *)
729     RES_TAC \\  

730     Q.EXISTS_TAC 'restr L E1' \\  

731     CONJ_TAC >- ( MATCH_MP_TAC RESTR >> FULL_SIMP_TAC std_ss [] ) \\  

732     POP_ASSUM MP_TAC \\  

733     REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)
734     [ (* goal 12.1.1 (of 2) *)
735       Q.EXISTS_TAC 'restr L y' \\  

736       REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [
737       Q.EXISTS_TAC 'restr L y' \\  

738       CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

739       DISJ1_TAC >> REWRITE_TAC [],
740       (* goal 12.1.2 (of 2) *)
741       Q.EXISTS_TAC 'restr L (G' Q)' \\  

742       REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [

```



```

743 Q.EXISTS_TAC 'restr L (G' P)' \\  

744 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

745 DISJ2_TAC \\  

746 Q.EXISTS_TAC '\t. restr L (G' t)' >> BETA_TAC >> REWRITE_TAC [] \\  

747 MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ],  

748 (* goal 12.2 (of 2) *)  

749 RES_TAC \\  

750 Q.EXISTS_TAC 'restr L E1' \\  

751 CONJ_TAC >- ( MATCH_MP_TAC RESTR >> Q.EXISTS_TAC '1' >> ASM_REWRITE_TAC [] ) \\  

752 POP_ASSUM MP_TAC \\  

753 REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)  

754 [ (* goal 12.2.1 (of 2) *)  

755 Q.EXISTS_TAC 'restr L y' \\  

756 REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

757 Q.EXISTS_TAC 'restr L y' \\  

758 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

759 DISJ1_TAC >> REWRITE_TAC [],  

760 (* goal 12.2.2 (of 2) *)  

761 Q.EXISTS_TAC 'restr L (G' Q)' \\  

762 REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

763 Q.EXISTS_TAC 'restr L (G' P)' \\  

764 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RESTR >> ASM_REWRITE_TAC [] ) \\  

765 DISJ2_TAC \\  

766 Q.EXISTS_TAC '\t. restr L (G' t)' >> BETA_TAC >> REWRITE_TAC [] \\  

767 MATCH_MP_TAC EXPR6 >> ASM_REWRITE_TAC [] ] ],  

768  

769 (* goal 13 (of 14) *)  

770 IMP_RES_TAC TRANS_RELAB \\  

771 FULL_SIMP_TAC std_ss [] \\  

772 RES_TAC \\  

773 Q.EXISTS_TAC 'restr E2 rf' \\  

774 CONJ_TAC >- ( MATCH_MP_TAC RELABELING >> ASM_REWRITE_TAC [] ) \\  

775 POP_ASSUM MP_TAC \\  

776 REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)  

777 [ (* goal 13.1 (of 2) *)  

778 Q.EXISTS_TAC 'restr y rf' \\  

779 REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\  

780 Q.EXISTS_TAC 'restr y rf' \\  

781 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\  

782 DISJ1_TAC >> REWRITE_TAC [],  

783 (* goal 13.2 (of 2) *)  

784 Q.EXISTS_TAC 'restr (G' Q) rf' \\  

785 REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\  

786 Q.EXISTS_TAC 'restr (G' P) rf' \\  

787 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\  

788 DISJ2_TAC \\  

789 Q.EXISTS_TAC '\t. restr (G' t) rf' >> BETA_TAC >> REWRITE_TAC [] \\  

790 MATCH_MP_TAC EXPR7 >> ASM_REWRITE_TAC [] ],  

791 (* goal 14 (of 14) *)  

792 IMP_RES_TAC TRANS_RELAB \\  

793 FULL_SIMP_TAC std_ss [] \\  

794 RES_TAC \\  

795 Q.EXISTS_TAC 'restr E1 rf' \\  

796 CONJ_TAC >- ( MATCH_MP_TAC RELABELING >> ASM_REWRITE_TAC [] ) \\  

797 POP_ASSUM MP_TAC \\  

798 REWRITE_TAC [O_DEF] >> BETA_TAC >> RW_TAC std_ss [] >| (* 2 sub-goals here *)  

799 [ (* goal 14.1 (of 2) *)  

800 Q.EXISTS_TAC 'restr y rf' \\  

801 REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\  

802 Q.EXISTS_TAC 'restr y rf' \\  

803 CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \

```

```

804     DISJ1_TAC >> REWRITE_TAC [],
805     (* goal 14.2 (of 2) *)
806     Q.EXISTS_TAC 'relab (G' Q) rf' \\
807     REVERSE CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC []
808     Q.EXISTS_TAC 'relab (G' P) rf' \\
809     CONJ_TAC >- ( MATCH_MP_TAC STRONG_EQUIV_SUBST_RELAB >> ASM_REWRITE_TAC [] ) \\
810     DISJ2_TAC \\
811     Q.EXISTS_TAC '\t. relab (G' t) rf' >> BETA_TAC >> REWRITE_TAC [] \\
812     MATCH_MP_TAC EXPR7 >> ASM_REWRITE_TAC [] ] ] );
813
814 (** Bibliography:
815  *
816  * Milner, R.: Communication and concurrency. (1989).
817  * Sangiorgi, D.: Equations, contractions, and unique solutions. ACM SIGPLAN Notices. (201
818  *)
819
820 val _ = export_theory ();
821 val _ = DB.html_theory "UniqueSolutions";
822
823 (* last updated: Aug 3, 2017 *)

```