Formalized bisimulation-up-to

Chun Tian

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

Abstract. Basic results for "bisimulation up to $\sim$ and $\approx$" are formalized in this work. The work is a necessary path towards Milner's "unique solutions of equations" theorems.

1 Introduction

This is another prelimitary thesis work, after [1], in which basic results for "bisimulation up to $\sim$" were already proved. Both this short paper and [1] are based on the author's previous two projects [2] [3], which are finally based on the work of Monica Nesi [4] during 1992-1995.

Here we first added a few more common results for the strong bisimulation cases, then turned to the (weak) bisimulation up to $\approx$ and have proved all needed results towards Milner's "unique solutions of equations" theorems.

It turned out that, the proof size for the weak bisimulation case, although essentially the same as in the strong bisimulation case, is almost 5 times longer, and we have to prove a large amount of lemmas to support the "short" proof of the final lemma and theorem.

2 Bisimulation up to $\sim$

In previous paper [1], we have shown our definition of "strong bisimulation upto $\sim$":

Definition 1. *(Strong bisimulation up to $\sim$)*

$$
\begin{array}{l}
\vdash \texttt{STRONG\_BISIM\_UPTO}\ Bsm \iff \\
\quad \forall E\ E'. \\
\qquad Bsm\ E\ E' \Rightarrow \\
\qquad \forall u. \\
\qquad\quad (\forall E_1. \\
\qquad\qquad E\ \texttt{--}u\texttt{->}\ E_1 \Rightarrow \\
\qquad\qquad \exists E_2. \\
\qquad\qquad\quad E'\ \texttt{--}u\texttt{->}\ E_2\ \wedge \\
\qquad\qquad\quad (\texttt{STRONG\_EQUIV} \circ_r Bsm \circ_r \texttt{STRONG\_EQUIV})\ E_1\ E_2) \wedge \\
\qquad\quad \forall E_2. \\
\qquad\qquad E'\ \texttt{--}u\texttt{->}\ E_2 \Rightarrow \\
\qquad\qquad \exists E_1. \\
\qquad\qquad\quad E\ \texttt{--}u\texttt{->}\ E_1\ \wedge \\
\qquad\qquad\quad (\texttt{STRONG\_EQUIV} \circ_r Bsm \circ_r \texttt{STRONG\_EQUIV})\ E_1\ E_2
\end{array}
$$

New additions for this part, are the following common properties:

Proposition 1. *Properties of "strong bisimulation upto $\sim$"*

1. *Identity relation is a "strong bisimulation upto $\sim$":*

   $\vdash \texttt{STRONG\_BISIM\_UPTO}\ (\lambda x\ y.\ x = y)$

2. *The converse of a "strong bisimulation upto $\sim$" is still "strong bisimulation upto $\sim$":*

   $\vdash \texttt{STRONG\_BISIM\_UPTO}\ Wbsm \Rightarrow \texttt{STRONG\_BISIM\_UPTO}\ (\lambda x\ y.\ Wbsm\ y\ x)$

And we have proved the following lemma, which 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:*

$\vdash$ `STRONG_BISIM_UPTO` $Bsm \Rightarrow$
`STRONG_BISIM (STRONG_EQUIV` $\circ_r$ Bsm $\circ_r$ `STRONG_EQUIV)`

Proof. The idea is to fix two process E and E', which satisfies $E \sim \circ Bsm \circ E'$, then check it for the definition of strong bisimulation: for all E_1 such that E `-`u`->` E_1, there exists E_2'' such that E' `-`u`->` E_2'' (the other side is totally symmetric), as shown in the following graph:

$$\begin{tikzcd}
E \arrow[rr, dash, "\sim"] \arrow[d, "\forall u"'] & & \exists y' \arrow[r, dash, "Bsm"] \arrow[dl, "u"] & \exists y \arrow[rr, dash, "\sim"] \arrow[dr, "u"] & & E' \arrow[d, "u"] \\
\forall E_1 \arrow[r, dash, "\sim"] & \exists E_2 \arrow[r, dash, "\sim"] & \exists y''' \arrow[r, dash, "Bsm"] & \exists y'' \arrow[r, dash, "\sim"] & \exists E_2' \arrow[r, dash, "\sim"] & \exists E_2''
\end{tikzcd}$$

During the proof, needed lemmas are the definition of "bisimulation up to $\sim$" (for expanding "y' Bsm y" into "$E_2 \sim y'''$ Bsm $y'' \sim E_2'$"), plus the property (*) and transitivity of strong equivalence.

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

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

$\vdash$ `STRONG_BISIM_UPTO` $Bsm \Rightarrow Bsm \subseteq_r$ `STRONG_EQUIV`

Hence, to prove $P \sim Q$, we only have to find a strong bisimulation up to $\sim$ which contains (P, Q).

3 Bisimulation up to $\approx$

The concept of bisimulation up to $\approx$ is defined in the following way:

Definition 2. *(Bisimulation up to $\approx$)*

$\vdash$ `WEAK_BISIM_UPTO` $Wbsm \iff$
  $\forall\, E\ E'.$
    $Wbsm\ E\ E' \Rightarrow$
    $(\forall\, l.$
      $(\forall\, E_1.$
        E `--label` l`->` $E_1 \Rightarrow$
        $\exists\, E_2.$
          E' `==label` l`=>>` $E_2\ \wedge$
          `(WEAK_EQUIV` $\circ_r$ $Wbsm$ $\circ_r$ `STRONG_EQUIV)` $E_1\ E_2) \wedge$
      $\forall\, E_2.$
        E' `--label` l`->` $E_2 \Rightarrow$
        $\exists\, E_1.$
          E `==label` l`=>>` $E_1\ \wedge$
          `(STRONG_EQUIV` $\circ_r$ $Wbsm$ $\circ_r$ `WEAK_EQUIV)` $E_1\ E_2) \wedge$
    $(\forall\, E_1.$
      E `--`τ`->` $E_1 \Rightarrow$
      $\exists\, E_2.$
        $E' \stackrel{\epsilon}{\Rightarrow} E_2\ \wedge$ `(WEAK_EQUIV` $\circ_r$ $Wbsm$ $\circ_r$ `STRONG_EQUIV)` $E_1\ E_2) \wedge$
    $\forall\, E_2.$
      E' `--`τ`->` $E_2 \Rightarrow$
      $\exists\, E_1.\ E \stackrel{\epsilon}{\Rightarrow} E_1\ \wedge$ `(STRONG_EQUIV` $\circ_r$ $Wbsm$ $\circ_r$ `WEAK_EQUIV)` $E_1\ E_2$

However, there're a few things to be noticed:

1. In HOL4, the big "O" notion as relation composition has different orders with usual Math notion: *the right-most relation takes the input argument first*, which is actually the case for function composition: $(f \circ g)(x) = f(g(x))$. Thus in all HOL-generted terms like

   `(WEAK_EQUIV` $\circ_r$ *Wbsm* $\circ_r$ `STRONG_EQUIV)` E_1 E_2

   in this paper, it should be understood like "$E_1 \sim y$ Wbsm $y' \approx E_2$". (There was no such issues for the strong bisimulation cases, because we had $\sim$ on both side)
2. The original definition in Milner's book [5], in which he used $\approx \circ R \circ \approx$ in all places in above definition, is wrong. The reason has been explained in Gorrieri's book [6] (page 65), that the resulting relation may not be a subset of $\approx$! Thus we have used the definition from Gorrieri's book, with the definition in Sangiorgi's book [7] (page 115) doubly confirmed.
3. Some authors (e.g. Prof. Sangiorgi) uses the notions like $P \overset{\hat{\mu}}{\Rightarrow} P'$ to represent special case that, $P \overset{\epsilon}{\Rightarrow} P'$ when $\mu = \tau$ (i.e. it's possible that $P = Q$). Such notions are concise, but inconvenient to use in formalization work, because the EPS transition, in many cases, has very different characteristics. Thus we have above long definition but easier to use when proving all needed results.

Two basic properties to help understanding "bisimulation up to $\approx$":

Lemma 2. *(Properties of bisimulation up to $\approx$)*

1. *The identity relation is "bisimulation up to $\approx$":*

   $\vdash$ `WEAK_BISIM_UPTO` $(\lambda x\ y.\ x = y)$
2. *The converse of a "bisimulation up to $\approx$" is still "bisimulation up to $\approx$":*

   $\vdash$ `WEAK_BISIM_UPTO` *Wbsm* $\Rightarrow$ `WEAK_BISIM_UPTO` $(\lambda x\ y.\ Wbsm\ y\ x)$

Now we want to prove the following main lemma:

Lemma 3. *If $\mathcal{S}$ is a bisimulation up to $\approx$, then $\approx \mathcal{S} \approx$ is a bisimulation.*

$\vdash$ `WEAK_BISIM_UPTO` *Wbsm* $\Rightarrow$
`WEAK_BISIM (WEAK_EQUIV` $\circ_r$ *Wbsm* $\circ_r$ `WEAK_EQUIV)`

Proof. Milner's books simply said that the proof is "analogous" to the same lemma for strong bisimulation. This is basically true, from left to right (for visible transitions):

$$\begin{tikzcd}
E \arrow[rrr, dash, "\approx"] \arrow[d, "\forall l"'] & & & \exists y' \arrow[r, dash, "Wbsm"] \arrow[dll, Rightarrow, "l"] & \exists y \arrow[rr, dash, "\approx"] \arrow[dr, Rightarrow, "l"] & & E' \arrow[d, Rightarrow, "l"] \\
\forall E_1 \arrow[r, dash, "\approx"] & \exists E_2 \arrow[r, dash, "\sim"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[r, dash, "\approx"] & \exists E_2' \arrow[r, dash, "\approx"] & \exists E_2''
\end{tikzcd}$$

There's a little difficulty, however. Given $y \approx E'$ and $y \overset{l}{\Rightarrow} E_2'$, the existence of E_2'' doesn't follow directly from the definition or property (*) of weak equivalence. Instead, we have to prove a lemma (to be presented below) to finish this last step.

More difficulities appear from right to left:

$$\begin{tikzcd}
E \arrow[rrr, dash, "\approx"] \arrow[d, Rightarrow, "l"'] & & & \exists y' \arrow[r, dash, "Wbsm"] \arrow[dll, Rightarrow, "l"] & \exists y \arrow[rr, dash, "\approx"] \arrow[dr, Rightarrow, "l"] & & E' \arrow[d, "\forall l"] \\
\exists E_1'' \arrow[r, dash, "\approx"] & \exists E_1' \arrow[r, dash, "\approx"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[r, dash, "\sim"] & \exists E_1 \arrow[r, dash, "\approx"] & \forall E_2
\end{tikzcd}$$

The problem is, given y' Bsm y and $y \overset{l}{\Rightarrow} E_1$, the existence of E_1' doesn't follow directly from the definition of "bisimulation up to $\approx$", instead this result must be proved (to be presented below) and the proof is non-trivial.

The other two cases concerning τ-transitions:

$$\begin{tikzcd}
E \arrow[rr, dash, "\approx"] \arrow[d, "\tau"] & & \exists y' \arrow[rr, dash, "Wbsm"] \arrow[dl, Rightarrow, "\epsilon"] & & \exists y \arrow[rr, dash, "\approx"] \arrow[dr, Rightarrow, "\epsilon"] & & E' \arrow[d, Rightarrow, "\epsilon"] \\
\forall E_1 \arrow[r, dash, "\approx"] & \exists E_2 \arrow[r, dash, "\sim"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[rr, dash, "\approx"] & & \exists E_2' \arrow[r, dash, "\approx"] & \exists E_2''
\end{tikzcd}$$

in which the EPS transition bypass of weak equivalence (from E_2' to E_2'') must be proved, and

$$\begin{tikzcd}
E \arrow[rr, dash, "\approx"] \arrow[d, Rightarrow, "\epsilon"] & & \exists y' \arrow[rr, dash, "Wbsm"] \arrow[dl, Rightarrow, "\epsilon"] & & \exists y \arrow[rr, dash, "\approx"] \arrow[dr, Rightarrow, "\epsilon"] & & E' \arrow[d, "\tau"] \\
\exists E_1'' \arrow[r, dash, "\approx"] & \exists E_1' \arrow[r, dash, "\approx"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[rr, dash, "\sim"] & & \exists E_1 \arrow[r, dash, "\approx"] & \forall E_2
\end{tikzcd}$$

in which the EPS transition bypass for "bisimulation up to $\approx$" (from E_1 to E_1') must be proved as a lemma.

As a summary of all "difficulities", here is a list of lemmas we have used to prove the previous lemma. Each lemma has also their "companion lemma" concerning the other directions (here we omit them):

Lemma 4. *(Useful lemmas concerning the first weak transitions from $\sim$, $\approx$ and "bisimulation up to $\approx$")*

STRONG_EQUIV_EPS:
$\vdash E \sim E' \Rightarrow \forall E_1.\ E \stackrel{\epsilon}{\Rightarrow} E_1 \Rightarrow \exists E_2.\ E' \stackrel{\epsilon}{\Rightarrow} E_2 \wedge E_1 \sim E_2$
WEAK_EQUIV_EPS:
$\vdash E \approx E' \Rightarrow \forall E_1.\ E \stackrel{\epsilon}{\Rightarrow} E_1 \Rightarrow \exists E_2.\ E' \stackrel{\epsilon}{\Rightarrow} E_2 \wedge E_1 \approx E_2$
WEAK_EQUIV_WEAK_TRANS_label:
$\vdash E \approx E' \Rightarrow$
  $\forall l\ E_1.\ E$ ==label l=>> $E_1 \Rightarrow \exists E_2.\ E'$ ==label l=>> $E_2 \wedge E_1 \approx E_2$
WEAK_EQUIV_WEAK_TRANS_tau:
$\vdash E \approx E' \Rightarrow \forall E_1.\ E$ ==τ=>> $E_1 \Rightarrow \exists E_2.\ E' \stackrel{\epsilon}{\Rightarrow} E_2 \wedge E_1 \approx E_2$
WEAK_BISIM_UPTO_EPS:
$\vdash$ WEAK_BISIM_UPTO $Wbsm \Rightarrow$
  $\forall E\ E'.$
    $Wbsm\ E\ E' \Rightarrow$
    $\forall E_1.$
      $E \stackrel{\epsilon}{\Rightarrow} E_1 \Rightarrow$
      $\exists E_2.\ E' \stackrel{\epsilon}{\Rightarrow} E_2 \wedge$ (WEAK_EQUIV $\circ_r$ $Wbsm$ $\circ_r$ STRONG_EQUIV) $E_1\ E_2$
WEAK_BISIM_UPTO_WEAK_TRANS_label:
$\vdash$ WEAK_BISIM_UPTO $Wbsm \Rightarrow$
  $\forall E\ E'.$
    $Wbsm\ E\ E' \Rightarrow$
    $\forall l\ E_1.$
      E ==label l=>> $E_1 \Rightarrow$
      $\exists E_2.$
        E' ==label l=>> $E_2 \wedge$
        (WEAK_EQUIV $\circ_r$ $Wbsm$ $\circ_r$ STRONG_EQUIV) $E_1\ E_2$

Proof. (Proof sketch of above lemmas) The proof of STORNG_EQUIV_EPS and WEAK_EQUIV_EPS depends on the following "right induction theorem"[1] of the EPS transition:

[1] Such induction theorems are part of HOL's theorem library for all RTCs (reflexive transitive closure). The proof of transitivity of observation congruence also heavily depends on this induction theorem, but in the work of Monica Nesi where the EPS relation is manually defined inductively, such an induction theorem is not available (and it's not easy to prove it), as a result Monica Nesi couldn't finish the proof for transitivity of observation congruence, which is incredible hard to prove without proving lemmas like WEAK_EQUIV_EPS first.

```
EPS_ind_right:
```
$\vdash$ `(∀x. P x x) ∧ (∀x y z. P x y ∧ y --τ-> z ⇒ P x z) ⇒`
  `∀x y. x` $\overset{\epsilon}{\Rightarrow}$ `y ⇒ P x y`

Basically, we need to prove that, if the lemma already holds for $n-1$ τ-transitions, it also holds for n τ-transitions.

The proof of `WEAK_BISIM_UPTO_EPS` is also based on above induction theorem. The induction case of this proof can be sketched using the following graph:

$$\begin{tikzcd}
 & E \arrow[rrr, dash, "Wbsm"] \arrow[d, dash, dotted] & & & E' \arrow[d, Rightarrow, "\epsilon"] & \\
 & E_1 \arrow[r, dash, "\sim"] \arrow[dl, "\tau"] & \exists y' \arrow[r, dash, "Wbsm"] \arrow[dl, "\tau"] & \exists y \arrow[r, dash, "\approx"] \arrow[dr, Rightarrow, "\epsilon"] & E_2 \arrow[dr, Rightarrow, "\epsilon"] & \\
\forall E_1' \arrow[r, dash, "\approx"] & \exists E_2' \arrow[r, dash, "\sim"] & \exists y''' \arrow[r, dash, "Wbsm"] & \exists y'' \arrow[r, dash, "\approx"] & \exists E_2'' \arrow[r, dash, "\approx"] & \exists E_2'''
\end{tikzcd}$$

The goal is to find E_2''' which satisfy $E' \overset{\epsilon}{\Rightarrow} E2'''$. As we can see from the graph, using the induction, now given y' Wbsm y and $y' \overset{\tau}{\rightarrow} E_2'$, we can easily crossover the "bisimulation up to $\approx$" and assert the existence of E_2'' to finish the proof.

The proof of `WEAK_BISIM_UPTO_WEAK_TRANS_label` is based on `WEAK_BISIM_UPTO_EPS`. It is much more difficult, because an even bigger graph must be step-by-step constructed:

$$\begin{tikzcd}
 & E \arrow[rrr, dash, "Wbsm"] \arrow[d, Rightarrow, "\epsilon"] & & & E' \arrow[d, Rightarrow, "\epsilon"] & \\
 & \exists E_1' \arrow[r, dash, "\sim"] \arrow[dl, "l"] & \exists y' \arrow[r, dash, "Wbsm"] \arrow[dl, "l"] & \exists y \arrow[r, dash, "\approx"] \arrow[dr, Rightarrow, "l"] & E_2' \arrow[dr, Rightarrow, "l"] & \\
\exists E_2 \arrow[r, dash, "\sim"] \arrow[dr, dash, dotted] & \exists E_2'' \arrow[rrr, dash, "\sim \text{ Wbsm } \approx"] \arrow[dr, dash, dotted] & & & \exists E_2''' \arrow[r, dash, "\approx"] \arrow[dl, dash, dotted] & \exists E_2^{(4)} \arrow[dl, dash, dotted] \\
 & \exists E_2 \arrow[r, dash, "\sim"] \arrow[dl, Rightarrow, "\epsilon"] & \exists y''' \arrow[r, dash, "Wbsm"] \arrow[dl, Rightarrow, "\epsilon"] & \exists y'' \arrow[r, dash, "\approx"] \arrow[dr, Rightarrow, "\epsilon"] & \exists E_2^{(4)} \arrow[dr, Rightarrow, "\epsilon"] & \\
\forall E_1 \arrow[r, dash, "\sim"] & \exists E_2^{(5)} \arrow[rrr, dash, "\sim \text{ Wbsm } \approx"] & & & \exists E_2^{(6)} \arrow[r, dash, "\approx"] & \exists E_2^{(7)}
\end{tikzcd}$$

That is, for all E_1 such that $E \overset{l}{\Rightarrow} E_1$ (which by definition of weak transitions exists E_1' and E_2 such that $E \overset{\epsilon}{\Rightarrow} E_1'$, $E_1' \overset{l}{\rightarrow} E_2$ and $E_2 \overset{\epsilon}{\Rightarrow} E_1$), we would like to finally find an $E_2^{(7)}$ such that $E' \overset{l}{\Rightarrow} E_2^{(7)}$. This process is long and painful, and we have to use `WEAK_BISIM_UPTO_EPS` twice. The formal proof tries to build above graph by asserting the existences of each process step-by-step, until it finally reached to $E_2^{(}7)$. This proof is so-far the largest formal proof (in single branch) that the author ever met, before closing it has 26 assumptions which represents above graph:

```
?E2. E' ==label l=>> E2 /\ (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E2
------------------------------------
  0.  WEAK_BISIM_UPTO Wbsm
  1.  Wbsm E E'
  2.  E ==label l=>> E1
  3.  EPS E E1'
  4.  E1' --label l-> E2
  5.  EPS E2 E1
  6.  EPS E' E2'
  7.  STRONG_EQUIV E1' y'
```

```
8.  Wbsm y' y
9.  WEAK_EQUIV y E2'
10.  y' --label l-> E2''
11.  STRONG_EQUIV E2 E2''
12.  y ==label l=>> E2'''
13.  (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E2'' E2'''
14.  E2' ==label l=>> E2''''
15.  WEAK_EQUIV E2''' E2''''
16.  STRONG_EQUIV E2 y'''
17.  Wbsm y''' y''
18.  WEAK_EQUIV y'' E2''''
19.  EPS y''' E2'''''
20.  STRONG_EQUIV E1 E2'''''
21.  EPS y'' E2''''''
22.  (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E2''''' E2''''''
23.  EPS E2'''' E2'''''''
24.  WEAK_EQUIV E2'''''' E2'''''''
25.  (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E2'''''''
```

All above lemmas concern the cases from left and right (for all P, exists Q such that ...) To prove the other side (for all Q there exist P such that ...), there's no need to go over the painful proving process again, instead we can easily derive the other side by using `CONVERSE_WEAK_BISIM_UPTO`. For example, once above lemma `WEAK_BISIM_UPTO_WEAK_TRANS_label` is proved, it's trivial to get the following companion lemma:

Lemma 5. `WEAK_BISIM_UPTO_WEAK_TRANS_label'`:

$\vdash$ `WEAK_BISIM_UPTO` $\mathit{Wbsm} \Rightarrow$
  $\forall E\ E'.$
    $\mathit{Wbsm}\ E\ E' \Rightarrow$
    $\forall l\ E_2.$
      E' `==label` l`=>>` $E_2 \Rightarrow$
      $\exists E_1.$
        E `==label` l`=>>` $E_1 \wedge$
        (`STRONG_EQUIV` $\circ_r$ Wbsm $\circ_r$ `WEAK_EQUIV`) $E_1\ E_2$

Finally, once the main lemma `WEAK_BISIM_UPTO_LEMMA`, the following final result can be easily proved, following the same idea in the proof of strong bisimulation cases:

Theorem 1. *WEAK_BISIM_UPTO_THM:*

$\vdash$ `WEAK_BISIM_UPTO` $\mathit{Wbsm} \Rightarrow \mathit{Wbsm} \subseteq_r$ `WEAK_EQUIV`

4 Conclusions

So far we have proved all needed results for "bisimulation up to $\approx$", now we're ready to proceed Milner's "unique solutions of equations" theorem for weak bisimulation cases. But the work mentioned in this paper is also the basis to prove the futher "unique solutions" theorem in Prof. Sangiorgi's paper [8].

To the best of our knowledge, beside the work of Monica Nesi in Hol88 and the author's porting work to HOL4 with extensions upto this paper, no other formalization ever reached the "bisimulation up to" concepts. Beside our work, currently the most comprehensive (publicly available) formalization of CCS is the formalization found in Isabelle/AFP [2] based on so-called "nominal logic", done by Jesper Bengtson as part of his thesis (from the length (498 pages) of that thesis paper [3], it must be a ph.D thesis.), but "bisimulation up to" are not part of the formalization. That student tried to cover not only CCS but also π-calculus and the so-called ψ-calculi (don't know what it is), it seems that the goal is too big and as a result, none is really touching any deep

[2] `https://www.isa-afp.org/entries/CCS.html`
[3] `http://www.itu.dk/people/jebe/files/thesis.pdf`

result in these formal systems. The author, instead, would like to focus on CCS (at least before his graduation) and tries to touch the frontier through multiple projects on CCS formalization, with each one based on previous parts.

References

1. Tian, C.: Unique solutions of equations for strong equivalence. (August 2017) 1–18
2. Tian, C.: A Formalization of the Process Algebra CCS in HOL4. arXiv.org, `http://arxiv.org/abs/1705.07313v2` (May 2017)
3. Tian, C.: Further Formalization of the Process Algebra CCS in HOL4. arXiv.org, `http://arxiv.org/abs/1707.04894v2` (July 2017)
4. Nesi, M.: A formalization of the process algebra CCS in high order logic. Technical report, University of Cambridge, Computer Laboratory (1992)
5. Milner, R.: Communication and concurrency. Prentice Hall (1989)
6. Gorrieri, R., Versari, C.: Introduction to Concurrency Theory. Transition Systems and CCS. Springer, Cham (September 2015)
7. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press (October 2011)
8. Sangiorgi, D.: Equations, contractions, and unique solutions. ACM SIGPLAN Notices (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;

val _ = new_theory "BisimulationUpto";

(******************************************************************************)
(*
*)
(*                        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 IDENTITY_STRONG_BISIM_UPTO_lemma = store_thm (
   "IDENTITY_STRONG_BISIM_UPTO_lemma",
  ``!E. (STRONG_EQUIV O (\x y. x = y) O STRONG_EQUIV) E E``,
    GEN_TAC >> REWRITE_TAC [O_DEF] >> BETA_TAC
 >> NTAC 2 (Q.EXISTS_TAC `E` >> REWRITE_TAC [STRONG_EQUIV_REFL]));

val IDENTITY_STRONG_BISIM_UPTO = store_thm (
   "IDENTITY_STRONG_BISIM_UPTO", ``STRONG_BISIM_UPTO (\x y. x = y)``,
    PURE_ONCE_REWRITE_TAC [STRONG_BISIM_UPTO]
 >> BETA_TAC
 >> REPEAT STRIP_TAC (* 2 sub-goals *)
 >| [ (* goal 1 *)
      ASSUME_TAC (REWRITE_RULE [ASSUME ``E:('a, 'b) CCS = E'``]
                               (ASSUME ``TRANS E u E1``)) \\
      EXISTS_TAC ``E1 :('a, 'b) CCS`` \\
      ASM_REWRITE_TAC [] \\
      REWRITE_TAC [IDENTITY_STRONG_BISIM_UPTO_lemma],
      (* goal 2 *)
      PURE_ONCE_ASM_REWRITE_TAC [] \\
      EXISTS_TAC ``E2 :('a, 'b) CCS`` \\
      ASM_REWRITE_TAC [] \\
      REWRITE_TAC [IDENTITY_STRONG_BISIM_UPTO_lemma] ]);

val CONVERSE_STRONG_BISIM_UPTO_lemma = Q.prove (
   `!Wbsm E E'. (STRONG_EQUIV O (\x y. Wbsm y x) O STRONG_EQUIV) E E' =
                (STRONG_EQUIV O Wbsm O STRONG_EQUIV) E' E`,
    rpt GEN_TAC
 >> EQ_TAC (* 2 sub-goals here *)
 >| [ (* goal 1 (of 2) *)
      REWRITE_TAC [O_DEF] >> BETA_TAC >> rpt STRIP_TAC \\
      `STRONG_EQUIV y' E` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      `STRONG_EQUIV E' y` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      Q.EXISTS_TAC `y'` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y` >> ASM_REWRITE_TAC [],
      (* goal 2 (of 2) *)
      REWRITE_TAC [O_DEF] >> BETA_TAC >> rpt STRIP_TAC \\
      `STRONG_EQUIV E y` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      `STRONG_EQUIV y' E'` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      Q.EXISTS_TAC `y'` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y` >> ASM_REWRITE_TAC [] ]);

val CONVERSE_STRONG_BISIM_UPTO = store_thm (
   "CONVERSE_STRONG_BISIM_UPTO",
  ``!Wbsm. STRONG_BISIM_UPTO Wbsm ==> STRONG_BISIM_UPTO (\x y. Wbsm y x)``,
    GEN_TAC
 >> PURE_ONCE_REWRITE_TAC [STRONG_BISIM_UPTO]
 >> BETA_TAC
 >> rpt STRIP_TAC
 >> RES_TAC (* 2 sub-goals here *)
 >| [ (* goal 1 (of 2) *)
      Q.EXISTS_TAC `E1'` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [CONVERSE_STRONG_BISIM_UPTO_lemma] \\
      ASM_REWRITE_TAC [],
      (* goal 2 (of 2) *)
      Q.EXISTS_TAC `E2'` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [CONVERSE_STRONG_BISIM_UPTO_lemma] \\
```

```
      ASM_REWRITE_TAC [] ]);

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]
 >> rpt 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_UPTO_THM = store_thm (
   "STRONG_BISIM_UPTO_THM",
  ``!Bsm. STRONG_BISIM_UPTO Bsm ==> Bsm RSUBSET STRONG_EQUIV``,
    rpt STRIP_TAC
 >> IMP_RES_TAC STRONG_BISIM_UPTO_LEMMA
 >> IMP_RES_TAC STRONG_BISIM_SUBSET_STRONG_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]
 >> rpt 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) *));

(******************************************************************************)
(*
*)
(*                       Weak Bisimulation Upto ~
*)
(*
*)
(******************************************************************************)

(* NOTE: the definition in Milner's book [1] is wrong, we use the one in Gorrieri's book [
         double-confirmed with Sangiorgi's book [3].

   IMPORTANT: in HOL's big "O", the second argument to composition acts on the "input" fir
         so we need to revert the order of (?a O Wbsm O ?b) when ?a and ?b are different.
 *)
val WEAK_BISIM_UPTO = new_definition (
   "WEAK_BISIM_UPTO",
  ``WEAK_BISIM_UPTO (Wbsm: ('a, 'b) simulation) =
    !E E'.
        Wbsm E E' ==>
        (!l.
          (!E1. TRANS E  (label l) E1 ==>
                ?E2. WEAK_TRANS E' (label l) E2 /\ (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E
          (!E2. TRANS E' (label l) E2 ==>
                ?E1. WEAK_TRANS E  (label l) E1 /\ (STRONG_EQUIV O Wbsm O WEAK_EQUIV) E1 E
        (!E1. TRANS E  tau E1 ==> ?E2. EPS E' E2 /\ (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 
        (!E2. TRANS E' tau E2 ==> ?E1. EPS E  E1 /\ (STRONG_EQUIV O Wbsm O WEAK_EQUIV) E1 

(* Extracted above definition into smaller pieces for easier uses *)
val WEAK_BISIM_UPTO_TRANS_label = store_thm (
   "WEAK_BISIM_UPTO_TRANS_label",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==>
        !E E'. Wbsm E E' ==>
               !l E1. TRANS E (label l) E1 ==>
                      ?E2. WEAK_TRANS E' (label l) E2 /\ (WEAK_EQUIV O Wbsm O STRONG_EQUIV
```

```
    PROVE_TAC [WEAK_BISIM_UPTO]);

val WEAK_BISIM_UPTO_TRANS_label' = store_thm (
   "WEAK_BISIM_UPTO_TRANS_label'",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==>
        !E E'. Wbsm E E' ==>
               !l E2. TRANS E' (label l) E2 ==>
                      ?E1. WEAK_TRANS E  (label l) E1 /\ (STRONG_EQUIV O Wbsm O WEAK_EQUIV
    PROVE_TAC [WEAK_BISIM_UPTO]);

val WEAK_BISIM_UPTO_TRANS_tau = store_thm (
   "WEAK_BISIM_UPTO_TRANS_tau",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==>
        !E E'. Wbsm E E' ==>
               !E1. TRANS E tau E1 ==> ?E2. EPS E' E2 /\ (WEAK_EQUIV O Wbsm O STRONG_EQUIV
    PROVE_TAC [WEAK_BISIM_UPTO]);

val WEAK_BISIM_UPTO_TRANS_tau' = store_thm (
   "WEAK_BISIM_UPTO_TRANS_tau'",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==>
        !E E'. Wbsm E E' ==>
               !E2. TRANS E' tau E2 ==> ?E1. EPS E  E1 /\ (STRONG_EQUIV O Wbsm O WEAK_EQUI
    PROVE_TAC [WEAK_BISIM_UPTO]);

val IDENTITY_WEAK_BISIM_UPTO_lemma = store_thm (
   "IDENTITY_WEAK_BISIM_UPTO_lemma",
  ``!E. (WEAK_EQUIV O (\x y. x = y) O STRONG_EQUIV) E E``,
    GEN_TAC >> REWRITE_TAC [O_DEF] >> BETA_TAC
 >> Q.EXISTS_TAC `E` >> REWRITE_TAC [WEAK_EQUIV_REFL]
 >> Q.EXISTS_TAC `E` >> REWRITE_TAC [STRONG_EQUIV_REFL]);

val IDENTITY_WEAK_BISIM_UPTO_lemma' = store_thm (
   "IDENTITY_WEAK_BISIM_UPTO_lemma'",
  ``!E. (STRONG_EQUIV O (\x y. x = y) O WEAK_EQUIV) E E``,
    GEN_TAC >> REWRITE_TAC [O_DEF] >> BETA_TAC
 >> Q.EXISTS_TAC `E` >> REWRITE_TAC [STRONG_EQUIV_REFL]
 >> Q.EXISTS_TAC `E` >> REWRITE_TAC [WEAK_EQUIV_REFL]);

val IDENTITY_WEAK_BISIM_UPTO = store_thm (
   "IDENTITY_WEAK_BISIM_UPTO", ``WEAK_BISIM_UPTO (\x y. x = y)``,
    PURE_ONCE_REWRITE_TAC [WEAK_BISIM_UPTO]
 >> BETA_TAC
 >> REPEAT STRIP_TAC (* 4 sub-goals here *)
 >| [ (* goal 1 (of 4) *)
      ASSUME_TAC (REWRITE_RULE [ASSUME ``E :('a, 'b) CCS = E'``]
                               (ASSUME ``TRANS E (label l) E1``)) \\
      IMP_RES_TAC TRANS_IMP_WEAK_TRANS \\
      Q.EXISTS_TAC `E1` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [IDENTITY_WEAK_BISIM_UPTO_lemma],
      (* goal 2 (of 4) *)
      IMP_RES_TAC TRANS_IMP_WEAK_TRANS \\
      Q.EXISTS_TAC `E2` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [IDENTITY_WEAK_BISIM_UPTO_lemma'],
      (* goal 3 (of 4) *)
      ASSUME_TAC (REWRITE_RULE [ASSUME ``E :('a, 'b) CCS = E'``]
                               (ASSUME ``TRANS E tau E1``)) \\
      IMP_RES_TAC ONE_TAU \\
      Q.EXISTS_TAC `E1` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [IDENTITY_WEAK_BISIM_UPTO_lemma],
      (* goal 4 (of 4) *)
      IMP_RES_TAC ONE_TAU \\
```

```
      Q.EXISTS_TAC `E2` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [IDENTITY_WEAK_BISIM_UPTO_lemma'] ]);

val CONVERSE_WEAK_BISIM_UPTO_lemma = store_thm (
   "CONVERSE_WEAK_BISIM_UPTO_lemma",
  ``!Wbsm E E'. (WEAK_EQUIV O (\x y. Wbsm y x) O STRONG_EQUIV) E E' =
                (STRONG_EQUIV O Wbsm O WEAK_EQUIV) E' E``,
    rpt GEN_TAC
 >> EQ_TAC (* 2 sub-goals here *)
 >| [ (* goal 1 (of 2) *)
      REWRITE_TAC [O_DEF] >> BETA_TAC >> rpt STRIP_TAC \\
      `STRONG_EQUIV y' E` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      `WEAK_EQUIV E' y` by PROVE_TAC [WEAK_EQUIV_SYM] \\
      Q.EXISTS_TAC `y'` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y` >> ASM_REWRITE_TAC [],
      (* goal 2 (of 2) *)
      REWRITE_TAC [O_DEF] >> BETA_TAC >> rpt STRIP_TAC \\
      `STRONG_EQUIV E y` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      `WEAK_EQUIV y' E'` by PROVE_TAC [WEAK_EQUIV_SYM] \\
      Q.EXISTS_TAC `y'` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y` >> ASM_REWRITE_TAC [] ]);

val CONVERSE_WEAK_BISIM_UPTO_lemma' = store_thm (
   "CONVERSE_WEAK_BISIM_UPTO_lemma'",
  ``!Wbsm E E'. (STRONG_EQUIV O (\x y. Wbsm y x) O WEAK_EQUIV) E E' =
                (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E' E``,
    rpt GEN_TAC
 >> EQ_TAC (* 2 sub-goals here *)
 >| [ (* goal 1 (of 2) *)
      REWRITE_TAC [O_DEF] >> BETA_TAC >> rpt STRIP_TAC \\
      `STRONG_EQUIV E' y` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      `WEAK_EQUIV y' E` by PROVE_TAC [WEAK_EQUIV_SYM] \\
      Q.EXISTS_TAC `y'` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y` >> ASM_REWRITE_TAC [],
      (* goal 2 (of 2) *)
      REWRITE_TAC [O_DEF] >> BETA_TAC >> rpt STRIP_TAC \\
      `STRONG_EQUIV y' E'` by PROVE_TAC [STRONG_EQUIV_SYM] \\
      `WEAK_EQUIV E y` by PROVE_TAC [WEAK_EQUIV_SYM] \\
      Q.EXISTS_TAC `y'` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y` >> ASM_REWRITE_TAC [] ]);

val CONVERSE_WEAK_BISIM_UPTO = store_thm (
   "CONVERSE_WEAK_BISIM_UPTO",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==> WEAK_BISIM_UPTO (\x y. Wbsm y x)``,
    GEN_TAC
 >> PURE_ONCE_REWRITE_TAC [WEAK_BISIM_UPTO]
 >> BETA_TAC
 >> rpt STRIP_TAC
 >> RES_TAC (* 4 sub-goals here *)
 >| [ (* goal 1 (of 4) *)
      Q.EXISTS_TAC `E1'` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [CONVERSE_WEAK_BISIM_UPTO_lemma] \\
      ASM_REWRITE_TAC [],
      (* goal 2 (of 4) *)
      Q.EXISTS_TAC `E2'` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [CONVERSE_WEAK_BISIM_UPTO_lemma'] \\
      ASM_REWRITE_TAC [],
      (* goal 3 (of 4) *)
      Q.EXISTS_TAC `E1'` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [CONVERSE_WEAK_BISIM_UPTO_lemma] \\
      ASM_REWRITE_TAC [],
```

```
      (* goal 4 (of 4) *)
      Q.EXISTS_TAC `E2'` >> ASM_REWRITE_TAC [] \\
      REWRITE_TAC [CONVERSE_WEAK_BISIM_UPTO_lemma'] \\
      ASM_REWRITE_TAC [] ]);

val WEAK_BISIM_UPTO_EPS = store_thm ((* NEW *)
   "WEAK_BISIM_UPTO_EPS",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==>
        !E E'. Wbsm E E' ==>
               !E1. EPS E E1 ==> ?E2. EPS E' E2 /\ (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E
    rpt STRIP_TAC
 >> PAT_X_ASSUM ``WEAK_BISIM_UPTO Wbsm`` MP_TAC
 >> Q.PAT_X_ASSUM `Wbsm E E'` MP_TAC
 >> POP_ASSUM MP_TAC
 >> Q.SPEC_TAC (`E1`, `E1`)
 >> Q.SPEC_TAC (`E`, `E`)
 >> HO_MATCH_MP_TAC EPS_ind_right (* must use right induct here! *)
 >> rpt STRIP_TAC (* 2 sub-goals here *)
 >| [ (* goal 1 (of 2) *)
      Q.EXISTS_TAC `E'` \\
      RW_TAC std_ss [EPS_REFL] \\
      REWRITE_TAC [O_DEF] >> BETA_TAC \\
      Q.EXISTS_TAC `E'` >> REWRITE_TAC [WEAK_EQUIV_REFL] \\
      Q.EXISTS_TAC `E` >> ASM_REWRITE_TAC [STRONG_EQUIV_REFL],
      (* goal 2 (of 2) *)
      RES_TAC \\
      POP_ASSUM (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF])) \\
      STRIP_ASSUME_TAC (ONCE_REWRITE_RULE [PROPERTY_STAR]
                                          (ASSUME ``STRONG_EQUIV E1 y'``)) \\
      RES_TAC \\
      NTAC 2 (POP_ASSUM K_TAC) \\
      STRIP_ASSUME_TAC (REWRITE_RULE [WEAK_BISIM_UPTO]
                                     (ASSUME ``WEAK_BISIM_UPTO Wbsm``)) \\
      POP_ASSUM (STRIP_ASSUME_TAC o (Q.SPECL [`y'`, `y`])) \\
      RES_TAC \\
      NTAC 7 (POP_ASSUM K_TAC) \\
      Q.PAT_X_ASSUM `Wbsm y' y ==> X` K_TAC \\
      Q.PAT_X_ASSUM `!l E1. TRANS y' (label l) E1 ==> P` K_TAC \\
      Q.PAT_X_ASSUM `!l E2. TRANS y (label l) E2 ==> P` K_TAC \\
      IMP_RES_TAC WEAK_EQUIV_EPS \\
      Q.EXISTS_TAC `E2'''` \\
      CONJ_TAC >- IMP_RES_TAC EPS_TRANS \\
      Q.PAT_X_ASSUM `X E2' E2''` MP_TAC \\
      REWRITE_TAC [O_DEF] >> BETA_TAC >> rpt STRIP_TAC \\
(* Induct case:
      E                 Wbsm                E'
                                            //
      ...                                  eps
                                            //
      E1    ~    y'     Wbsm      y    =~   E2
      /        /                  \\        //
     tau     tau                  eps      eps
      /      /                      \\      //
      E1' ~ E2' ~ y''' Wbsm y'' =~ E2'' =~ E2'''
 *)
      `WEAK_EQUIV y'' E2'''` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
      `STRONG_EQUIV E1' y'''` by PROVE_TAC [STRONG_EQUIV_TRANS] \\
      Q.EXISTS_TAC `y''` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y'''` >> ASM_REWRITE_TAC [] ]);

val WEAK_BISIM_UPTO_EPS' = store_thm ((* NEW *)
```

```
   "WEAK_BISIM_UPTO_EPS'",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==>
        !E E'. Wbsm E E' ==>
               !E2. EPS E' E2 ==> ?E1. EPS E E1 /\ (STRONG_EQUIV O Wbsm O WEAK_EQUIV) E1 E
    GEN_TAC >> DISCH_TAC
 >> POP_ASSUM (ASSUME_TAC o (MATCH_MP CONVERSE_WEAK_BISIM_UPTO))
 >> IMP_RES_TAC WEAK_BISIM_UPTO_EPS
 >> POP_ASSUM MP_TAC
 >> BETA_TAC >> rpt STRIP_TAC
 >> RES_TAC
 >> Q.EXISTS_TAC `E2'` >> ASM_REWRITE_TAC []
 >> REWRITE_TAC [GSYM CONVERSE_WEAK_BISIM_UPTO_lemma]
 >> ASM_REWRITE_TAC []);

(* Proof sketch:
      E             Wbsm                E'
      //                                //
      eps                               eps
      //                                //
      E1' ~ y'      Wbsm      y  =~     E2'    (WEAK_BISIM_UPTO_EPS)
      /     /                 //        //
      /     l                 l         //
      l     /                 //        l
      /   ~ E2'' (~ Wbsm =~) E2''' =~ //
      E2                                E2'''' (WEAK_BISIM_UPTO_TRANS_label)
      // ~  y'''   Wbsm      y''   =~ //
      eps   //                //        eps
      //    eps               eps       //
      //    //                //        //
      E1  ~ E2'5 (~ Wbsm =~) E2'6  =~ E2'7   (WEAK_BISIM_UPTO_EPS)
 *)
val WEAK_BISIM_UPTO_WEAK_TRANS_label = store_thm ((* NEW *)
   "WEAK_BISIM_UPTO_WEAK_TRANS_label",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==>
        !E E'. Wbsm E E' ==>
               !l E1. WEAK_TRANS E (label l) E1 ==>
                      ?E2. WEAK_TRANS E' (label l) E2 /\
                           (WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E2``,
    rpt STRIP_TAC
 >> IMP_RES_TAC WEAK_TRANS
 >> IMP_RES_TAC (MATCH_MP WEAK_BISIM_UPTO_EPS (* lemma 1 used here *)
                          (ASSUME ``WEAK_BISIM_UPTO Wbsm``))
 >> POP_ASSUM (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF]))
 >> IMP_RES_TAC PROPERTY_STAR_TRANS
 >> IMP_RES_TAC WEAK_BISIM_UPTO_TRANS_label
 >> POP_ASSUM K_TAC
 >> IMP_RES_TAC WEAK_EQUIV_WEAK_TRANS_label
 >> Know `(WEAK_EQUIV O Wbsm O STRONG_EQUIV) E2 E2''''`
 >- ( Q.PAT_X_ASSUM `X E2'' E2'''` MP_TAC \\
      REWRITE_TAC [O_DEF] >> BETA_TAC >> rpt STRIP_TAC \\
      `STRONG_EQUIV E2 y'''` by PROVE_TAC [STRONG_EQUIV_TRANS] \\
      `WEAK_EQUIV y'' E2''''` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
      Q.EXISTS_TAC `y''` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y'''` >> ASM_REWRITE_TAC [] )
 >> DISCH_TAC
 >> POP_ASSUM (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF]))
 >> IMP_RES_TAC (MATCH_MP STRONG_EQUIV_EPS
                          (ASSUME ``STRONG_EQUIV E2 y'''``))
 >> IMP_RES_TAC (Q.SPECL [`y'''`, `y''`]
                         (MATCH_MP WEAK_BISIM_UPTO_EPS (* lemma 1 used here *)
                                   (ASSUME ``WEAK_BISIM_UPTO Wbsm``)))
```

```
 >> NTAC 3 (POP_ASSUM K_TAC)
 >> IMP_RES_TAC (MATCH_MP WEAK_EQUIV_EPS
                          (ASSUME ``WEAK_EQUIV y'' E2''''``))
 >> Know `(WEAK_EQUIV O Wbsm O STRONG_EQUIV) E1 E2''''''''`
 >- ( Q.PAT_X_ASSUM `X E2'''''' E2''''''''` MP_TAC \\
      REWRITE_TAC [O_DEF] >> BETA_TAC >> rpt STRIP_TAC \\
      `STRONG_EQUIV E1 y''''''` by PROVE_TAC [STRONG_EQUIV_TRANS] \\
      `WEAK_EQUIV y'''' E2''''''''` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
      Q.EXISTS_TAC `y''''` >> ASM_REWRITE_TAC [] \\
      Q.EXISTS_TAC `y''''''` >> ASM_REWRITE_TAC [] )
 >> DISCH_TAC
 >> Q.EXISTS_TAC `E2''''''''`
 >> ASM_REWRITE_TAC []
 >> MATCH_MP_TAC EPS_AND_WEAK
 >> take [`E2'`, `E2''''`]
 >> ASM_REWRITE_TAC []);

val WEAK_BISIM_UPTO_WEAK_TRANS_label' = store_thm ((* NEW *)
   "WEAK_BISIM_UPTO_WEAK_TRANS_label'",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==>
        !E E'. Wbsm E E' ==>
               !l E2. WEAK_TRANS E' (label l) E2 ==>
                      ?E1. WEAK_TRANS E (label l) E1 /\
                           (STRONG_EQUIV O Wbsm O WEAK_EQUIV) E1 E2``,
    GEN_TAC >> DISCH_TAC
 >> POP_ASSUM (ASSUME_TAC o (MATCH_MP CONVERSE_WEAK_BISIM_UPTO))
 >> IMP_RES_TAC WEAK_BISIM_UPTO_WEAK_TRANS_label
 >> POP_ASSUM MP_TAC
 >> BETA_TAC >> rpt STRIP_TAC
 >> RES_TAC
 >> Q.EXISTS_TAC `E2'` >> ASM_REWRITE_TAC []
 >> REWRITE_TAC [GSYM CONVERSE_WEAK_BISIM_UPTO_lemma]
 >> ASM_REWRITE_TAC []);

(* If S is a bisimulation up to WEAK_EQUIV, then (WEAK_EQUIV O S O WEAK_EQUIV) is
   a weak bisimultion. *)
val WEAK_BISIM_UPTO_LEMMA = store_thm (
   "WEAK_BISIM_UPTO_LEMMA",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==> WEAK_BISIM (WEAK_EQUIV O Wbsm O WEAK_EQUIV)``,
    GEN_TAC
 >> REWRITE_TAC [WEAK_BISIM, O_DEF]
 >> rpt STRIP_TAC (* 4 sub-goals here *)
 >| [ (* goal 1 (of 4) *)
      IMP_RES_TAC (MATCH_MP WEAK_EQUIV_TRANS_label (ASSUME ``WEAK_EQUIV E y``)) \\
      IMP_RES_TAC (MATCH_MP WEAK_BISIM_UPTO_WEAK_TRANS_label
                            (ASSUME ``WEAK_BISIM_UPTO Wbsm``)) \\
      IMP_RES_TAC (MATCH_MP WEAK_EQUIV_WEAK_TRANS_label
                            (ASSUME ``WEAK_EQUIV y E'``)) \\
      Q.EXISTS_TAC `E2''` >> ASM_REWRITE_TAC [] \\
      Q.PAT_X_ASSUM `X E2 E2'` (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF])) \\
(***
              E     ~=    y'     Wbsm      y     ~=    E'
              |          //                \\         //
             !l        l                    l         l
              |       //                     \\       //
              E1  ~= E2  ~ y''' Wbsm y'' ~= E2' ~= E2''
 ***)
      `WEAK_EQUIV E2 y'''` by PROVE_TAC [STRONG_IMP_WEAK_EQUIV] \\
      `WEAK_EQUIV E1 y'''` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
      `WEAK_EQUIV y'' E2''` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
      Q.EXISTS_TAC `y''` >> ASM_REWRITE_TAC [] \\
```

```
        Q.EXISTS_TAC `y'''` >> ASM_REWRITE_TAC [],
        (* goal 2 (of 4) *)
        IMP_RES_TAC (MATCH_MP WEAK_EQUIV_TRANS_label' (ASSUME ``WEAK_EQUIV y E'``)) \\
        IMP_RES_TAC (MATCH_MP WEAK_BISIM_UPTO_WEAK_TRANS_label'
                              (ASSUME ``WEAK_BISIM_UPTO Wbsm``)) \\
        IMP_RES_TAC (MATCH_MP WEAK_EQUIV_WEAK_TRANS_label'
                              (ASSUME ``WEAK_EQUIV E y'``)) \\
        Q.EXISTS_TAC `E1''` >> ASM_REWRITE_TAC [] \\
        Q.PAT_X_ASSUM `X E1' E1` (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF])) \\
(***
              E     ~=      y'       Wbsm     y    ~=    E'
              ||          //                  \\         |
              l         l                      l         l
              ||       //                       \\       |
              E1'' ~= E1' ~= y''' Wbsm y'' ~ E1 ~= E2
 ***)
        `WEAK_EQUIV E1'' y'''` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
        `WEAK_EQUIV y'' E1` by PROVE_TAC [STRONG_IMP_WEAK_EQUIV] \\
        `WEAK_EQUIV y'' E2` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
        Q.EXISTS_TAC `y''` >> ASM_REWRITE_TAC [] \\
        Q.EXISTS_TAC `y'''` >> ASM_REWRITE_TAC [],
        (* goal 3 (of 4) *)
        IMP_RES_TAC (MATCH_MP WEAK_EQUIV_TRANS_tau (ASSUME ``WEAK_EQUIV E y'``)) \\
        IMP_RES_TAC (MATCH_MP WEAK_BISIM_UPTO_EPS (ASSUME ``WEAK_BISIM_UPTO Wbsm``)) \\
        IMP_RES_TAC (MATCH_MP WEAK_EQUIV_EPS (ASSUME ``WEAK_EQUIV y E'``)) \\
        Q.EXISTS_TAC `E2''` >> ASM_REWRITE_TAC [] \\
        Q.PAT_X_ASSUM `X E2 E2'` (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF])) \\
(***
              E     ~=    y'     Wbsm      y     ~=    E'
              |         //               \\          ||
             tau      eps                 eps       eps
              |       //                   \\        ||
              E1  ~= E2  ~ y''' Wbsm y'' ~= E2' ~= E2''
 ***)
        `WEAK_EQUIV E2 y'''` by PROVE_TAC [STRONG_IMP_WEAK_EQUIV] \\
        `WEAK_EQUIV E1 y'''` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
        `WEAK_EQUIV y'' E2''` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
        Q.EXISTS_TAC `y''` >> ASM_REWRITE_TAC [] \\
        Q.EXISTS_TAC `y'''` >> ASM_REWRITE_TAC [],
        (* goal 4 (of 4) *)
        IMP_RES_TAC (MATCH_MP WEAK_EQUIV_TRANS_tau' (ASSUME ``WEAK_EQUIV y E'``)) \\
        IMP_RES_TAC (MATCH_MP WEAK_BISIM_UPTO_EPS' (ASSUME ``WEAK_BISIM_UPTO Wbsm``)) \\
        IMP_RES_TAC (MATCH_MP WEAK_EQUIV_EPS' (ASSUME ``WEAK_EQUIV E y'``)) \\
        Q.EXISTS_TAC `E1''` >> ASM_REWRITE_TAC [] \\
        Q.PAT_X_ASSUM `X E1' E1` (STRIP_ASSUME_TAC o (REWRITE_RULE [O_DEF])) \\
(***
              E     ~=      y'       Wbsm     y    ~=    E'
              ||          //                  \\         |
              eps       eps                    eps      tau
              ||       //                       \\       |
              E1'' ~= E1' ~= y''' Wbsm y'' ~ E1 ~= E2
 ***)
        `WEAK_EQUIV E1'' y'''` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
        `WEAK_EQUIV y'' E1` by PROVE_TAC [STRONG_IMP_WEAK_EQUIV] \\
        `WEAK_EQUIV y'' E2` by PROVE_TAC [WEAK_EQUIV_TRANS] \\
        Q.EXISTS_TAC `y''` >> ASM_REWRITE_TAC [] \\
        Q.EXISTS_TAC `y'''` >> ASM_REWRITE_TAC [] ]);

val WEAK_BISIM_UPTO_THM = store_thm (
   "WEAK_BISIM_UPTO_THM",
  ``!Wbsm. WEAK_BISIM_UPTO Wbsm ==> Wbsm RSUBSET WEAK_EQUIV``,
```

```
    rpt STRIP_TAC
 >> IMP_RES_TAC WEAK_BISIM_UPTO_LEMMA
 >> IMP_RES_TAC WEAK_BISIM_SUBSET_WEAK_EQUIV
 >> Suff `Wbsm RSUBSET (WEAK_EQUIV O Wbsm O WEAK_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]
 >> rpt STRIP_TAC
 >> `WEAK_EQUIV x x /\ WEAK_EQUIV y y` by PROVE_TAC [WEAK_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) *));

(* Bibliography:
 *
 * [1] Milner, R.: Communication and concurrency. (1989).
.* [2] Gorrieri, R., Versari, C.: Introduction to Concurrency Theory. Springer, Cham (2015
 * [3] Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University P
 * [4] Sangiorgi, D., Rutten, J.: Advanced Topics in Bisimulation and Coinduction.
                                  Cambridge University Press (2011).
 *)

val _ = export_theory ();
val _ = print_theory_to_file "-" "BisimulationUpto.lst";
val _ = DB.html_theory "BisimulationUpto";

(* last updated: Aug 5, 2017 *)
```