Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 212 (2008) 269–284

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Tree Process Calculus

Mingren Chai Nan Qu Ying Jiang[1] ,[2]

State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing 100190, P.R.China

Abstract

We extend Robin Milner's sequential process calculus to tree process calculus in a way similar to that tree automata extend word automata. By this way, we establish a sound and complete inference system which describes the equivalence via bisimulation over tree processes. As a corollary, we obtain a new equivalence relation between tree automata, which is strictly finer than the classical equivalence over tree automata, and is strictly coarser than the equivalence relation via bisimulation defined in [1].

Keywords: Tree process calculus, sequential process calculus, tree automata, bisimulation.

1 Introduction

Classical (word) automata theory was introduced for analyzing relations between automata from the view of language equivalence [6]. Since then it has been developed in several directions, one of which is Robin Milner's nondeterministic sequential process calculus (NSPC for short) [7], invented for analyzing properties of sequential processes by observing the behavior of processes. In the last two decades, a series of correlated theories were developed rapidly, noticeably, CCS [9], π-calculus [10].

Sequential process calculus deals with relations between automata from the view of behavior. It considers the matters as when two different automata may be regarded as having the same behavior, and how automata can be classified in terms of this behavior [7]. Also an operational semantic known as labelled transition system (LTS) was offered for sequential process calculus. A LTS can be thought of as an automaton without a start state or accepting states [10]. On the other hand, tree automata [2,4] can be considered as a kind of extension of (word) automata, which provides among others a mathematical model for reactive system as operating

[1] Corresponding author

[2] Email: {chaimr, qunan, jy}@ios.ac.cn

doi:10.1016/j.entcs.2008.04.067

systems or banking systems and get successful applications in concurrent program verification, automated theorem proving and XML schema languages.

The goal of this paper is to extend NSPC to tree process calculus (see Section 3 for the formal definition) in a way similar to that tree automata extend word automata in order to analyze properties of tree processes and the relationship between tree process calculus and tree automata. First, we establish an inference system. Secondly we provide the calculus an operational semantic called tree labelled transition system (TLTS), and define a notion of bisimulation between tree processes and the equivalence relation derived from it. Finally we prove the soundness and completeness of the calculus with respect to the equivalence via bisimulation. As a corollary, we obtain a new equivalence relation over tree automata, which is strictly finer than the equivalence via language equality of tree automata, and is strictly coarser than the equivalence via bisimulation defined by Parosh Aziz Abdulla et al. in [1]. This equivalence relation may give smaller automata during minimization. However, it is not considered how costly checking the new equivalence will be compared to [1].

The rest of the paper is structured along the line of background knowledge, tree process calculus, bisimulation, the theory of bisimulation, application and conclusion. Because of size limit, the details of some proofs are omitted.

2 Preliminaries

Leaving the rudimentary sequential process calculus and tree automata to [2,4,7,8,9,10], we begin with a description of labelled transition system. Let $\mathcal{A}$ be a set of actions, sometimes called an alphabet. A *labelled transition system* (LTS) over $\mathcal{A}$ is a triple $(Q, \mathcal{A}, \Delta)$ consisting of (1) a set Q of states; (2) a set $\mathcal{A}$ of actions; (3) a ternary relation $\Delta \subseteq Q \times \mathcal{F} \times Q$, known as a transition relation. We denote $(q, \alpha, q') \in \Delta$ by $q \overset{\alpha}{\to} q'$ and call q the source and q' the target of the transition. In general, q_n is said to be a derivative of q under $\alpha_1\alpha_2\cdots\alpha_n$ if $q \overset{\alpha_1}{\to} q_1 \overset{\alpha_2}{\to} q_2 \cdots \overset{\alpha_n}{\to} q_n$.

Let $\mathcal{A}$ be a set of of atomic actions, $\mathcal{V}$ be a denumerable set of variables, $X \in \mathcal{V}$ and $\alpha \in \mathcal{A}$. The set $\mathbb{P}^{seq}$ of *sequential process expressions* is the smallest class defined inductively as follows:

$$\begin{array}{rll} \mathbb{P}^{seq} ::= & 0 & \text{(inaction)} \\ & \mid \alpha.P & \text{(prefix)} \\ & \mid X & \text{(variables)} \\ & \mid P_1 + P_2 & \text{(summation)} \\ & \mid \mu X.P & \text{(recursion)} \end{array}$$

The *labelled transition system of sequential processes* over $\mathcal{A}$ is defined to be the LTS $(\mathbb{P}^{seq}, \mathcal{A}, \Delta)$ where the transitions are exactly those which can be inferred from the rules below, together with α-conversion:

TRANSITION RULES

Prefix $\alpha.P \xrightarrow{\alpha} P$, provided $\alpha \in \mathcal{A}$

Sum1 $\dfrac{P_1 \xrightarrow{\alpha} P}{P_1 + P_2 \xrightarrow{\alpha} P}$ Sum2 $\dfrac{P_2 \xrightarrow{\alpha} P}{P_1 + P_2 \xrightarrow{\alpha} P}$

Rec $\dfrac{P\{\mu X.P/X\} \xrightarrow{\alpha} P'}{\mu X.P \xrightarrow{\alpha} P'}$

Now we switch to the notions concerning tree automata.

Let N be the set of positive integers. A *ranked alphabet* is a couple $(\mathcal{F}, \mathit{Arity})$ where $\mathcal{F}$ is a finite set and Arity is a mapping from $\mathcal{F}$ into N. We denote by $\mathit{Arity}(f)$ the *arity* of a symbol $f \in \mathcal{F}$ and $\mathcal{F}_p$ the set of symbols of arity p. Symbols of arity $0, 1, \ldots, p$ are called constants, unary, ... and p-ary symbols, respectively. We assume that $\mathcal{F}$ contains at least one constant.

The set $T(\mathcal{F})$ of *terms* over the ranked alphabet $\mathcal{F}$ is the smallest set defined by:

- $\mathcal{F}_0 \subseteq T(\mathcal{F})$;
- if $p \geq 1$, $f \in \mathcal{F}_p$ and $t_1, \ldots, t_p \in T(\mathcal{F})$, then $f(t_1, \ldots, t_p) \in T(\mathcal{F})$.

A subset of $T(\mathcal{F})$ is called a *tree language.*

A *non-deterministic top-down finite tree automata* (top-down NFTA) over $\mathcal{F}$ is a tuple $A = (Q, \mathcal{F}, Q_I, \Delta)$ where Q is a finite set of states, $\mathcal{F}$ a set of ranked alphabet, $Q_I (\subseteq Q)$ is a set of initial states, Δ is a finite set of transition rules $q \rightarrow f(q_1, \ldots, q_n)$ with $f \in \mathcal{F}_n$ and $q_1, \ldots, q_n, q \in Q$ for some $n \in N$. Obviously, Δ determines a function $\Delta^* : Q \rightarrow \mathcal{P}(T(\mathcal{F}))$ by (the least fixpoint equation)

$$\Delta^*(q) = \{t \mid t = f(t_1, \ldots, t_k), q \rightarrow f(q_1, \ldots, q_k) \in \Delta, t_i \in \Delta^*(q_i), i \in \{1, \ldots, k\}\}$$

The tree language recognized by A is the set $L(A) = \{t \mid t \in \Delta^*(q), q \in Q_I\}$.

Notice that the expressive power of top-down NFTA and bottom-up NFTA is the same [2]. More precisely, the class of languages accepted by top-down NFTAs is exactly the class of languages accepted by bottom-up NFTAs. In this paper, we only consider the set of top-down NFTAs.

3 Tree Process Calculus

This section introduces a process calculus which will be called tree process calculus (TPC for short). The essential difference between TPC and R. Milner's sequential process calculus is that the actions of sequential process calculus are all unaries while those of tree process calculus have different arities.

3.1 Tree Labelled Transition System

A tree labelled transition system is, roughly speaking, a labelled transition system given in Section 2, but the labels have different arities, the source of each transition rule is a state and the target of the transition is a vector of states.

Definition 3.1 [Tree labelled transition system] Let $\mathcal{F}$ be a set of ranked symbols as the set of actions. A *tree labelled transition system (TLTS)* over $\mathcal{F}$ is a triple $(Q, \mathcal{F}, \Delta)$ consisting of: (i) a set Q of states; (ii) a set $\mathcal{F}$ of ranked symbols; (iii) a ternary relation (called a transition relation) $\Delta \subseteq Q \times \mathcal{F} \times Q^*$, where $Q^0 = \{\varepsilon\}$, ε denotes the 0-dimension vector, $Q^n = \underbrace{Q \times Q \times \cdots \times Q}_{n}$, for $n \geq 1$ and $Q^* = \bigcup_{n \in N} Q^n$.

Note that $(q, f, \overrightarrow{q}) \in \Delta$ means $q \in Q$, $f \in \mathcal{F}_i$ and $\overrightarrow{q} \in Q^i$, for some $i \geq 0$.

Informally, a TLTS can be thought of as a tree automata without initial states or final states. In fact each different selection of a set of initial states defines a different top-down tree automaton, based upon the same TLTS.

3.2 Tree Process Expressions and TLTS of Tree Process

This section focus on the syntax of TPC.

Definition 3.2 [Tree process expressions] Let $\mathcal{F}$ be a set of ranked symbols as the set of actions. Let $\mathcal{V}$ be a set of process variables, and X range over $\mathcal{V}$. The class $\mathbb{P}$ of *tree process expressions* over $\mathcal{F}$ is the smallest class inductively defined as follows:

$$\mathbb{P} ::= f.(P_1, \ldots, P_n) \text{ (prefix)} \mid X \text{ (variables)} \mid P_1 + P_2 \text{ (summation)} \mid \mu X.P \text{ (recursion)}$$

where $f \in \mathcal{F}_n$ and $(P_1, \ldots, P_n)$ is an n-dimension vector of expressions.

The meaning of the combinators is similar to the ones of sequential case, the one and only difference is that here $(P_1, P_2, \ldots P_n)$ is not a process expression but a vector consists of process expressions. Recall that ε denotes the vector of 0-dimension.

For legibility, we adopt the convention that the combinators defined above have the following precedence: prefix, recursion and summation.

Example 3.3 $a.\varepsilon$ $f.(a.\varepsilon)$ $g.(b.\varepsilon, c.\varepsilon + P_1)$ $h.(P_1, P_2, P_3, P_4)$ are all tree process expressions, where $P_1, P_2, P_3, P_4 \in \mathbb{P}$, $a, b, c \in \mathcal{F}_0$, $f \in \mathcal{F}_1$, $g \in \mathcal{F}_2$ and $h \in \mathcal{F}_4$.

Definition 3.4 [Free variables and guarded variables] X is *free* in a tree process expression P if P contains an occurrence of X not contained in subexpressions $\mu X.P'$. X is *guarded* in an expression P if every free occurrence of X in P is contained in a subexpression $f.\overrightarrow{P'}$. When X is not guarded in P, we write $P \rhd X$.

Example 3.5 X is guarded in $f.(X, X)$, $\mu X.X$ and $Y + Z$; but not in X, $f.(X, Y) + X$ or $\mu Y.(Y + X)$, where $f \in \mathcal{F}_2$.

In the following, $P\{P_1,\dots,P_n/X_1,\dots,X_n\}$ will denote the result of simultaneously substituting P_i for each free occurrence of X_i in P $(1 \le i \le n)$, renaming bound variables if necessary.

In giving meaning to our basic language, we define

Definition 3.6 [The TLTS of tree process] Let $\mathcal{F}$ be a set of ranked symbols. Let $\mathbb{P}$ be the class of tree process expressions over $\mathcal{F}$. *A TLTS of tree process* is the triple $(Q,\mathcal{F},\Delta)$, where $Q \subset_{fin} \mathbb{P}$ and the transition relation Δ is exactly those which can be inferred from the rules below, together with α-conversion:

TRANSITION RULES

Prefix $f.(P_1,\dots,P_n) \xrightarrow{f} (P_1,\dots,P_n)$, provided $f \in \mathcal{F}_n$

Sum1 $\dfrac{P_1 \xrightarrow{f} \overrightarrow{P}}{P_1 + P_2 \xrightarrow{f} \overrightarrow{P}}$ Sum2 $\dfrac{P_2 \xrightarrow{f} \overrightarrow{P}}{P_1 + P_2 \xrightarrow{f} \overrightarrow{P}}$

Rec $\dfrac{P\{\mu X.P/X\} \xrightarrow{f} \overrightarrow{P}}{\mu X.P \xrightarrow{f} \overrightarrow{P}}$

Again, the rules are similar to the ones in the sequential case, the one and only difference is that here $\overrightarrow{P}$ does not denote a state but a vector consists of states.

The following proposition will be useful later.

Proposition 3.7 *Let $f \in \mathcal{F}_n$ $(n \ge 0)$. If $P\{Q/X\} \xrightarrow{f} (P_1,\dots,P_n)$, then:*

(i) *If X is guarded in P, then $P \xrightarrow{f} (P_1',\dots,P_n')$ and $P_i \equiv P_i'\{Q/X\}$ $(1 \le i \le n)$;*

(ii) *If X is not guarded in P, then $P \xrightarrow{f} (P_1',\dots,P_n')$ and $P_i \equiv P_i'\{Q/X\}$ for any $i \in \{1,\dots,n\}$, or $Q \xrightarrow{f} (P_1,\dots,P_n)$;*

where $\equiv$ denotes the syntax equality.

Proof. Both statements are proved by induction on the depth of the inference to get $P\{Q/X\} \xrightarrow{f} (P_1,\dots,P_n)$. The detail is argued by cases on the form of P. □

3.3 Multi-processes and Multi-step Transitions

In the previous section, we defined one-step transition rules from a process to a process vector. In this section, we will present multi-step transitions.

Definition 3.8 [Multi-processes] The class $\mathbb{MP}$ of *multi-processes* is the least class inductively defined as follows:

(i) $\varepsilon \in \mathbb{MP}$;

(ii) if $P \in \mathbb{P}$, then $P \in \mathbb{MP}$;

(iii) if $\hat{P}_1,\dots,\hat{P}_n \in \mathbb{MP}$ and $n \ge 1$, then $(\hat{P}_1,\dots,\hat{P}_n) \in \mathbb{MP}$.

When a multi-process $\hat{P}$ is constructed only from ε, we call it an *ε-multi-process*. The examples of ε-multi-processes are ε, (ε), $(\varepsilon, (\varepsilon, \varepsilon), (\varepsilon, (\varepsilon)))$.

In Section 2, we introduced the set $T(\mathcal{F})$ of terms over the ranked alphabet $\mathcal{F}$. To express multi-step transitions, we introduce moreover a particular set $\mathcal{K}$ of constants to indicate the positions in trees (terms) constructed over actions, where substitutions take place. These new constants are denoted by $\Box_1, \Box_2, \ldots$. We consider trees constructed on $\mathcal{F} \cup \mathcal{K}$.

We use $\rightarrow_*$ to express multi-step transitions from a process to a multi-process, the transitions are exactly those which can be inferred from the rules below, together with α-conversion:

$$(1)\quad \frac{P \xrightarrow{f} (P_1, \ldots, P_n)}{P \xrightarrow{f(\Box_1,\ldots,\Box_n)}_* (P_1, \ldots, P_n)} \qquad \text{provided } f \in \mathcal{F}_n$$

$$(2)\quad \frac{P \xrightarrow{t}_* \hat{P}\{P_i/X\} \qquad P_i \xrightarrow{g(\Box_{i1},\ldots,\Box_{im})}_* (P_{i1}, \ldots, P_{im})}{P \xrightarrow{t\{g(\Box_{i1},\ldots,\Box_{im})/\Box_i\}}_* \hat{P}\{(P_{i1}, \ldots, P_{im})/X\}}$$

provided $g \in \mathcal{F}_m$, $t \in T(\mathcal{F} \cup \mathcal{K})$, $\Box_i$ occurs in t.

Notice that the assumption of rule (1) is defined in Definition 3.6 and, the basic idea of rule (2) is that $\Box_i$ and X have the same position respectively in t and $\hat{P}$.

Let $t \in T(\mathcal{F} \cup \mathcal{K})$. We call t a *path* from P to $\hat{P}$, if $P \xrightarrow{t}_* \hat{P}$.

Example 3.9 Consider three processes $P_1 = f.(P_2, P_3) + g.(P_2)$, $P_2 = g.(P_3) + a.\varepsilon$ and $P_3 = b.\varepsilon$, where $f \in \mathcal{F}_2$, $g \in \mathcal{F}_1$, $a, b \in \mathcal{F}_0$. By rules Sum1 and Prefix, we get $P_1 \xrightarrow{f} (P_2, P_3)$. By rule (1) above, we get $P_1 \xrightarrow{f(\Box_1,\Box_2)}_* (P_2, P_3)$. By rule (1) again, we have $P_2 \xrightarrow{g(\Box_{21})}_* (P_3)$. By rule (2), we get $P_1 \xrightarrow{f(g(\Box_{21}),\Box_2)}_* ((P_3), P_3)$. Moreover, $P_3 = b.\varepsilon$, then $P_3 \xrightarrow{b} \varepsilon$, and so $P_3 \xrightarrow{b}_* \varepsilon$. Applying rule (2) on $P_1 \xrightarrow{f(g(\Box_{21}),\Box_2)}_* ((P_3), P_3)$ and $P_3 \xrightarrow{b}_* \varepsilon$, we get $P_1 \xrightarrow{f(g(b),\Box_2)}_* ((\varepsilon), P_3)$. Apply rule (2) once more, we get $P_1 \xrightarrow{f(g(b),b)}_* ((\varepsilon), \varepsilon)$. So $f(g(b), b)$ is a path from P_1 to $((\varepsilon), \varepsilon)$.

4 Bisimulation

Similar to R. Milner's approach [9], we set up a notion of equivalence between tree processes by using the notion of bisimulation described as follows.

Definition 4.1 [Bisimulation] A binary relation $S \subseteq \mathbb{P} \times \mathbb{P}$ over tree processes is a *bisimulation* if $(P, Q) \in S$ implies, for all $f \in \mathcal{F}_n$ $(n \in N)$ and all $X \in \mathcal{V}$,

(i) Whenever $P \xrightarrow{f} (P_1, \ldots, P_n)$, then for some $Q_1, \ldots, Q_n$, $Q \xrightarrow{f} (Q_1, \ldots, Q_n)$ and $(P_i, Q_i) \in S$ $(i = 1, \ldots, n)$;

(ii) Whenever $Q \xrightarrow{f} (Q_1, \ldots, Q_n)$, then for some $P_1, \ldots, P_n$, $P \xrightarrow{f} (P_1, \ldots, P_n)$ and

$(P_i, Q_i) \in S$ $(i = 1, \ldots, n)$.

(iii) $P \rhd X$ iff $Q \rhd X$.

Proposition 4.2 *The following relations are bisimulations:*

1. $Id_{\mathbb{P}}$;
2. S^{-1}, *where* $S^{-1} = \{(y,x) \mid (x,y) \in S\}$, *provided* S *is a bisimulation;*
3. S_1S_2, *where* $S_1S_2 = \{(x,z) \mid$ *for some* y, $(x,y) \in S_1$ *and* $(y,z) \in S_2\}$, *provided* S_1, S_2 *are bisimulations;*
4. $\bigcup_{i \in I} S_i$, *where* I *is a set of indices, provided each* S_i *is a bisimulation.*

Proof. By the definition of bisimulation. □

We are now ready to define the equivalence.

Definition 4.3 P and Q are called *bisimilar*, written $P \sim Q$, if $(P,Q) \in S$ for some bisimulation S. This may be equivalently expressed as follow:

$$\sim \; = \bigcup \{S \mid S \text{ is a bisimulation}\}$$

It follows straightforward from Proposition 4.2 that

Proposition 4.4

(i) $\sim$ *is the largest bisimulation.*

(ii) $\sim$ *is an equivalence relation.*

Proof. It follows straightforward from Proposition 4.2. □

Now let us prove the main result of this section.

Proposition 4.5 $P \sim Q$ *if and only if, for every* $f \in \mathcal{F}_n$ $(n \in N)$ *and every* $X \in \mathcal{V}$,

(i) Whenever $P \xrightarrow{f} (P_1, \ldots, P_n)$, *then for some* $Q_1, \ldots, Q_n$, *we have* $Q \xrightarrow{f} (Q_1, \ldots, Q_n)$ *and* $P_i \sim Q_i$ $(i = 1, \ldots, n)$;

(ii) Whenever $Q \xrightarrow{f} (Q_1, \ldots, Q_n)$, *then for some* $P_1, \ldots, P_n$, *we have* $P \xrightarrow{f} (P_1, \ldots, P_n)$ *and* $P_i \sim Q_i$ $(i = 1, \ldots, n)$;

(iii) $P \rhd X$ *iff* $Q \rhd X$.

To show this proposition, we define a new relation $\sim'$ in terms of $\sim$ as follows:

Definition 4.6 $P \sim' Q$ if and only if, for every $f \in \mathcal{F}_n$ $(n \in N)$ and every $X \in \mathcal{V}$,

(i) Whenever $P \xrightarrow{f} (P_1, \ldots, P_n)$, then for some $Q_1, \ldots, Q_n$, we have $Q \xrightarrow{f} (Q_1, \ldots, Q_n)$ and $P_i \sim Q_i$ $(i = 1, \ldots, n)$;

(ii) Whenever $Q \xrightarrow{f} (Q_1, \ldots, Q_n)$, then for some $P_1, \ldots, P_n$, we have $P \xrightarrow{f} (P_1, \ldots, P_n)$ and $P_i \sim Q_i$ $(i = 1, \ldots, n)$;

(iii) $P \rhd X$ iff $Q \rhd X$.

Proposition 4.4(1) ensures that $\sim$ is a bisimulation. By Definitions 4.3 and 4.6, we have

$$P \sim Q \text{ implies } \; P \sim' Q \qquad (*)$$

It remains to show that $P \sim' Q$ implies $P \sim Q$, which follows from the lemma below

Lemma 4.7 *The relation $\sim'$ is a bisimulation.*

5 The Theory of Bisimulation

First we complete the syntax in Section 3.2 by establishing an inference system. Then we prove the soundness and completeness of the system w.r.t. the equivalent relation $\sim$.

5.1 *TPC: An Inference System for Tree Process*

TPC is formed by two sets of rules: the basic rules and the rules for recursion.

Basic Rules $\mathcal{A}$

A1 $P + Q = Q + P$

A2 $P + (Q + R) = (P + Q) + R$

A3 $P + P = P$

A4 if $P = P'$ and $Q = Q'$, then $Q\{P/X\} = Q'\{P'/X\}$

Rules for Recursion $\mathcal{R}$

R1 $\mu X.P = P\{\mu X.P/X\}$

R2 If $Q = P\{Q/X\}$ then $Q = \mu X.P$, provided that X is guarded in P

R3 $\mu X.(P + X) = \mu X.P$

R4 If $P = Q$ then $\mu X.P = \mu X.Q$

Notice that we need no explicit rules for prefix $f.\overrightarrow{P}$. In particular, the distributive law $f.(\overrightarrow{P} + \overrightarrow{Q}) = f.\overrightarrow{P} + f.\overrightarrow{Q}$ is not valid.

Notations. We write $\vdash P = Q$, if $P = Q$ can be deduced by the rules of $\mathcal{A}$ and $\mathcal{R}$; and write $\mathcal{A} \vdash P = Q$, if $P = Q$ can be deduced by the rules of $\mathcal{A}$.

5.2 *Soundness and Completeness*

The main result of the section is that the "truth" and "derivability" coincide. To be precise, for any tree processes P, Q, we have $\vdash P = Q$ if and only if $P \sim Q$.

Notation. Let S be a binary relation. PSQ denotes $(P, Q) \in S$; and $P \sim S \sim Q$ denotes that, for some P' and Q', we have $P \sim P'$, $P'SQ'$ and $Q' \sim Q$, where $\sim S \sim$ denotes the composition of binary relations under consideration.

Definition 5.1 [Bisimulation up to $\sim$] Let S be a binary relation. S is said to be a *bisimulation up to* $\sim$ if PSQ implies that, for every $f \in \mathcal{F}_n$ $(n \in N)$ and every $X \in \mathcal{V}$,

(i) Whenever $P \xrightarrow{f} (P_1, \ldots, P_n)$, then for some $Q_1, \ldots, Q_n$, one has $Q \xrightarrow{f} (Q_1, \ldots, Q_n)$ and $P_i \sim S \sim Q_i$, for any $i \in \{1, \ldots, n\}$;

(ii) Whenever $Q \xrightarrow{f} (Q_1, \ldots, Q_n)$, then for some $P_1, \ldots, P_n$, one has $P \xrightarrow{f} (P_1, \ldots, P_n)$ and $P_i \sim S \sim Q_i$, for any $i \in \{1, \ldots, n\}$;

(iii) $P \rhd X$ iff $Q \rhd X$.

This notion is used to show that two processes are bisimilar. More precisely, to prove $P \sim Q$, it suffices to find a bisimulation up to $\sim$ which contains (P, Q). This result stats in Proposition 5.3, which is in turn a short step from the following lemma.

Lemma 5.2 *If S is a bisimulation up to $\sim$, then $\sim S \sim$ is a bisimulation.*

Proof. By the definition of bisimulation and that of bisimulation up to $\sim$. □

Proposition 5.3 *If S is a bisimulation up to $\sim$ then $S \subseteq \sim$.*

Proof. By Lemma 5.2, $\sim S \sim$ is a bisimulation. Then it follows from the definition of $\sim$ that $\sim S \sim \subseteq \sim$. On the other hand, $Id_{\mathbb{P}} \subseteq \sim$, so $S \subseteq \sim S \sim$, and we are done. □

Proposition 5.3 and Lemma 5.4 below are useful in the proof of Propositions 5.5 and 5.6, which in turn server to show the soundness of the system.

Lemma 5.4 *If $P \sim Q$ and X is free in P and Q, then $P\{H/X\} \sim Q\{H/X\}$.*

Proof. Notice first that $P \sim Q$ ensures either X is guarded in both P and Q or in none of them. Now let $S = \{(P\{H/X\}, Q\{H/X\}) \mid P \sim Q\}$. It suffices to show that S is a bisimulation, which is proved inductively by using the definition of bisimulation and Proposition 3.7. □

Proposition 5.5 *If $P \sim Q$ and $P_i \sim Q_i$ $(i = 1, \ldots, n)$, then the followings hold:*

1. *$f.(P_1, \ldots, P_n) \sim f.(Q_1, \ldots .Q_n)$, for any $f \in \mathcal{F}_n$ $(n \in N)$;*
2. *$P_1 + \cdots + P_n \sim Q_1 + \cdots + Q_n$;*
3. *$\mu X.P \sim \mu X.Q$.*

Proof. 1. For any fixed $i \in \{1, \ldots, n\}$, since $P_i \sim Q_i$, there exists some bisimulation S_i, such that $P_i S_i Q_i$. It is easy to show that $\bigcup_{i=1}^{n} S_i \cup \{(f.(P_1, \ldots, P_n), f.(Q_1, \ldots .Q_n))\}$ is still a bisimulation, and hence $f.(P_1, \ldots, P_n) \sim f.(Q_1, \ldots .Q_n)$.

2. A similar proof as in Case 1 shows the desired result.

3. Let $S = \{(H\{\mu X.P/X\}, H\{\mu X.Q/X\}) \mid H \in \mathbb{P}\}$. When $H \equiv X$, we obtain $(\mu X.P, \mu X.Q) \in S$. So it suffices to show that S is a bisimulation. By Proposition

5.3, we only need to show that S is a bisimulation up to $\sim$, that is, we need to show

- For any $f \in \mathcal{F}_m$ ($m \in N$), whenever $H\{\mu X.P/X\} \xrightarrow{f} (P_1, \ldots, P_m)$, there exist some $Q_1, \ldots, Q_m$, such that $H\{\mu X.Q/X\} \xrightarrow{f} (Q_1, \ldots, Q_m)$ and that $P_i \sim S \sim Q_i$ for any $i \in \{1, \ldots, m\}$.
- Conversely.
- For any $Y \in \mathcal{V}$, $P \rhd Y$ iff $Q \rhd Y$.

We consider only the first case. the result is proved by induction on the depth of the inference to get $H\{\mu X.P/X\} \xrightarrow{f} (P_1, \ldots, P_m)$. The details are argued by cases on the form of H. □

Proposition 5.6

1. *$\mu X.P \sim P\{\mu X.P/X\}$.*
2. *If $Q \sim P\{Q/X\}$, then $Q \sim \mu X.P$, provided that X is guarded in P.*
3. *$\mu X.(P + X) \sim \mu X.P$.*

Proof. 1. By definition of $\sim$, we need to consider the cases below:

- if $\mu X.P \xrightarrow{f} \overrightarrow{P}$, then we have $P\{\mu X.P/X\} \xrightarrow{f} \overrightarrow{P}$ (only REC refers to $\mu X.P$);
- if $P\{\mu X.P/X\} \xrightarrow{f} \overrightarrow{P}$, then by REC we have $\mu X.P \xrightarrow{f} \overrightarrow{P}$;
- for any $Y \in \mathcal{V}$, $\mu X.P \rhd Y$ iff $P\{\mu X.P/X\} \rhd Y$.

So $\mu X.P \sim P\{\mu X.P/X\}$, as desired.

2. Let $S = \{(H\{Q/X\}, H\{\mu X.P/X\}) \mid H \in \mathbb{P}\}$. We need to show that S a bisimulation up to $\sim$. This is done by a similar proof as that for Proposition 5.5(3).

3. Let $S = \{(H\{\mu X.(P{+}X)/X\}, H\{\mu X.P/X\}) \mid H \in \mathbb{P}\}$. We need to show that S is a bisimulation up to $\sim$. This is done by a similar proof as that for Proposition 5.5(3). □

Theorem 5.7 (Soundness) *If $\vdash P = Q$, then $P \sim Q$.*

Proof. It is shown by induction on the depth of the inference. If the last rule used is one of $A1-A3$, we apply bisimulation definition. If it is $A4$, we apply Proposition 5.5. If it is one of $R1 - R4$, we apply Proposition 5.6 and Proposition 5.5(3). □

Now let us switch to the proof of completeness. The lemma below will be useful to show the unique solution of equations.

Lemma 5.8 *Let $P, Q, R \in \mathbb{P}$ and $X, Y \in \mathcal{V}$. We have*

(i) *If X is not free in P, then $\vdash P\{Q/X\} = P$.*

(ii) *If X and Y are distinct and there is no free occurrences of X in R, then*

$$\vdash P\{Q/X\}\{R/Y\} = P\{Q\{R/Y\}/X, R/Y\}$$

Proof. By induction on the structure of P. □

Proposition 5.9 (Unique solution of equations) *Let X_i, Y_j $(1 \leq i \leq m, 1 \leq j \leq n)$ be distinct variables, and $\tilde{X} = \{X_1, \ldots, X_m\}$, $\tilde{Y} = \{Y_1, \ldots, Y_n\}$. Let Q_i $(i = 1, \ldots, m)$ are process expressions with free variables in $\tilde{X} \cup \tilde{Y}$, in which each X_i is guarded. Then there exists a set of expressions $\tilde{P} = \{P_1, \ldots, P_m\}$ with free variables in $\tilde{Y}$ such that $\vdash P_i = Q_i\{\tilde{P}/\tilde{X}\}$ $(i = 1, \ldots, m)$. Moreover, if there is $\tilde{P'} = \{P'_1, \ldots, P'_m\}$ with free variables in $\tilde{Y}$ s.t. $\vdash P'_i = Q_i\{\tilde{P'}/\tilde{X}\}$ $(i = 1, \ldots, m)$, then $\vdash P'_i = P_i (i = 1, \ldots, m)$.*

Proof. It is proved by induction on m and using Lemma 5.8. □

The next proposition states that every expression provably satisfies a set of equations.

Proposition 5.10 (Equational characterization) *For every tree process expression P with free variables in $\tilde{Y} = \{Y_1, \ldots, Y_n\}$, there exist tree process expressions $P_1, \ldots, P_p$ $(p \geqslant 1)$ with free variables in $\tilde{Y}$, satisfying the following equations*

$$\vdash P = P_1$$
$$\vdash P_i = \sum_{j=1}^{m(i)} f_{ij}.\overrightarrow{P}_{ij} + \sum_{k=1}^{n(i)} Y_{g(i,k)} \quad (i = 1, \ldots, p)$$

where $m(i), n(i), g(i,k) \in N$, $n(i), g(i,k) \leqslant n$, $f_{ij} \in \mathcal{F}$ and each $\overrightarrow{P}_{ij}$ is an expression-vector whose components are in $\{P_1, \ldots, P_p\}$.

Proof. By induction on the structure of P. □

Theorem 5.11 (Completeness) *If $P \sim P'$, then $\vdash P = P'$.*

Proof. Suppose P and P' have free variables in $\{Y_1, \ldots, Y_n\}$. By Proposition 5.10, there are provable equations $\vdash P = P_1$, $\vdash P' = P'_1$ and

$$\vdash P_i = \sum_{j=1}^{m(i)} f_{ij}.\overrightarrow{P}_{ij} + \sum_{k=1}^{n(i)} Y_{g(i,k)} \ (i = 1, \ldots, p)$$
$$\vdash P'_i = \sum_{j=1}^{m'(i)} f'_{ij}.\overrightarrow{P'}_{ij} + \sum_{k=1}^{n'(i)} Y_{g'(i,k)} \ (i = 1, \ldots, p').$$

For any fixed i, j, we may suppose without loss of generality that

$$\overrightarrow{P}_{ij} = (P_{h(i,j,1)}, \ldots, P_{h(i,j,l)})$$
$$\overrightarrow{P'}_{ij} = (P'_{h'(i,j,1)}, \ldots, P'_{h'(i,j,l')})$$

where $l = Arity(f_{ij}), l' = Arity(f'_{ij})$ are as defined in Preliminaries.

Let $I = \{\langle i, i' \rangle \mid P_i \sim P'_{i'}\}$. By $\vdash P = P_1$, $\vdash P' = P'_1$ and Theorem 5.7, we have $\langle 1, 1 \rangle \in I$. Moreover, for each $\langle i, i' \rangle \in I$, we have $P_i \sim P'_{i'}$ and hence

(i) There exists a total surjective relation $J_{ii'}$ between $\{1,\ldots,m(i)\}$ and $\{1,\ldots,m'(i')\}$:

$$J_{ii'} = \{\langle j,j'\rangle \mid f_{ij} = f'_{i'j'} \text{ and } \langle h(i,j,k), h'(i',j',k')\rangle \in I\}$$

where $k \in \{1,\ldots,Arity(f_{ij})\}$ and $k' \in \{1,\ldots,Arity(f'_{i'j'})\}$;

(ii) $\vdash \sum_{j=1}^{n(i)} Y_{g(i,j)} = \sum_{j=1}^{n'(i')} Y_{g'(i',j)}$.

Now let us consider the formal equations, one for each $\langle i,i'\rangle \in I$:

$$X_{ii'} = \sum_{\langle j,j'\rangle \in J_{ii'}} f_{ij}.\overrightarrow{X}_{ij} + \sum_{j=1}^{n(i)} Y_{g(i,j)}$$

where $\overrightarrow{X}_{ij} = (X_{h(i,j,1)h'(i',j',1)},\ldots,X_{h(i,j,Arity(f_{ij}))h'(i',j',Arity(f_{ij}))})$, and the $X_{ii'}$ are not in $\{Y_1,\ldots,Y_n\}$.

First notice that they are provably satisfied when each $X_{ii'}$ is instantiated to P_i. This is because the typical equation becomes

$$P_i = \sum_{\langle j,j'\rangle \in J_{ii'}} f_{ij}.\overrightarrow{P}_{ij} + \sum_{j=1}^{n(i)} Y_{g(i,j)}$$

and is provable since its right-hand side differs at most by repeated summands from that of the already proved equation for P_i, the latter depends on the totality of $J_{ii'}$. Second, the formal equations are provably satisfied when each $X_{ii'}$ is instantiated to $P'_{i'}$. This follows from the surjectivity of $J_{ii'}$. Finally, notice that each $X_{ii'}$ is guarded in the right-hand sides of the formal equations. Therefore it follows straightforward from Proposition 5.9 that $\vdash P_i = P'_{i'}$ for each $\langle i,i'\rangle \in I$, and hence $\vdash P = P'$. □

6 Application

In this section, we will provide a method to decide the equivalence via bisimulation over tree processes. As a corollary, we obtain a new equivalence relation between tree automata, which is strictly coarser than the equivalence relation via bisimulation defined in [1], and is strictly finer than the classical equivalence over tree automata, that is the equivalence via language equality of tree automata.

6.1 TLTS with Initial States

In Section 3.1 we have mentioned that a TLTS can be thought of as a tree automaton without initial states or final states. Now we study TLTSs with initial states and their languages.

In the following, we denote the tree processes by lowercase $p, q, \ldots$.

Definition 6.1 [TLTS with initial states] Let $\mathcal{F}$ be a set of ranked symbols. A *TLTS with initial states* (TLTSI) over $\mathcal{F}$ is a quartuple $A = (Q, \mathcal{F}, Q_I, \Delta)$, where $(Q, \mathcal{F}, \Delta)$ is a TLTS, Q_I ($\subseteq Q \subset_{fin} \mathbb{P}$) is a set of initial states. *The language of a TLTSI A is*

$$L(A) = \{t \mid p \xrightarrow{t}_* \hat{p},\ p \in Q_I, t \in T(\mathcal{F})\}$$

where $\hat{p}$ is a ϵ-mult-process and $\longrightarrow_*$ is a multi-step transition determined by Δ.

6.2 TLTSIs vs. Top-down Tree Automata

Given a TLTSI $A = (Q, \mathcal{F}, Q_I, \Delta)$, we will build a top-down tree automaton which recognizes $L(A)$. Actually, we define an automata $A' = (Q', \mathcal{F}, Q'_I, \Delta')$ by choosing

(1) $Q' = Q$; (2) $Q'_I = Q_I$; (3) $q \to f(q_1, \ldots, q_n) \in \Delta'$ iff $q \xrightarrow{f} (q_1, \ldots, q_n) \in \Delta$ where $f \in \mathcal{F}_n$, $q, q_1, \ldots, q_n \in Q$.

A standard proof by induction on derivation length yields $L(A') = L(A)$. Thus, $L(A)$ is recognized by the top-down tree automaton A'.

Conversely, suppose L is the tree language recognized by a top-down tree automaton $A' = (Q', \mathcal{F}, Q'_I, \Delta')$, that is, $L = L(A')$. We define a TLTSI $A = (Q, \mathcal{F}, Q_I, \Delta)$ by

(1) $Q = Q'$; (2) $Q_I = Q'_I$; (3) $q \xrightarrow{f} (q_1, \ldots, q_n) \in \Delta$ iff $q \to f(q_1, \ldots, q_n) \in \Delta'$ where $f \in \mathcal{F}_n$, $q, q_1, \ldots, q_n \in Q$.

A standard proof by induction on derivation length yields $L(A) = L(A')$. Combining this with what we have proved above, we get the following theorem:

Theorem 6.2 *A tree language is recognized by a top-down tree automata if and only if it is a language of TLTSI.*

6.3 A Deciding Method

In the previous section, we have shown that for a given top-down tree automaton $A' = (Q', \mathcal{F}, Q'_I, \Delta')$, there exists a TLTSI $A = (Q, \mathcal{F}, Q_I, \Delta)$ such that $Q = Q'$, $Q_I = Q'_I$ and $L(A) = L(A')$. Such a A will be called the *generated TLTSI* of A'.

Definition 6.3 Let $A_i = (Q_i, \mathcal{F}, Q_{Ii}, \Delta_i)$ $(i = 1, 2)$ be TLTSIs. A_1 and A_2 are said to be *equivalent* (written $A_1 \doteq A_2$), if the following conditions hold:

(i) for any $q_1 \in Q_{I1}$, there exists some $q_2 \in Q_{I2}$ such that $q_2 \sim q_1$,

(ii) for any $q_2 \in Q_{I2}$, there exists some $q_1 \in Q_{I1}$ such that $q_1 \sim q_2$,

where $\sim$ is the equivalence relation defined in Definition 4.3.

Clearly, the relation $\doteq$ defined above is an equivalence relation .

Lemma 6.4 *Let $A_1 = (Q_1, \mathcal{F}, Q_{I1}, \Delta_1)$ and $A_2 = (Q_2, \mathcal{F}, Q_{I2}, \Delta_2)$ be TLTSIs. If $A_1 \doteq A_2$, then $L(A_1) = L(A_2)$.*

Based on this Lemma 6.4, we get immediately the following theorem:

Theorem 6.5 *Let $A'_i = (Q'_i, \mathcal{F}, Q'_{Ii}, \Delta'_i)$ $(i = 1, 2)$ be two top-down tree automata. Let $A_i = (Q_i, \mathcal{F}, Q_{Ii}, \Delta_i)$ $(i = 1, 2)$ are their generated TLTSIs respectively. If $A_1 \doteq A_2$, then $L(A'_1) = L(A'_2)$.*

This theorem provides a method of deciding equivalence on tree automata (in the classical sense) based on bisimulation between processes of tree process calculus. But the example below shows that the converse does not hold.

Example 6.6 Let $\mathcal{F} = \{f, g, a\}$, $f \in \mathcal{F}_2$, $g \in \mathcal{F}_1$, $a \in \mathcal{F}_0$. Let $A' = (Q'_1, \mathcal{F}, Q'_{I1}, \Delta'_1)$, $B' = (Q'_2, \mathcal{F}, Q'_{I2}, \Delta'_2)$ be two tree automata, where

$Q'_1 = \{q, q_g, q_f\}$, $Q'_{I1} = \{q_f\}$, $Q'_2 = \{q', q'_g, q'_f, q'_m\}$, $Q'_{I2} = \{q'_f\}$,

$\Delta'_1 = \{q \rightarrow a,\ q \rightarrow g(q),\ q_g \rightarrow g(q),\ q_f \rightarrow g(q_g),\ q \rightarrow f(q, q)\}$,

$\Delta'_2 = \{q' \rightarrow a,\ q' \rightarrow g(q'),\ q'_g \rightarrow g(q'),\ q'_f \rightarrow g(q'_g),\ q' \rightarrow f(q', q'),\ q'_f \rightarrow g(q'_m)\}$.

Clearly $L(A') = L(B')$. We show now $(A', B') \notin \doteq$:

Let $A = (Q_1, \mathcal{F}, Q_{I1}, \Delta_1)$, $B = (Q_2, \mathcal{F}, Q_{I2}, \Delta_2)$ be the generated TLTSIs of A' and B', respectively. We have

$Q_1 = \{q, q_g, q_f\}$, $Q_{I1} = \{q_f\}$, $Q_2 = \{q', q'_g, q'_f, q'_m\}$, $Q_{I2} = \{q'_f\}$,

$\Delta_1 = \{q \xrightarrow{a} \varepsilon,\ q \xrightarrow{g} q,\ q_g \xrightarrow{g} q,\ q_f \xrightarrow{g} q_g,\ q \xrightarrow{f} (q, q)\}$,

$\Delta_2 = \{q' \xrightarrow{a} \varepsilon,\ q' \xrightarrow{g} q',\ q'_g \xrightarrow{g} q',\ q'_f \xrightarrow{g} q'_g,\ q' \xrightarrow{f} (q', q'),\ q'_f \xrightarrow{g} q'_m\}$.

So $q = a.\varepsilon + g.(q) + f.(q, q)$, $q_g = g.(q)$, $q_f = g.(q_g)$ and $q' = a.\varepsilon + g.(q') + f.(q', q')$, $q'_g = g.(q')$, $q'_f = g.(q'_g) + g.(q'_m)$. Obviously, $q_f \nsim q'_f$, then $(A', B') \notin \doteq$.

6.4 Three Equivalences on Tree Automata

In the following, we denote by NTA the set of non-deterministic finite tree automata.

Definition 6.7 Two automata A_1 and A_2 are said to be equivalent (written as $A_1 \doteq A_2$ by abuse of language), if their generated TLTSIs are equivalent.

Now there are actually three equivalence relations defined on NTA: one is the bisimulation equivalent introduced by Parosh Aziz Abdulla et al. in [1]; one is given in Definition 6.7; and one is the language equivalent in the sense of automata theory. In this section, we will study the relation between these equivalences.

The definition of bisimulation equivalence [1] determined by a notion of bisimulation on bottom-up tree automata. We translate it to the counterpart on top-down tree automata:

Definition 6.8 [Bisimulation equivalence on tree automata] Let $A'_1 = (Q'_1, \mathcal{F}, Q'_{I1}, \Delta'_1)$ and $A'_2 = (Q'_2, \mathcal{F}, Q'_{I2}, \Delta'_2)$ be both top-down tree automata. A relation $\simeq \subseteq Q'_1 \times Q'_2$ is said to be a bisimulation relation if the following two conditions hold for all states $q \in Q'_1$ and $q' \in Q'_2$ such that $q \simeq q'$. (1) $q \in Q'_{I1}$ if and only if $q' \in Q'_{I2}$; (2) the fact that $q_k \rightarrow f(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_{k-1}) \in \Delta'_1$, where $i \leq k, f \in \mathcal{F}_{k-1}$, implies that there exists a rule $q'_k \rightarrow f(q'_1, \ldots, q'_{i-1}, q', q'_{i+1}, \ldots, q'_{k-1}) \in \Delta'_2$, such that $q_j \simeq q'_j$ for all $j \in \{1, \ldots, k\}$, and vice versa.

States q and q' as above are said to be *bisimular* (with respect to $\simeq$). We consider A'_1 and A'_2 to be *bisimulation equivalent* if there is a bisimulation relation such that every state in Q'_1 is bisimilar to a state in Q'_2, and vice versa.

Let us display explicitly the three equivalence relations defined on NTA:

$$\approx\ = \{(A,B) \in NTA \times NTA \mid A \text{ and } B \text{ are bisimulation equivalent}\};$$

$$\doteq\ = \{(A,B) \in NTA \times NTA \mid A \doteq B\};$$

$$\cong\ = \{(A,B) \in NTA \times NTA \mid L(A) = L(B)\}.$$

By Theorem 6.5 and Example 6.6, we get immediately that $\doteq\ \subset\ \cong$. To study the relation between $\approx$ and $\doteq$, we introduce a new relation $\approx' \subseteq NTA \times NTA$ as follows.

Definition 6.9 Let A', B' be top-down tree automata and their generated TLTSIs be $A = (Q_1, \mathcal{F}, Q_{I1}, \Delta_1)$ and $B = (Q_2, \mathcal{F}, Q_{I2}, \Delta_2)$, respectively. $A' \approx' B'$ if the following conditions hold:

(i) for any $q_1 \in Q_{I1}$, there exists some $q_2 \in Q_{I2}$ such that $q_2 \sim q_1$;

(ii) for any $q_2 \in Q_{I2}$, there exists some $q_1 \in Q_{I1}$ such that $q_1 \sim q_2$;

(iii) for any $q_1 \in Q_1$, there exists some $q_2 \in Q_2$ such that $q_2 \sim q_1$;

(iv) for any $q_2 \in Q_2$, there exists some $q_1 \in Q_1$ such that $q_1 \sim q_2$.

By Definitions 6.3, 6.7, 6.8 and 6.9, we can prove easily that $\approx\ \subseteq\ \approx'$ and $\approx'\ \subseteq\ \doteq$. So $\approx\ \subseteq\ \doteq$. However the example below shows $\approx'\ \neq\ \doteq$. So $\approx'\ \subset\ \doteq$, and hence $\approx\ \subset\ \doteq$.

Example 6.10 Let $\mathcal{F} = \{f, g, h, a, \}$, $f \in \mathcal{F}_2$, $g, h \in \mathcal{F}_1$ and $a \in \mathcal{F}_0$. Let $A' = (Q'_1, \mathcal{F}, Q'_{I1}, \Delta'_1)$, $B' = (Q'_2, \mathcal{F}, Q'_{I2}, \Delta'_2)$ be two tree automata, where

$Q'_1 = \{q, q_g, q_f\}$, $Q'_{I1} = \{q_f\}$, $Q'_2 = \{q', q'_g, q'_f, q'_m\}$, $Q'_{I2} = \{q'_f\}$,

$\Delta'_1 = \{q \to a,\ q \to g(q),\ q_g \to g(q),\ q_f \to g(q_g),\ q \to f(q, q)\}$,

$\Delta'_2 = \{q' \to a,\ q' \to g(q'),\ q'_g \to g(q'),\ q'_f \to g(q'_g),\ q' \to f(q', q'),\ q'_m \to h(q')\}$.

Clearly $(A', B') \notin\ \approx'$, because there is no such state q_m in Q'_1 which satisfies $q_m \simeq q'_m$.

We show now $(A', B') \in\ \doteq$. Let $A = (Q_1, \mathcal{F}, Q_{I1}, \Delta_1)$, $B = (Q_2, \mathcal{F}, Q_{I2}, \Delta_2)$ be the generated TLTSIs respectively of A' and B'. Hence

$Q_1 = \{q, q_g, q_f\}$, $Q_{I1} = \{q_f\}$, $Q_2 = \{q', q'_g, q'_f, q'_m\}$, $Q_{I2} = \{q'_f\}$,

$\Delta_1 = \{q \xrightarrow{a} \varepsilon,\ q \xrightarrow{g} q,\ q_g \xrightarrow{g} q,\ q_f \xrightarrow{g} q_g,\ q \xrightarrow{f} (q, q)\}$,

$\Delta_2 = \{q' \xrightarrow{a} \varepsilon,\ q' \xrightarrow{g} q',\ q'_g \xrightarrow{g} q',\ q'_f \xrightarrow{g} q'_g,\ q' \xrightarrow{f} (q', q'),\ q'_m \xrightarrow{h} q'\}$.

So $q = a.\varepsilon + g.(q) + f.(q, q)$, $q_g = g.(q)$, $q_f = g.(q_g)$ and $q' = a.\varepsilon + g.(q') + f.(q', q')$, $q'_g = g.(q')$, $q'_f = g.(q'_g)$, $q'_m = h.(q')$. Obviously, $q_f \sim q'_f$, then $(A', B') \in\ \doteq$.

We get now the main result of the section.

Theorem 6.11 $\approx\ \subset\ \doteq\ \subset\ \cong$.

7 Conclusion

We have provided a generalization of Milner's NSPC from the context of word automata to that of tree automata. By this way, we established a sound and complete inference system TPC which describes the equivalence via bisimulation of tree processes. As a by-product, we obtained a new equivalence relation on tree automata, which is strictly finer than the language equivalence, and is strictly coarser than the bisimulation equivalence defined in [1]. This equivalence relation may give smaller automata during minimization. TPC seems providing a new approach for modeling broadcast-based communication, but it needs more exploration through examples, as well as the comparison with broadcast-based calculi like CBS [12,5], MBS [13], HOBS [11] and πb calculus [3]. Another extension of TPC, which is an ongoing work, is to consider communication between tree processes.

Acknowledgement

This work is partially supported by NSFC 60673045, NSFC major research program 60496321 and NSFC 60721061.

References

[1] Abdulla P.-A., L. Kaati, and J, Högberg,*Bisimulation Minimization of Tree Automata*, CIAA(2006), 171-185.

[2] Comon H., M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, "Tree Automata: Techniques and Applications," 1997.

[3] Ene C. and T. Muntean, *A broadcast-based calculus for communicating systems*, IPDPS(2001), 149pp.

[4] Gécseg F. and M. Steinby, "Tree Automata," Akadémiai kiadó, Budapest , 1984.

[5] Hennnessy M. and J. Rathke, *Bisimulations for a calculus of broadcasting systems* , TCS **200(1-2)**(1998), 225-260.

[6] Hopcroft J.-E., R. Motwani, and J.-D. Ullman, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, New York, 2nd Edition, 2001.

[7] Milner R., *A Complete Inference System for a Class of Regular Behaviours*, JCSS **28(3)**(1984), 439-466.

[8] Milner R., *A Complete Axiomatisation for Observational Congruence of Finite-State Behaviours*, I&C **81(2)**(1989), 227-247.

[9] Milner R., "Communication and Concurrency," Prentice-Hall, 1989.

[10] Milner R., "Communicating and Mobile Systems: the π-Calculus," Cambridge University Press, 1999.

[11] Ostrovsky K. , K.V.S. Prasad, and W. Taha, *Towards a primitive higher order calculus of broadcasting systems*, PPDP(2002), 2-13.

[12] Prasad K.V.S., *A calculus of broadcasting systems*, SCP **25(2)**(1995), 285-327.

[13] Prasad K.V.S., *A prospectus for mobile broadcasting systems*, ENTCS **162** (2006), 295-300.