A Congruence Theorem for Structured Operational Semantics of Higher-Order Languages

Karen L. Bernstein
School of Computer Science, Telecommunications, and Information Systems
DePaul University, Chicago, IL 60604 USA
kbernstein@cs.depaul.edu

Abstract

In this paper we describe the promoted tyft/tyxt *rule format for defining higher-order languages. The rule format is a generalization of Groote and Vaandrager's* tyft/tyxt *format in which terms are allowed as labels on transitions in rules. We prove that bisimulation is a congruence for any language defined in* promoted tyft/tyxt *format and demonstrate the usefulness of the rule format by presenting* promoted tyft/tyxt *definitions for the lazy λ-calculus, CHOCS and the π-calculus.*

$$\frac{}{fv(a) = \{a\}} \qquad \frac{fv(x) = S}{fv(\lambda a.x) = S - a} \qquad \frac{fv(x_1) = S_1 \quad fv(x_2) = S_2}{fv(x_1\ x_2) = S_1 \cup S_2}$$

$$\frac{fv(w) = \{\}}{a \xrightarrow{[w/a]} w} \qquad \frac{fv(w) = \{\}}{b \xrightarrow{[w/a]} b}\ (b \neq a) \qquad \frac{x_1 \xrightarrow{[w/a]} y_1 \quad x_2 \xrightarrow{[w/a]} y_2}{x_1\ x_2 \xrightarrow{[w/a]} y_1\ y_2}$$

$$\frac{fv(w) = \{\}}{\lambda a.x \xrightarrow{[w/a]} \lambda a.x} \qquad \frac{x \xrightarrow{[w/b]} y}{\lambda a.x \xrightarrow{[w/b]} \lambda a.y}\ (a \neq b)$$

Figure 1: Substitution rules for the λ-calculus

1 Introduction

For a programming language definition that uses bisimulation as the notion of equivalence, it is desirable for the bisimulation relation to be compatible with the language constructs; i.e. that bisimulation be a congruence. Several rule formats have been defined, so that as long as a definition satisfies certain syntactic constraints, then the defined bisimulation relation is guaranteed to be a congruence. However these rule formats have not been widely used for defining languages with higher-order features. In this paper we introduce *promoted tyft/tyxt*, a new rule format that is suitable for defining higher-order languages. We demonstrate the expressiveness of the rule format by giving *promoted tyft/tyxt* definitions for the lazy λ-calculus, CHOCS [Tho95] and the π-calculus [MPW92]. Although not presented here, the definition of a small strict functional language from [BS95] is also in *promoted tyft/tyxt* format. The main result of this paper is that for any language defined in *promoted tyft/tyxt* format, bisimulation is a congruence.

To make the existing definitions for the lazy λ-calculus, CHOCS and the π-calculus conform to the restrictions of the *promoted tyft/tyxt format*, we had to make three types of changes. First, a novel aspect of the definitions presented in this paper is that the entire programming language is defined by means of the transition system specification. In particular, we define syntactic substitution and free variable tests with *promoted tyft/tyxt* rules. As can be seen in Figure 1, these definitions are straightforward. The main difference is that they are now part of the language definition rather than the meta-theory. Second, our definitions always use bisimilarity as the notion of equivalence in contrast with the existing definitions which use other bisimulation relations, such as applicative bisimulation, higher order bisimulation, or open bisimulation. The third change is that in order to meet the syntactic constraints of the rule format, in several cases, we had to replace a single rule in the original definition with more than one rule in the *promoted tyft/tyxt* definition.

1.1 Related Work

In order to simplify the specification of concurrent languages, extensive work has been done in meta-theories for SOS definitions of process algebras, especially with regard to the congruence property for bisimulation. Rule formats such as de Simone's format [dS85], the *tyft/tyxt* by Groote and Vaandrager [GV92], and the GSOS rule format by Bloom, Istrael and Meyer [BIM95] guarantee that bisimulation is a congruence. Extensions of these rule formats for including negative premises and predicates have made the rule formats of greater practical use [Gro93, BV93, Ver95, BG96].

Abramsky was the first to use bisimulation as the notion of equivalence for a functional programming language [Abr90] and this style of definition is now often used for languages with higher-order features [San92, CG95, FHJ96, Jef95, Gor95, GR96]. These definitions typically prove that bisimulation is a congruence by using Howe's proof technique [How96].

The traditional rule formats for process algebras have not proven useful for higher-order languages and several rule formats have been developed specifically for higher-order languages. A primary characteristic that distinguishes the *promoted tyft/tyxt* from the other rule formats is that it does not include substitution as a part of the meta-theory. Howe defines structured evaluation systems for the evaluation of functional languages and proves that some generalizations of applicative bisimulation are a congruence [How96]. Sands defines the Globally Deterministic SOS rule format (GDSOS), which guarantees the congruence property for small-step evaluation in functional languages. He also establishes proof principles for reasoning about languages that can be expressed in this format [San97]. Weber and Bloom define rule format for defining languages similar to the π-calculus [WB96]. Sangiorgi defines a rule format that extends the *tyft/tyxt* rule format with λ-calculus evaluation rules [San94]. Howe showed that applicative bisimulation is a congruence for Sangiorgi's rule format [How95]. Fokkink and Verhoef define a rule format that allows the definition of a wide class of languages, but it guarantees conservativity rather than the congruence property [FV97].

1.2 Preliminaries

A *signature* consists of a set of function symbols along with a rank function that gives the arity for each function symbol. A function symbol with arity zero is called a *constant symbol*. The set of terms defined by a signature Σ, over a set V of variables, is denoted $T(\Sigma, V)$. The set $T(\Sigma, \emptyset)$ is abbreviated $T(\Sigma)$ and elements of the set are called *closed terms*. In general, terms from $T(\Sigma, V)$ are called *open* terms. A *substitution* ρ is a mapping from variables to terms. Substitution is extended to terms in the obvious way. We will use the notation $\rho[x \mapsto t]$ to mean the substitution that maps the variable x to the term t and otherwise behaves like ρ.

A *transition relation (over the set A)* is either a *labeled transition* $\rightarrow \subseteq A \times A \times A$, an *unlabeled transition* $\rightarrow \subseteq A \times A$ or a *predicate* $P \subseteq A$. We will often write $t \xrightarrow{s} t'$ for the labeled transition $\rightarrow (t, s, t')$, where t is called the *source of the transition*, s is called the *label of the transition*, and t' is called the *target of the transition*.

A *transition system specification (TSS)* is a structure (Σ, T, R) with Σ a signature, T a set of transition relations over terms $T(\Sigma, V)$, and R a set of deduction rules of the form: $\frac{H}{C}$, where H (the hypothesis) is a set of transitions in T over terms and C (the conclusion) is a single transition in T over terms. The definitions of "substitution", "Var" and "closed" extend to rules in the obvious way.

For $P = (\Sigma, T, R)$ a TSS, a *proof* from P of a transition ψ is a finite, upwardly branching tree whose nodes are labeled by transition formulas such that (1) the root is labeled with ψ, and (2) if χ is the label of a node q and $\{\chi_i \mid i \in I\}$ is the set of labels of the nodes directly above q, then there is a rule

$$\frac{\{\phi_i \mid i \in I\}}{\phi}$$

in R and a substitution $\rho : V \rightarrow T(\Sigma, V)$ such that $\chi = \rho(\phi)$ and $\chi_i = \rho(\phi_i)$ for $i \in I$. If a proof of ψ from P exists then we say that ψ is *provable* from P and a proof is *closed* if it contains only closed transitions.

A *transition system (TS)* is a structure (S, T) where: S is a set of *states* and T is a set of transition relations over S. We use (strong) bisimulation as in [Par81]. Let $\mathcal{A} = (S, T)$ be a transition system, and $R \subseteq S \times S$ a symmetric relation. R is a *(strong) bisimulation* if for all $s, s', t, a \in S$ it satisfies, for all labeled transitions $\rightarrow \in T$: $(s \ R \ t$ and $s \xrightarrow{a} s')$ implies $(\exists t' \in S, t \xrightarrow{a} t'$ and $s' \ R \ t')$, for all unlabeled transitions $\rightarrow \in T$: $(s \ R \ t$ and $s \longrightarrow s')$ implies $(\exists t' \in S, t \longrightarrow t'$ and $s' \ R \ t')$ and for all predicates $P \in T$: $(s \ R \ t$ and $P(s))$ implies $P(t)$. Two states $s, t \in S$ are *bisimilar* $(s \sim t)$, if there exists a bisimulation relating them.

For $P = (\Sigma, T, R)$ a TSS, the *transition system $TS(P)$ specified by P* is given by $TS(P) = (T(\Sigma), T_P)$, where for all $\rightarrow \in T$, (s, a, s') is in $\rightarrow$ $(\in T_P)$ if and only if there exists a proof from P of $s \xrightarrow{a} s'$ (similarly for unlabeled transitions and predicates).

Assume a set $\{t_i \xrightarrow{s_i} t'_i \mid i \in I\}$ of labeled transitions. The *dependency graph* for the labeled transitions is a directed graph, with a vertex for each labeled transition in the set and an edge from vertex i to vertex j if a variable that occurs t_i or s_i also occurs in t'_j. A set of transitions is *well-founded* if, for all the labeled transitions, any backward chain of edges in its dependency graph is finite. A transition rule is well-founded if its collection of premises is so, and a TSS is well-founded if all of its transition rules are well-founded.

2 The *Promoted tyft/tyxt* Rule Format

The *promoted tyft/tyxt* rule format is a generalization of the *tyft/tyxt* rule format [GV92] which allows terms as labels on transitions. If the labels on transitions are all constants, then definition reduces to Groote and Vaandrager's *tyft/tyxt* rule format.

We use $\vec{x}$, to stand for the vector of variables $x_1, .., x_n$ where the value of n should be clear from the context and could be zero. The notation $\vec{x}[i]$ refers to the ith element of the vector x. For a binary relation R, we will use $\vec{x}\ R\ \vec{y}$ to mean $\forall i,\ \vec{x}[i]\ R\ \vec{y}[i]$. Similarly, $\vec{x} \in X$ means $\forall i,\ \vec{x}[i] \in X$. $(x_i\ R\ y_i)^{i \in I}$ means for all i in I, $(x_i\ R\ y_i)$. $[x_i \mapsto e_i]^{i \in I}$ stands for the substitution that for each $i \in I$ maps x_i to e_i. Note that x and $\vec{x}$ denote unrelated variables.

Definition 2.1 *For $P = (\Sigma, T, R)$ a TSS, a rule is in* promoted tyft *format if it has the form:*

$$\frac{\{t_i \xrightarrow{g_i(\vec{t_i})}_i y_i \mid i \in I\}}{f(\vec{x}) \xrightarrow{g(\vec{w})} t}$$

with I an index set, f, g and g_i function symbols in Σ, $\vec{w} \in W$, $\vec{x} \in X$, $y_i \in Y$ all different variables, $t_i, \vec{t_i}, t \in T(\Sigma, V)$, and $\to, \to_i$ labeled transition relations in T. P is in promoted tyft format *if every rule in R is in* promoted tyft format.

Definition 2.2 *A rule is in* promoted tyft/tyxt *format if it is in* promoted tyft *format except: (1) the source of the conclusion may be a variable, (2) the label over the arrow in the conclusion may be a variable, (3) any labeled transition can be an unlabeled transition, (4) the conclusion may be a predicate of the form $P(f(\vec{x}))$, and (5) transitions in the hypothesis may be predicates of the form $P(t)$. A TSS is in* promoted tyft/tyxt format *if every rule is in* promoted tyft/tyxt format.

Proposition 2.1 *For any (well-founded) TSS in* promoted tyft/tyxt *format, there is a transition equivalent (well-founded) TSS in* promoted tyft *format.*

Proof: Let $P = (\Sigma, T, R)$ be a transition system specification in *promoted tyft/tyxt* format. We will define a transition equivalent (well-founded) TSS $P = (\Sigma', T', R')$. Let Σ' be Σ with a new constant symbol $\bullet$ and for each predicate P in T, include a new constant c_p. Let T' include all the labeled transition relations in T. We will define R' to be R with the following modifications. [Case 1] Replace every rule r in which the source of the conclusion is a variable with a rule for each function symbol $f \in \Sigma$, where each r_f is obtained by substituting $f(\vec{x})$ for x in the rule r. [Case 2] Replace every rule r in which the label over the arrow in the conclusion is a variable with a rule for each function symbol $f \in F$, where each r_f is obtained by substituting $f(\vec{x})$ for x in the rule r. [Case 3] In every rule with an unlabeled transition, replace each unlabeled transition $t \longrightarrow t'$ with the transition $t \xrightarrow{\bullet} t'$. [Cases 4&5] In every rule with a predicate, replace each predicate $P(t)$ with the transition $t \xrightarrow{c_p} \bullet$.

The TSS $P' = (\Sigma', R')$ is in *promoted tyft* format. If the original rules in R were well-founded then the rules in R' are also well-founded. Finally, for any proof of a transition $e \xrightarrow{a} e'$ in P there is a corresponding proof in P'. $\Box$

2.1 Counterexamples

The following examples demonstrate that the restrictions in our extensions to *tyft/tyxt* format are necessary in order for bisimulation to be congruence. Each example below is a transition system specification over the signature with constants 1 and 2, unary function symbol f, and binary function symbol g. For each condition in the definition, we give a set of rules so that 1 is bisimilar to 2 but there exists some context $C[\,]$ such that $C[1]$ is not bisimilar to $C[2]$.

The label over premise must have at least one function symbol. We have that $1 \sim 2$ but $f(1) \not\sim f(2)$.

$$\frac{}{1 \xrightarrow{1} 1} \qquad \frac{}{2 \xrightarrow{1} 1} \qquad \frac{w \xrightarrow{x} y}{f(x) \xrightarrow{f(w)} 1}$$

The label over the conclusion can have at most one function symbol. We have that $1 \sim 2$ but $f(1) \not\sim f(2)$.

$$\frac{}{1 \xrightarrow{f(1)} 1} \qquad \frac{}{2 \xrightarrow{f(1)} 1} \qquad \frac{w \xrightarrow{f(x)} y}{f(x) \xrightarrow{f(w)} 1}$$

Variables in the source of the conclusion cannot occur in the label of the conclusion. We have that $1 \sim 2$ but $f(1) \not\sim f(2)$.

$$\frac{}{f(x) \xrightarrow{f(x)} x}$$

Variables in the target of premise cannot occur in the label of the conclusion. We have that $1 \sim 2$ but $f(1) \not\sim f(2)$.

$$\frac{}{1 \longrightarrow 1} \qquad \frac{}{2 \longrightarrow 2} \qquad \frac{x \longrightarrow y}{f(x) \xrightarrow{y} 1}$$

Variables in the label over the conclusion must be different. We have that $1 \sim 2$ but $g(1,2) \not\sim g(1,1)$.

$$\frac{}{1 \xrightarrow{1} 1} \qquad \frac{}{2 \xrightarrow{1} 1}$$

$$\frac{w \xrightarrow{1} y}{1 \xrightarrow{g(w,w)} y} \qquad \frac{w \xrightarrow{1} y}{2 \xrightarrow{g(w,w)} y} \qquad \frac{w \xrightarrow{g(x_1,x_2)} y}{g(x_1,x_2) \xrightarrow{w} y}$$

2.2 Congruence Proof

For our main result, we will show that if a transition system specification is in *promoted tyft/tyxt* format then (strong) bisimulation is a congruence for the corresponding transition system. From Proposition 2.1, we know it is sufficient to consider a TSS in *promoted tyft* format. We will show that bisimilarity is a congruence by using Howe's technique [How96] for constructing $\widehat{R}$ for any arbitrary preorder R. By construction, $R \subseteq \widehat{R}$ and $\widehat{R}$ respects constructors. The theorem in this section shows that if R is a bisimulation relation then $\widehat{R}$ is also a bisimulation relation. In particular, this mean that for bisimilarity (the largest bisimulation relation) that $\sim\, \subseteq \widehat{\sim}$ and $\widehat{\sim} \subseteq\, \sim$, and therefore $\sim\, = \widehat{\sim}$. Since bisimilarity is an equivalence and respects constructors, it is a congruence.

An interesting aspect of our proof is that rather showing that if $c\ \widehat{R}\ e$ respects bisimulation, we show the stronger result that if $c\ \widehat{R}\ e$, $\vec{d}\ \widehat{R}\ \vec{d'}$ and $c \xrightarrow{g(\vec{d})} c'$ then there exists an e' such that $e \xrightarrow{g(\vec{d'})} e'$ and $c'\ \widehat{R}\ e'$.

Definition 2.3 *Given a preorder R on terms in $T(\Sigma, V)$, define $\widehat{R}$ as follows:*

$$e\ \widehat{R}\ e \qquad \vec{c}\ \widehat{R}\ \vec{e} \text{ and } f(\vec{e})\ R\ e \ \Rightarrow\ f(\vec{c})\ \widehat{R}\ e$$

Lemma 2.2 *We have the following properties:*

1. *$\widehat{R}$ is reflexive,*
2. *$R \subseteq \widehat{R}$,*
3. *$\widehat{R}$ is constructor respecting, and*
4. *if $e\ \widehat{R}\ e'$ and $e'\ R\ e''$, then $e\ \widehat{R}\ e''$.*

Proof: (1) Follows immediately from the first case of the definition. (2) If $c\ R\ e$, where $c = f(\vec{c})$ and we choose $\vec{e} = \vec{c}$ in the definition, then the property follows immediately. (3) If $\vec{c}\ \widehat{R}\ \vec{e}$ where $c = f(\vec{c})$ and we choose $e = f(\vec{e})$ in the definition, then the property follows immediately. (4) From definition, if $e = f(\vec{e})$, there exists some $\vec{e}_0$. $\vec{c}\ \widehat{R}\ \vec{e}_0$ and $f(\vec{e}_0)\ R\ e'$. Since R is transitive, we have $f(\vec{e}_0)\ R\ e''$ and by the definition of $\widehat{R}$ we have $e\ \widehat{R}\ e''$. □

Theorem 2.1 *Consider the transition system specified by a well-founded TSS in* promoted tyft *format. For all terms e and e' and contexts $C[\,]$, if $e \sim e'$ then $C[e] \sim C[e']$.*

Proof:

We want to show that for $c, c', \vec{d}, e \in T(\Sigma, V)$, $g \in \Sigma$, if $c\ \widehat{R}\ e$ and $c \xrightarrow{g(\vec{d})} c'$ then there exists an $e' \in T(\Sigma, V)$ such that $e \xrightarrow{g(\vec{d})} e'$ and $c'\ \widehat{R}\ e'$. To do this, we will actually show the stronger result that $c, c', \vec{d}, \vec{d'}, e \in T(\Sigma, V)$, $g \in \Sigma$, if $c\ \widehat{R}\ e$, $\vec{d}\ \widehat{R}\ \vec{d'}$ and $c \xrightarrow{g(\vec{d})} c'$ then there exists an $e' \in T(\Sigma, V)$ such that $e \xrightarrow{g(\vec{d'})} e'$ and $c'\ \widehat{R}\ e'$. We will do the proof by induction on the height of the proof of $c \xrightarrow{g(\vec{d})} c'$.

$$\frac{\{t_i \xrightarrow{g_i(\vec{t}_i)}_i y_i \mid i \in I\}}{f(\vec{x}) \xrightarrow{g(\vec{w})} t}\,[r1]$$

$$\frac{\{c_i \xrightarrow{g_i(\vec{e}_i)}_i c'_i \mid i \in I\}}{f(\vec{c}) \xrightarrow{g(\vec{d})} c'}\,[r2] \qquad \frac{\{e_i \xrightarrow{g(\vec{e}_i)}_i e'_i \mid i \in I\}}{f(\vec{e}) \xrightarrow{g(\vec{d'})} e''}\,[r3]$$

Since the TSS is in *promoted tyft* format, the last rule applied in the proof has the form of rule [r1] and we have $c = f(\vec{c})$, where $\vec{c}$ might have zero elements and c would simply be a constant. Therefore we have a substitution $\rho = [\vec{w} \mapsto \vec{d}][\vec{x} \mapsto \vec{c}][y_i \mapsto c'_i]^{i \in I}$ and $\rho(r1)$ is rule [r2], where we have proofs of all the premises. Since $c\ \widehat{R}\ e$, by the definition of $\widehat{R}$ we have that there exists a term $f(\vec{e})$ such that $\vec{c}\ \widehat{R}\ \vec{e}$ and $f(\vec{e})\ R\ e$. We want to construct a map ρ' so that $\rho'(r1)$ is rule [r3], where we have proofs of all of the premises and we also have that for all $v \in V$, $\rho(v)\ \widehat{R}\ \rho'(v)$. We will construct the map in several steps.

Step 1: Let $\rho'_0 = [\vec{w} \mapsto \vec{d'}][\vec{x} \mapsto \vec{e}]$. Notice that for all variables v in the domain of ρ_1, $\rho(v) = \rho'_1(v)$.

Step n: There exists some subset of the premises $\{t_i \xrightarrow{g(t_i)}_i y_i \mid i \in I_n \subseteq I\}$ that are dependent only on variables in the domain of ρ'_{n-1}. For all $i \in I_n$, let $e_i = \rho'_{n-1}(t_i)$ and $\vec{e}_i = \rho'_{n-1}(\vec{t}_i)$. Since $\widehat{R}$ respects constructors we have $(c_i\ \widehat{R}\ e_i)^{i \in I_n}$ and $(\vec{c}_i\ \widehat{R}\ \vec{e}_i)^{i \in I_n}$. By the induction hypothesis, since $(c_i \xrightarrow{g(\vec{c}_i)}_i c'_i)^{i \in I_n}$ we have that $(e_i \xrightarrow{g(\vec{e}_i)}_i e'_i)^{i \in I_n}$ where $(c'_i\ \widehat{R}\ e'_i)^{i \in I_n}$. Let $\rho'_n = \rho'_{n-1}[y_i \mapsto e'_i]^{i \in I_n}$. Notice that for all v in the domain of ρ_n, $\rho(v)\ \widehat{R}\ \rho'_n(v)$.

Define ρ' to be the union of the ρ'_n. Since the TSS is well-founded, domain of ρ' will include all of the variables in the rule. We have a proof for each transition $e_i \xrightarrow{g(\vec{e}_i)}_i e'_i$ in the premise of $\rho'(P)$ and we also have that for all variables v that occur in P that $\rho(v)\ \widehat{R}\ \rho'(v)$. Since $c' = \rho(t)$, $e'' = \rho'(t)$ and $\widehat{R}$ respects constructors, we have that $c'\ \widehat{R}\ e''$. Since $f(\vec{e})\ R\ e$, we have that $e \xrightarrow{g(d')} e'$ where $e''\ R\ e'$. Finally by the previous lemma, we have that $c'\ \widehat{R}\ e$, bisimilarity respects evaluation and therefore bisimilarity is a congruence. □

Abramsky's rules (for closed terms):

$$\frac{}{\lambda a.x \Downarrow \lambda a.x}\ (\Downarrow 1) \qquad \frac{x \Downarrow \lambda a.x' \quad x'[y/a] \Downarrow z}{x\ y \Downarrow z}\ (\Downarrow 2)$$

Promoted tyft/tyxt rules:

$$\frac{}{\lambda a.x \Downarrow \lambda a.x}\ (\Downarrow 1) \qquad \frac{x_1 \xrightarrow{\square x_2} y}{x_1\ x_2 \Downarrow y}\ (\Downarrow 2\text{a}) \qquad \frac{x_1\ x_2 \Downarrow y' \quad y' \xrightarrow{\square w} y}{x_1\ x_2 \xrightarrow{\square w} y}\ (\Downarrow 2\text{b}) \qquad \frac{x \xrightarrow{[w/a]} y' \quad y' \Downarrow y}{\lambda a.x \xrightarrow{\square w} y}\ (\Downarrow 2\text{c})$$

Figure 2: Evaluation rules for the Lazy λ-Calculus

3 The Lazy Lambda Calculus

In this section we define the lazy λ-calculus in *promoted tyft/tyxt* format. The syntax of our language is:

$$a \in \text{Identifiers}$$

$$e \in \text{Expressions} ::= a \mid \lambda a.e \mid e_1\ e_2$$

The corresponding signature for our transition system specification includes a countable number of constant symbols for the identifiers, a binary function symbol for application and for each identifier a a unary function symbol $\lambda a.(\ )$. The signature also includes the necessary function symbols for labels on transitions: a unary function symbol $\square(\ )$ and for each identifier a a unary function symbol $[(\ )/a]$. The set of transitions includes two transition relations: an unlabeled transition $\Downarrow$ and a set of predicates fv( ) = S, one for each value of S.

In our definition of the λ-calculus, we define substitution and free variable tests directly with rules in the TSS. This is differs from earlier work, where substitution is defined as part of the rule format [How96, San94, WB96, FV97]. The necessary rules for our definition of the λ-calculus appear in figure 1 and are as one would expect. The predicate $fv(e) = S$ defines the set S of identifiers that occur free in the term e. The transition $e \xrightarrow{[d/a]} e'$ defines when e with closed term d substituted for the identifier a is e'.

Many of the rules in the *promoted tyft/tyxt* definition are actually rule schema. For example, the rule ($\Downarrow 1$) is a rule schema with one rule for each identifier a. Similarly there are free variable rules for each of the appropriate sets of identifiers S.

With the rules at the top of figure 2, Abramsky showed that applicative bisimulation can be used as a suitable notion of equivalence for the lazy λ-calculus [Abr90]. In the definition, the rule [$\Downarrow 2$] is not in *promoted tyft/tyxt* format. In the *promoted tyft/tyxt* definition, we have broken the rule into three parts ([$\Downarrow 2a$], [$\Downarrow 2b$] and [$\Downarrow 2c$]). The rule [$\Downarrow 2a$] is read "if x_1 evaluates to y in a context that offers x_2, then x_1 applied to x_2 evaluates to y." This rule demonstrates the main difference between the *promoted tyft/tyxt* rule format and Groote and Vandraager's *tyft/tyxt* rule format. The new rule format allows the flexibility to promote the term x_2 and place it over the transition arrow. The rules [$\Downarrow 2b$] and [$\Downarrow 2c$] correspond to the two transitions in the hypothesis of [$\Downarrow 2$].

When necessary, we will distinguish transitions in Abramsky's rules as $\Downarrow_a$ and *promoted tyft/tyxt* as $\Downarrow_p$.

Definition 3.1 Applicative bisimulation *(written $\simeq$) is the largest symmetric relation on closed terms such that if $e \simeq d$ and $e \Downarrow \lambda a.e'$ then $d \Downarrow \lambda b.d'$ and for all closed terms c, $e'[c/a] \simeq d'[c/b]$.*

The open extension *of a relation on terms R (written R^o) is a relation on open terms such that for terms e and d, $e\ R^o\ d$ if and only if for all closing substitutions ρ, $\rho(e)\ R\ \rho(d)$.*

Applicative bisimulation is extended to open terms by taking its open extension (written $\simeq^o$).

For closed terms, applicative bisimulation with Abramsky's rules corresponds exactly to bisimulation with the *promoted tyft/tyxt* rules. For open terms, the bisimulation defined by *promoted tyft* rules is strictly finer than applicative bisimulation with Abramsky's rules. The difference is that the "free variable rules" distinguish terms according to the set of free variables. That is, since $fv(\lambda a.\lambda a.a\ \lambda a.b) = \{b\}$ and $fv(\lambda a.\lambda a.a\ \lambda a.a) = \{\}$, we have that

$$(\lambda a.\lambda a.a\ \lambda a.b) \simeq (\lambda a.\lambda a.a\ \lambda a.a)$$

but

$$(\lambda a.\lambda a.a\ \lambda a.b) \not\sim (\lambda a.\lambda a.a\ \lambda a..a).$$

For open terms, we show that $\sim^o$ (the open extension of $\sim$) corresponds exactly to $\simeq^o$. The congruence result is not immediate from the rule format, but can be shown by a simple direct proof.

Lemma 3.1 *The* promoted tyft *rules for the lazy λ-calculus, have the following properties.*

1. *$fv(e) = S$ if and only if S is the set of identifiers that occur free in e.*
2. *$e \xrightarrow{[d/a]} e'$ if and only if d is closed and $e' = e[d/a]$.*
3. *If $e \Downarrow_p e'$ then e' is a λ-bound expression.*
4. *If $\lambda a.d \sim \lambda b.d'$ then for all closed terms c, $d[c/a] \sim d'[c/b]$.*
5. *$e_1\ e_2 \Downarrow_p e$ if and only if $e_1 \xrightarrow{\Box e_2} e$.*

Proof: The first four parts are immediate by structural induction. Part (5) is immediate from rule ($\Downarrow 2a$). □

Proposition 3.2 *For all closed terms e and e',*

$$e \Downarrow_a e' \iff e \Downarrow_p e'.$$

Proof: By induction on the height of the proof. The expression e is either a function definition or an application. If e is a function definition then the only applicable rule in either rule system is ($\Downarrow 1$) and the result follows immediately. Otherwise e could be of the form $(\lambda a.d'\ d)$ or of the form $(d_1\ d_2)\ d$.

First let us consider the case where $e = \lambda a.d'\ d$. Since d is closed, we have that $d' \xrightarrow{[d/a]} d'[d/a]$ and by the induction hypothesis that $d'[d/a] \Downarrow_a e'$ if and only if $d'[d/a] \Downarrow_p e'$. Therefore we have

$$\frac{\lambda a.d' \Downarrow_a \lambda a.d' \quad d'[d/a] \Downarrow_a e'}{\lambda a.d'\ d \Downarrow_a e'}$$

if and only if

$$\cfrac{\cfrac{d' \xrightarrow{[d/a]} d'[d/a] \quad d'[d/a] \Downarrow_p e'}{\lambda a.d' \xrightarrow{\Box d} e'}}{\lambda a.d'\ d \Downarrow_p e'}$$

Now let us consider the case where $e = (d_1\ d_2)\ d$. By Lemma 3.1, we have that the term that $d_1\ d_2$ evaluates to, has to be of the form $\lambda a.d'$. Since d is closed, we have $d'[d/a] \Downarrow e'$. By the induction hypothesis, we have that $d'[d/a] \Downarrow e'$ with Abramsky's rules if and only it is provable with the *promoted tyft*-rules. Therefore we have

$$\frac{d_1\ d_2 \Downarrow_a \lambda a.d' \quad d'[d/a] \Downarrow_a e'}{(d_1\ d_2)\ d \Downarrow_a e'}$$

if and only if

$$\cfrac{\cfrac{d_1\ d_2 \Downarrow_p \lambda a.d' \quad \cfrac{d' \xrightarrow{[d/a]} d'[d/a] \quad d'[d/a] \Downarrow_p e'}{\lambda a.d' \xrightarrow{\Box d} e'}}{d_1\ d_2 \xrightarrow{\Box d} e'}}{(d_1\ d_2)\ d \Downarrow_p e'}$$

□

Proposition 3.3 $\sim\ \subseteq\ \simeq^o$

Proof: We will show that $\sim$ respects applicative bisimulation. That is, for all closing substitutions ρ, $e \sim e'$ and $\rho(e) \Downarrow^a \lambda a.d$ implies that there exists $\lambda b.d'$ such that $\rho(e') \Downarrow^a \lambda b.d'$ and for all closed terms c, $d[c/a] \sim d'[c/b]$.

Consider some arbitrary closing substitution ρ. By Lemma 3.1, we know that e can do the corresponding substitution transitions to ρ and $\rho(e) \sim \rho(e')$. If $\rho(e) \Downarrow_a \lambda a.d$, then by the previous proposition, we have $\rho(e) \Downarrow_t \lambda a.d$. By Lemma 3.1 and since $\rho(e) \sim \rho(e')$, we have that $\rho(e') \Downarrow_t \lambda b.d'$ such that $\lambda a.d \sim \lambda b.d'$. By Lemma 3.1, we have that for all closed terms c, $d[c/a] \sim d'[c/b]$. □

Proposition 3.4 *For closed terms,* $\sim\ =\ \simeq$

Proof: Because of the previous result, we only need to show that $\simeq\ \subseteq\ \sim$. We will show that applicative bisimulation respects bisimulation. That is, if $e \simeq e'$ and $e \xrightarrow{c}_p d$ then there exists some d' such that $e' \xrightarrow{c}_p d'$ and $d \simeq d'$.

Since e and e' are closed expressions, by Lemma 3.1 we have that $fv(e) = S \iff fv(e') = S$ and that $e \xrightarrow{[c/a]} e$ and $e' \xrightarrow{[c/a]} e'$. If $e \Downarrow_p d$ then by the previous proposition $e \Downarrow_a d$ and since $e \simeq e'$, we have $d \simeq d'$. If $e \xrightarrow{\Box c} d$ then again by Lemma 3.1, we have that $e\ c \Downarrow_p d$ and $e\ c \Downarrow_a d$. Since bisimulation is a congruence for Abramsky's lazy lambda calculus, we have that $e\ c \simeq e'\ c$ and that $e'\ c \Downarrow_p d'$ such that $d \simeq d'$ and $e' \xrightarrow{\Box c} d$. □

Proposition 3.5 $\sim^o\ =\ \simeq^o$

Proof: Immediate from previous proposition. □

Since $\simeq^o$ is a congruence, clearly so is $\sim^o$. Even so, we show how to get a direct proof of the congruence result.

Proposition 3.6 $\sim^o$ *is a congruence.*

Proof: It is sufficient to show that $\sim^o$ is operator respecting. Consider an arbitrary context $C[\,]$ and closing substitution ρ. We want to show that, if $d \sim^o e$

Bent Thomsen's rules:

$$\frac{}{m?a.x \xrightarrow{m?w} x[w/a]}\ [\mathit{prefix1}] \qquad \frac{}{m!x'.x \xrightarrow{m!x'} x}\ [\mathit{prefix2}] \qquad \frac{}{\tau.x \xrightarrow{\tau} x}\ [\mathit{prefix3}] \qquad \frac{x \xrightarrow{\Gamma} y}{x + x' \xrightarrow{\Gamma} y}\ [\mathit{choice}]$$

$$\frac{x \xrightarrow{\Gamma} y}{x|x' \xrightarrow{\Gamma} y|x'}\ [\mathit{par1}] \qquad \frac{x \xrightarrow{m?w} y \quad x' \xrightarrow{m!w} y'}{x|x' \xrightarrow{\tau} y|y'}\ [\mathit{par2}] \qquad \frac{x \xrightarrow{m?w} y}{x\backslash n \xrightarrow{m?w} y\backslash n}\ m \neq n\ \ [\mathit{res1}] \qquad \frac{x \xrightarrow{m!w} y}{x\backslash n \xrightarrow{m!w} y\backslash n}\ m \neq n\ \ [\mathit{res2}]$$

$$\frac{x \xrightarrow{\tau} y}{x\backslash n \xrightarrow{\tau} y\backslash n}\ [\mathit{res3}] \qquad \frac{x \xrightarrow{m?w} y}{x[S] \xrightarrow{S(m)?w} y[S]}\ [\mathit{ren1}] \qquad \frac{x \xrightarrow{m!w} y}{x[S] \xrightarrow{S(m)!w} y[S]}\ [\mathit{ren2}] \qquad \frac{x \xrightarrow{\tau} y}{x[S] \xrightarrow{\tau} y[S]}\ [\mathit{ren3}]$$

Note: there are symmetric versions of the rules *choice*, *par1* and *par2*
$\Gamma \in \{m?e, m!e, \tau\}$

Changes for *promoted tyft/tyxt* rules:

$$\frac{x \xrightarrow{[w/a]} y}{m?a.x \xrightarrow{m?w} y}\ [\mathit{prefix1}] \qquad \frac{}{m!x'.x \xrightarrow{m!} (x', x)}\ [\mathit{prefix2}] \qquad \frac{}{(x', x) \xrightarrow{val} x'}\ [\mathit{value}] \qquad \frac{}{(x', x) \xrightarrow{res} x}\ [\mathit{residual}]$$

$$\frac{x \xrightarrow{m!} y \quad y \xrightarrow{val} y_1 \quad y \xrightarrow{res} y_2}{x|x' \xrightarrow{m!} (y_1, y_2|x')}\ [\mathit{par1}(!)] \qquad \frac{x \xrightarrow{m!} y \quad y \xrightarrow{val} y_1 \quad y \xrightarrow{res} y_2 \quad x' \xrightarrow{m?y_1} y'}{x|x' \xrightarrow{\tau} y_2|y'}\ [\mathit{par2}]$$

$$\frac{x \xrightarrow{m!} y}{x + x' \xrightarrow{m!} y}\ [\mathit{choice}(!)] \qquad \frac{x \xrightarrow{m!} y \quad y \xrightarrow{val} y_1 \quad y \xrightarrow{res} y_2}{x\backslash n \xrightarrow{m!} (y_1, y_2\backslash n)}\ m \neq n\ \ [\mathit{res2}] \qquad \frac{x \xrightarrow{m!} y \quad y \xrightarrow{val} y_1 \quad y \xrightarrow{res} y_2}{x[S] \xrightarrow{S(m)!} (y_1, y_2[S])}\ [\mathit{ren2}]$$

Note: there are also symmetric versions of the rules *choice(!)*, *par1(!)* and *par2(!)*.
$\Gamma \in \{m?e, \tau\}$

Figure 3: Evaluation rules for CHOCS

then $\rho(C[d]) \sim^o \rho(C[e])$. By the definition of $\sim^o$, $\rho(d) \sim^o \rho(e)$. So if the context is an application $([\,]\ c)$ then $\rho(d\ c) \sim^o \rho(e\ c)$ since bisimulation is a congruence for closed terms.

If the context is a function definition $\lambda a.[\ ]$, let $\lambda a.d' = \rho(\lambda a.d)$ and $\lambda a.e' = \rho(\lambda a.e)$. The last rule in a proof of a transition could either be $\Downarrow 1$ or $\Downarrow 2$. If one term does a transition by rule $\Downarrow 1$, the other can trivially match it. Since ρ is a closing substitution, the only variable that might occur free in d' or e' is a. By the definition of $\sim^o$ we have that for any closed term c, $d'[c/a] \sim e'[c/a]$ and if one term can do a transition by $\Downarrow 2c$, then the other term can do a matching transition. $\Box$

4 CHOCS

In this section, we present Bent Thomsen's definition for a calculus of higher order communicating systems (CHOCS) [Tho95]. The original definition for CHOCS by Thomsen is almost in *promoted tyft/tyxt* format, although he does use higher order bisimulation as the notion of equivalence. We show how by introducing concretions in a similar style to Milner's use in the polyadic π-calculus [Mil91], we can define CHOCS in promoted tyft/tyxt format.

We assume an infinite set of channel names ranged over by $m, n ...$, and an infinite set of process variables ranged over by $a, b, c....$ The syntax for CHOCS is:

$$p ::= nil \mid m?a.p \mid m!p'.p \mid \tau.p \mid p + p'$$

$$\mid p|p' \mid p\backslash a \mid p[S] \mid a \mid (p, p')$$

$$\Gamma ::= m?a \mid m!p \mid \tau$$

In order to facilitate our definition, we extended Thomsen's syntax to include pairs. The evaluation rules for CHOCS appear in figure 3 and the substitution rules appear in the Appendix.

Thomsen's rule [*prefix1*] relies on an external definition of substitution and we replaced it with a version that uses a substitution transition. The remaining rules are changed to handle concretions. When necessary, we will distinguish transitions in Thomsen's definition as $\longrightarrow_t$ and *promoted tyft/tyxt* as $\longrightarrow_p$.

Definition 4.1 *Given a transition system (S,T) Thomsen's* higher order bisimulation ($\simeq$) *is a binary relation on closed terms in S such that whenever $e\ R\ d$:*

1. *if $e \xrightarrow{\Gamma} e'$ then there exists d' and Γ' such that $d \xrightarrow{\Gamma'} d'$, $\Gamma\ \widehat{R}\ \Gamma'$ and $e'\ R\ d'$*
2. *if $d \xrightarrow{\Gamma} d'$ then there exists e' and Γ' such that $e \xrightarrow{\Gamma'} e'$, $\Gamma\ \widehat{R}\ \Gamma'$ and $e'\ R\ d'$*

where

$$\begin{aligned}\widehat{R} = \{(\Gamma,\Gamma') \mid \Gamma = \Gamma' = \tau\}\\ \cup\{(\Gamma,\Gamma') \mid \Gamma = m!e'' \text{ and } \Gamma' = m!d'' \text{ and } e''\ R\ d''\}\\ \cup\{(\Gamma,\Gamma') \mid \Gamma = m?e'', \Gamma' = m?d'' \text{ and } e''\ R\ d''\}.\end{aligned}$$

Higher order bisimulation is extended from closed terms to open terms by taking the open extension. We will use $\simeq$ for the largest higher order bisimulation.

The following Lemma characterizes the relationship between the two sets of rules.

Lemma 4.1 *For closed terms,*

1. $e \xrightarrow{m!d}_t e'$ *if and only if* $e \xrightarrow{m!}_p (d,e)$.
2. $e \xrightarrow{m?d}_t e'$ *if and only if* $e \xrightarrow{m?d}_p e'$.
3. $e \xrightarrow{\tau}_t e'$ *if and only if* $e \xrightarrow{\tau}_p e'$.

Proof: Straightforward induction on the height of the proof of the transition. □

Lemma 4.2 *(Some properties of pairs.) For all terms e, e', d, d',*

1. $e \xrightarrow{m!}_p e' \Rightarrow \exists d, e'', e' = (d, e'')$
2. $(e,d) \sim e_0 \iff e_0 = (e',d')$ *(where $e \sim e'$ and $d \sim d'$).*

Proof: Straightforward analysis of rules. □

Proposition 4.3 $\sim\ \subseteq\ \simeq^o$

Proof: We will show that bisimulation respects Thomsen's higher order bisimulation. Assume that $e \sim e'$. We will show that for $\Gamma \in \{\tau, m!c, m?c\}$, if $e \xrightarrow{\Gamma}_t e_o$ then there exists a e'_o such that $e' \xrightarrow{\Gamma'}_t e'_o$, $\Gamma \mathrel{\widehat{\sim}} \Gamma'$ and $e_o \sim e'_o$. If Γ is either τ or $m?d$, then by Lemma 4.1, $e \xrightarrow{\Gamma}_p e_o$ and since $e \sim e'$, $e' \xrightarrow{\Gamma}_p e'_o$ where $e_o \sim e'_o$. If Γ is $m!d$, then by Lemma 4.1, $e \xrightarrow{m!}_p (d, e_o)$ and by the definition of bisimulation and Lemma 4.2 $e' \xrightarrow{m!}_p (d', e'_o)$ such that $d \sim d'$ and $e_o \sim e'_o$. By Lemma 4.1, we now have that $e' \xrightarrow{m!d'}_t e'_o$ such that $m!d \mathrel{\widehat{\sim}} m!d'$ and $e_o \sim e'_o$. □

Lemma 4.4 *If $e \xrightarrow{m?d}_t e'$ then for all d' such that $d \simeq d'$, $e \xrightarrow{m?d'}_t e''$ such that $e \simeq e''$.*

Proof: Proposition 3.9 in [Tho95].

Proposition 4.5 *For closed terms,* $\sim\ =\ \simeq$

Proof: By the previous proposition, we only need to show that for closed terms, $\simeq\ \subseteq\ \sim$. Assume that $e \simeq e'$, we will show that for $\Gamma \in \{\tau, m?d, m!, [c/a]\}$ if $e \xrightarrow{\Gamma}_p e_o$ then there exists a e'_o such that $e' \xrightarrow{\Gamma}_p e'_o$ and $e_o \simeq e'_o$. If Γ is τ, then by Lemma 4.1, $e \xrightarrow{\tau}_t e_o$ and since $e \simeq e'$, $e' \xrightarrow{\tau}_t e'_o$ where $e_o \simeq e'_o$. If Γ is $m?d$, then by Lemma 4.1, $e \xrightarrow{m?d}_t e_o$ and since $e \simeq e'$, $e' \xrightarrow{m?d'}_t e'_o$ where $e_o \simeq e'_o$ and $d \simeq d'$. By the previous Lemma, we have $e' \xrightarrow{m?d}_t e''_o$ such that $e'_o \simeq e''_o$ and we have $e_o \simeq e''_o$. Since e and e' are closed expressions, we have that $fv(e) = \{\}$ and $fv(e') = \{\}$ and also that $e \xrightarrow{[c/a]} e$ and $e' \xrightarrow{[c/a]} e'$.

Finally, let us consider the case when $\Gamma = m!$. By Lemma 4.2, we have that $e \xrightarrow{m!}_p (d, e_o)$ and therefore by Lemma 4.1, $e \xrightarrow{m!d}_t e_o$. By the definition of Thomsen's higher order bisimulation, we have that $e' \xrightarrow{m!d'}_t e'_o$ where $d \simeq d'$ and $e_o \simeq e'_o$. Now by Lemma 4.1, we have $e' \xrightarrow{m!}_p (d', e'_o)$. If we treat pairs as related when the elements are related, then $(d, e_o) \simeq (d', e'_o)$. □

Proposition 4.6 $\sim$ *is a congruence.*

Proof: Immediate from Theorem 2.1. □

Our free variable test results in an equivalence that distinguishes terms with different free variables, although Thomsen's definition does not.

Proposition 4.7 $\sim\ \not\subseteq\ \simeq^o$

Sangiorgi's π-calculus rules:

$$\frac{}{\Gamma.x \xrightarrow{\Gamma} x}\ pre \qquad \frac{x_1 \xrightarrow{\Gamma} y}{x_1 + x_2 \xrightarrow{\Gamma} y}\ sum \qquad \frac{x_1 \xrightarrow{\overline{a}b} y_1 \quad x_2 \xrightarrow{a(c)} y_2}{x_1|x_2 \xrightarrow{\tau} y_1|y_2[b/c]}\ com$$

$$\frac{x_1 \xrightarrow{\Gamma} y_1'}{x_1|x_2 \xrightarrow{\Gamma} y_1'|x_2}\ par \quad \mathrm{bn}(\Gamma) \cap \mathrm{fn}(x_2) = \emptyset \qquad \frac{x|!x \xrightarrow{\Gamma} y}{!x \xrightarrow{\Gamma} y}\ rep \qquad \frac{x \xrightarrow{\Gamma} y}{[a=a]x \xrightarrow{\Gamma} y}\ match$$

$$\frac{x_1 \xrightarrow{\overline{a}(b)} y_1 \quad x_2 \xrightarrow{a(b)} y_2}{x_1|x_2 \xrightarrow{\tau} (b)(y_1|y_2)}\ close \qquad \frac{x \xrightarrow{\Gamma} y}{(b)x \xrightarrow{\Gamma} (b)y}\ b \notin n(\Gamma)\ \ res \qquad \frac{x \xrightarrow{\overline{a}b} y}{(b)x \xrightarrow{\overline{a}(b)} y}\ b \neq a\ \ open$$

Changes for *promoted tyft/tyxt* rules:

$$\frac{x_1 \xrightarrow{\overline{a}b} y_1 \quad x_2 \xrightarrow{a(c)} y_2' \quad y_2' \xrightarrow{[b/c]} y_2}{x_1|x_2 \xrightarrow{\tau} y_1|y_2}\ com \qquad \frac{x_1 \xrightarrow{\Gamma} y_1 \quad \mathrm{bn}(\Gamma) \cap \mathrm{fn}(x_2) = \emptyset}{x_1|x_2 \xrightarrow{\Gamma} y_1|x_2}\ par \qquad \frac{x \xrightarrow{\alpha} y' \quad y' \xrightarrow{\Gamma} y}{x \xrightarrow{\Gamma} y}$$

$$\frac{fv(x) \ = S}{\mathrm{bn}(a(b)) \cap \mathrm{fn}(x) = \emptyset}\ (\mathrm{b} \notin \mathrm{S}) \qquad \frac{}{\mathrm{bn}(\overline{a}b) \cap \mathrm{fn}(x) = \emptyset} \qquad \frac{}{\mathrm{bn}(\tau) \cap \mathrm{fn}(x) = \emptyset} \qquad \frac{fv(x) \ = S}{\mathrm{bn}(\overline{a}(b)) \cap \mathrm{fn}(x) = \emptyset}\ (\mathrm{b} \notin \mathrm{S})$$

Note: there are symmetric versions of sum, par, and com.

Figure 4: Evaluation rules for the π-calculus

Proof: Since the following terms $m!nil.nil\backslash m$ and $m!a.nil\backslash m$ have different sets of free variables, we have that $(m!nil.nil)\backslash m \ \simeq \ (m!a.nil)\backslash m$ but $(m!nil.nil)\backslash m \not\sim (m!a.nil)\backslash m$. □

Proposition 4.8 $\quad \simeq^o \ = \ \sim^o$.

Proof: Immediate from definition of open extension.

Proposition 4.9 $\quad \sim^o$ *is a congruence.*

Proof: It is sufficient to show that $\sim^o$ is operator respecting. Consider an arbitrary closing substitution ρ, we want to show

$$d \sim^o e \Rightarrow \rho(C[d]) \sim^o \rho(C[e]).$$

If the context is a function symbol that does not bind any variables, let $C'[\ ] = \rho(C[\ ])$. We have $\rho(C[d]) = C'[\rho(d)]$ and $\rho(C[d']) = C'[\rho(d')]$. By the definition of $\sim^o$, $\rho(d) \sim^o \rho(e)$ and since bisimulation is a congruence for closed terms, $C'[\rho(d)] \sim C'[\rho(d)]$.

The only context that remains to be considered is the input prefix $m?a.[\,]$. Let $m?a.d' = \rho(m?a.d)$ and $m?a.e' = \rho(m?a.e)$. The last rule in a proof of a transition could only be [prefix1]. Since ρ is a closing substitution, the only variable that might occur free in d' or e' is a. By the definition of $\sim^o$ we have that for any closed term c, $d'[c/a] \sim e'[c/a]$ and therefore if one term does a transition by [prefix1], then the other term can perform a matching transition. □

5 The π-Calculus

In this section, we will present a *promoted tyft/tyxt* definition for the π-calculus. The syntax of our language is:

$$P ::= 0 \mid \tau.P \mid b(a).P \mid \overline{b}a.P \mid [a=b]P$$

$$\mid P_1 + P_2 \mid P_1|P_2 \mid !P \mid (a)P$$

We will use Γ to represent an input, an output, or a silent move.

$$\Gamma ::= a(b) \mid \overline{a}b \mid \tau \mid \overline{a}(b)$$

In his treatment of the π-calculus, Sangiorgi identifies terms or actions which are equivalent up to the renaming of bound variables, and defines open bisimulation as the notion of equivalence.

Definition 5.1 *A relation S on process is an* open simulation *if $P\ S\ Q$ implies that for every substitution ρ, whenever $\rho(P) \xrightarrow{\Gamma} P'$ then Q' exists such that $\rho(Q) \xrightarrow{\Gamma} Q'$ and $P'\ S\ Q'$. S is an* open bisimulation *if both S and S^{-1} are open simulations. The processes P and Q are open bisimilar, if $P\ S\ Q$ for some open bisimulation S. We will use $\simeq$ for the largest open bisimulation.*

Sangiorgi's rules are actually already in *tyft/tyxt* format. The only issue is that the definition relies on outside definitions. Again we can define the rules for substitution, free variables and α-conversion in the usual way for the π-calculus. The one new side condition is, $\mathrm{bn}(\Gamma) \cap \mathrm{fn}(e) = \emptyset$. In figure 4, we show how a new predicate can easily be defined for this side condition. We also make the condition that terms are equivalent up to α-conversion explicit by introducing a new rule.

Proposition 5.1 $\quad \sim \subseteq \simeq$.

Proof: Immediate from straightforward analysis of the rules. $\Box$

Proposition 5.2 $\quad \simeq \not\subseteq \sim$.

Proof: Since the two terms have different free variables, $[a = b]nil \simeq nil$ but $[a = b]nil \not\sim nil$. $\Box$

Since closed terms can only do τ actions and cannot do input or output actions, we cannot simply consider closed terms. The issues in the treatment of names are subtle. For example, with the usual treatment of α-conversion, open bisimulation is not a congruence. The $c(d).nil$ is α-convertible with $c(a).nil$ but the term $c(d).[a = b]nil$ is not α-convertible with $c(a).[a = b]nil$. Therefore we have $c(d).nil \xrightarrow{c(a)} nil$ where $c(d).[a = b]nil$ cannot perform a matching transition. That is,

$$[a = b]nil \simeq nil \quad \text{but} \quad c(d).[a = b]nil \not\sim c(d).nil.$$

Sangiorgi implicitly takes an underlying representation of names based on de Bruijn indices. With such a treatment, the side condition in the [par] rule, the α-conversion rules and the free variable rules would be unnecessary and our definition with simulation would correspond exactly to Sangiorgi's definition with open simulation. We leave the development of the definition with de Bruijn indices for future work.

6 Conclusions

In this paper, we have described a rule format that is a simple but expressive generalization of Groote and Vaandrager's *tyft/tyxt* rule format. We proved that bisimulation is a congruence for any language defined in this format, and demonstrated the usefulness of the rule format by presenting definitions for the lazy λ-calculus, CHOCS and the π-calculus.

There are several open questions related to the work in this paper. It is not clear that the well-foundedness property is necessary for the congruence result. We are not sure how the extensions to *tyft/tyxt* format that allow negative premises are compatible with our extensions. It is not clear whether *promoted tyft/tyxt* format is strictly more expressive than *tyft/tyxt* format.

7 Acknowledgements

Many thanks to Alan Jeffrey, Eugene Stark, Wan Fokkink, Chris Verhoef and the anonymous referees for many helpful comments and suggestions.

References

[Abr90] Samson Abramsky. The lazy lambda calculus. In David Turner, editor, *Research Topics in Functional Programming*, The UT Year of Programming Series, pages 65–116. Addison-Wesley Publishing Company, 1990.

[BG96] Roland Bol and Jan Friso Groote. The meaning of negative premises in transition system specifications. *Journal of the ACM*, 43(5):863–914, September 1996.

[BIM95] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, January 1995.

[BS95] Karen L. Bernstein and Eugene W. Stark. Operational semantics of a focusing debugger. In *Eleventh Conference on the Mathematical Foundations of Programming Semantics*, New Orleans, USA, March 1995.

[BV93] Jos Baeten and Chris Verhoef. A congruence theorem for structured operational semantics with predicates. In *Proceedings of the 4th Conference on Concurrency Theory (CONCUR '93)*, volume 715 of *LNCS*, pages 477–492, Hildesheim, Germany, August 1993. Springer-Verlag.

[CG95] Roy L. Crole and Andrew D. Gordon. A sound metalogical semantics for input/output effects. In *CSL'94 Computer Science Logic, Kazimierz, Poland, September 1994*, volume 933 of *Lecture Notes in Computer Science*, pages 339–353. Springer-Verlag, 1995.

[dS85] Robert de Simone. Higher-level synchronising devices in Meije-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[FHJ96] William Ferreira, Matthew Hennessy, and Alan Jeffrey. A theory of weak bisimulation for core CML. In *Proc. ACM SIGPLAN Int. Conf. Functional Programming*. ACM Press, 1996. To appear in Journal of Functional Programming.

[FV97] Wan J. Fokkink and Chris Verhoef. A conservative look at term deduction systems and variable binding. Technical report, CWI, 1997. AP (Department of Software Technology), CS-R9508, To appear in Information and Computation.

[Gor95] Andrew D. Gordon. Bisimilarity as a theory of functional programming. In *Eleventh Annual Conference on Mathematical Foundations of Programming Semantics*, volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1995.

[GR96] Andrew D. Gordon and Gareth D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *Conference Record of the Twenty Third ACM Symposium on Principles of Programming Languages*, pages 386–395. ACM Press, 1996.

[Gro93] J. F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 27 September 1993.

[GV92] J. F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, 1992.

[How95] Douglas J. Howe. A note on proving congruence of bisimulation in a generalized lambda calculus. AT&T Bell Laboratories, 1995.

[How96] Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Control*, 124(2):103–112, February 1996.

[Jef95] Alan Jeffrey. A fully abstract semantics for a nondeterministic functional language with monadic types. In *Eleventh Conference on Mathematical Foundations of Programming Semantics*, volume 1 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1995.

[Mil91] Robin Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, lfcs, October 1991. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, sv , 1993.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I & II. *Information and Computation*, 100(1):1–77, September 1992.

[Par81] D. M. R. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[San92] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, Department of Computer Science, University of Edinburgh, 1992.

[San94] Davide Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1):120–153, 15 May 1994.

[San97] David Sands. From SOS rules to proof principles: An operational metatheory for functional languages. In *24th ACM Symposium on Principles of Programming Languages*, Paris, France, January 1997. ACM Press.

[Tho95] Bent Thomsen. A theory of higher order communication systems. *Information and Computation*, 116:38–57, 1995.

[Ver95] Chris Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, 2(2):274–302, Summer 1995.

[WB96] Sam Weber and Bard Bloom. Metatheory of the π-calculus. Technical report, Cornell University, January 1996.

$$\frac{}{fv(nil)\ =\{\}} \qquad \frac{}{fv(a)\ =\{a\}} \qquad \frac{fv(x)\ =T}{fv(m?a.x)\ =T-a} \qquad \frac{fv(x)\ =T \quad fv(x')\ =T'}{fv(m!x'.x)\ =T\cup T'} \qquad \frac{fv(x)\ =T}{fv(\tau.x)\ =T}$$

$$\frac{fv(x)\ =T \quad fv(x')\ =T'}{fv(x+x')\ =T\cup T'} \qquad \frac{fv(x)\ =T \quad fv(x')\ =T'}{fv(x|x')\ =T\cup T'} \qquad \frac{fv(x)\ =T}{fv(x\backslash a)\ =T} \qquad \frac{fv(x)\ =T}{fv(x[S])\ =T}$$

$$\frac{fv(w)\ =\{\}}{nil \xrightarrow{[w/a]} nil} \qquad \frac{fv(w)\ =\{\}}{a \xrightarrow{[w/a]} a} \qquad \frac{fv(w)\ =\{\} \quad (a\neq b)}{b \xrightarrow{[w/a]} b}$$

$$\frac{}{m?a.x \xrightarrow{[w/a]} m?a.x} \qquad \frac{x \xrightarrow{[w/b]} y \quad (a\neq b)}{m?a.x \xrightarrow{[w/b]} m?a.y} \qquad \frac{x_1 \xrightarrow{[w/a]} y_1 \quad x_2 \xrightarrow{[w/a]} y_2}{m!x_1.x_2 \xrightarrow{[w/a]} m!y_1.y_2} \qquad \frac{x \xrightarrow{[w/b]} y}{\tau.x \xrightarrow{[w/b]} \tau.y}$$

$$\frac{x_1 \xrightarrow{[w/a]} y_1 \quad x_2 \xrightarrow{[w/a]} y_2}{x_1+x_2 \xrightarrow{[w/a]} y_1+y_2} \qquad \frac{x_1 \xrightarrow{[w/a]} y_1 \quad x_2 \xrightarrow{[w/a]} y_2}{x_1|x_2 \xrightarrow{[w/a]} y_1|y_2} \qquad \frac{x \xrightarrow{[w/b]} y}{x\backslash a \xrightarrow{[w/b]} y\backslash a} \qquad \frac{x \xrightarrow{[w/b]} y}{x[S] \xrightarrow{[w/b]} y[S]}$$

Figure 5: Free Variable and substitution rules for CHOCS

Substitution rules

$$\frac{}{0 \xrightarrow{[a/b]} 0} \qquad \frac{x \xrightarrow{[a/b]} y}{\tau.x \xrightarrow{[a/b]} \tau.y} \qquad \frac{x_1 \xrightarrow{[a/b]} y_1 \quad x_2 \xrightarrow{[a/b]} y_2}{x_1+x_2 \xrightarrow{[a/b]} y_1+y_2} \qquad \frac{x_1 \xrightarrow{[a/b]} y_1 \quad x_2 \xrightarrow{[a/b]} y_2}{x_1|x_2 \xrightarrow{[a/b]} y_1|y_2} \qquad \frac{x \xrightarrow{[a/b]} y}{!x \xrightarrow{[a/b]} !y}$$

$$\frac{}{b(b).x, \xrightarrow{[a/b]} a(b).x} \qquad \frac{(a\neq b)}{a(b).x, \xrightarrow{[a'/b]} a(b).x} \qquad \frac{x \xrightarrow{[a'/a]} y \quad (a\neq b)}{a(b).x, \xrightarrow{[a'/a]} a'(b).y} \qquad \frac{x \xrightarrow{[a'/b']} y \quad (a\neq b') \quad (b\neq b')}{a(b).x, \xrightarrow{[a'/b']} a(b).y}$$

$$\frac{x \xrightarrow{[a/b]} y}{\overline{b}b.x, \xrightarrow{[a/b]} \overline{a}a.y} \qquad \frac{x \xrightarrow{[a'/b]} y \quad (a\neq b)}{\overline{a}b.x, \xrightarrow{[a'/a]} \overline{a'}b.y} \qquad \frac{x \xrightarrow{[a'/b]} y \quad (a\neq b)}{\overline{a}b.x, \xrightarrow{[a'/b]} \overline{a}a'.y} \qquad \frac{x \xrightarrow{[a'/b']} y \quad (a\neq b') \quad (b\neq b')}{\overline{a}b.x, \xrightarrow{[a'/b']} \overline{a}b.y}$$

$$\frac{x \xrightarrow{[a'/b]} y}{[a=a]x, \xrightarrow{[a'/b]} [a=a]y} \qquad \frac{x \xrightarrow{[a'/a]} y \quad (a\neq b)}{[a=b]x, \xrightarrow{[a'/a]} [a'=b]y} \qquad \frac{x \xrightarrow{[a'/b]} y \quad (a\neq b)}{[a=b]x, \xrightarrow{[a'/b]} [a=a']y} \qquad \frac{x \xrightarrow{[a'/b']} y \quad (a\neq b') \quad (b\neq b')}{[a=b]x, \xrightarrow{[a'/b']} [a=b]y}$$

$$\frac{}{(a)x, \xrightarrow{[b/a]} (a)x} \qquad \frac{x \xrightarrow{[a'/a]} y \quad (a\neq b)}{(b)x, \xrightarrow{[a'/a]} (b)y}$$

$$\frac{}{\overline{b}(b).x, \xrightarrow{[a/b]} \overline{a}(b).x} \qquad \frac{(a\neq b)}{\overline{a}(b).x, \xrightarrow{[a'/b]} \overline{a}(b).x} \qquad \frac{x \xrightarrow{[a'/a]} y \quad (a\neq b)}{\overline{a}(b).x, \xrightarrow{[a'/a]} \overline{a'}(b).y} \qquad \frac{x \xrightarrow{[a'/b']} y \quad (a\neq b') \quad (b\neq b')}{\overline{a}(b).x, \xrightarrow{[a'/b']} \overline{a}(b).y}$$

Alpha conversion rules

$$\frac{}{x \xrightarrow{\alpha} x} \qquad \frac{x \xrightarrow{[c/b]} y}{a(b).x \xrightarrow{\alpha} a(c).y} \qquad \frac{x \xrightarrow{[c/b]} y}{(b).x \xrightarrow{\alpha} (c).y}$$

Free Variable Rules

$$\frac{}{fv(0)\ =\{\}} \qquad \frac{fv(x)\ =S}{fv(\tau.x)\ =S} \qquad \frac{fv(x)\ =S}{fv(a(b).x)\ =(S-b)\cup\{a\}} \qquad \frac{fv(x)\ =S}{fv(\overline{a}b.x)\ =S\cup\{a,b\}} \qquad \frac{fv(x)\ =S}{fv([a=a].x)\ =S} \qquad \frac{fv(x)\ =S}{fv(!x)\ =S}$$

$$\frac{fv(x)\ =S \quad (a\neq b)}{fv([a=b].x)\ =S\cup\{a,b\}} \qquad \frac{fv(x_1)\ =S_1 \quad fv(x_2)\ =S_2}{fv(x_1+x_2)\ =S_1\cup S_2} \qquad \frac{fv(x_1)\ =S_1 \quad fv(x_2)\ =S_2}{fv(x_1|x_2)\ =S_1\cup S_2} \qquad \frac{fv(x)\ =S}{fv((a)x)\ =S-a} \qquad \frac{fv(a(b).x)\ =S}{fv(\overline{a}(b).x)\ =S}$$

Figure 6: Substitution and Free Variable Rules for the π-calculus