Turning GSOS Rules into Equations for Linear Time-Branching Time Semantics

MACIEJ GAZDA AND WAN FOKKINK

Eindhoven University of Technology, Department of Computer Science
PO Box 513, 5600 MB Eindhoven, The Netherlands
VU University Amsterdam, Department of Computer Science
Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
Email: m.w.gazda@tue.nl, w.j.fokkink@vu.nl

An existing axiomatisation strategy for process algebras modulo bisimulation semantics can be extended so that it can be applied to other behavioural semantics as well. We study term rewriting properties of the resulting axiomatisations.

Keywords: Process Algebra; Structural Operational Semantics; Term Rewriting

1. INTRODUCTION

Labelled transition systems constitute a widely used model of concurrent computation. They model processes by explicitly describing their states and transitions from state to state, together with the actions that produce these transitions. Several notions of behavioural semantics have been proposed, with the aim to identify those states that afford the same observations. In [13], van Glabbeek presented the linear time-branching time spectrum of behavioural semantics for finitely branching, concrete, sequential processes. These semantics are based on simulation notions or on decorated traces. Figure 1 depicts the linear time-branching time spectrum; an arrow from one semantics to another means that the source of the arrow is finer, i.e. more discriminating, than the target.

The process algebra *FINTREE* contains only the basic process algebraic operators from CCS and CSP, but is sufficiently powerful to express all finite labelled transition systems (without τ-transitions). Van Glabbeek [13] associated with most behavioural equivalences in his spectrum a *sound* axiomatisation, to equate closed *FINTREE* terms that are behaviourally equivalent. These axiomatisations were shown to be *ground-complete*, meaning that whenever two closed *FINTREE* terms are behaviourally equivalent, then they can be equated.

Structural operational semantics [2] is used to define the labelled transition system associated to a process algebra term. In structural operational semantics, transitions with action labels between algebraic terms are derived from inductive proof rules, called transition rules, which together make up a transition system specification. Intuitively, validity of the positive and negative premises of a transition rule, under a certain substitution, implies validity of the conclusion of this rule under the same substitution.

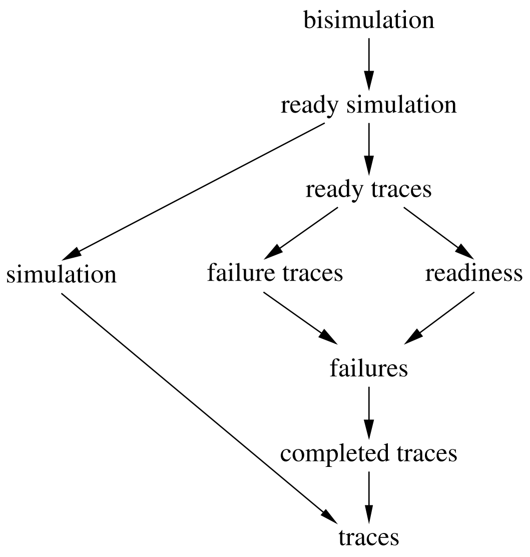

FIGURE 1. Linear time-branching time spectrum

Several syntactic formats have been developed, notably the GSOS format of Bloom, Istrail and Meyer [8], which ensure that the bisimulation equivalence induced by a transition system specification in such a format is always a congruence. Aceto, Bloom and Vaandrager [1] developed an algorithm to generate, given a GSOS system that incorporates *FINTREE*, an axiomatisation that is sound and ground-complete for the associated process algebra modulo bisimulation equivalence. Bosscher [9] studied the term rewriting properties of the obtained axiomatisations, and showed that they are weakly normalising and confluent.

Here we show how this work on generating axiomatisations for extensions of *FINTREE* from GSOS systems can be extended to other process semantics in the linear-time branching time spectrum. This boils down to simply adding the axioms from [13] for the process semantics under consideration. We also study the term rewriting properties of the obtained axiomatisations. They are all weakly normalising, but only some of them are confluent. Moreover, we argue that for two of the process semantics one cannot hope to find a confluent term rewriting system.

2. PRELIMINARIES

2.1. Labelled transition systems

The basic notion that will be used to specify a system is a labelled transition system.

Definition 2.1 (Labelled transition system). *Assume a finite set Act of actions. A* labelled transition system *is a pair $(P, \rightarrow)$ where P is a set of processes and $\rightarrow \subseteq P \times Act \times P$ is a transition relation. We will use $p \xrightarrow{a} q$ to denote $(p, a, q) \in \rightarrow$ (*positive literal*) and $p \not\xrightarrow{a}$ for $\neg\exists q \in P : p \xrightarrow{a} q$ (*negative literal*).*

We extend the transition relation so that it can be labelled with traces of actions. That is, we define $p \xrightarrow{\epsilon} p$ (where ϵ denotes the empty trace), and $p \xrightarrow{a\sigma} q$ for $\sigma \in Act^*$ iff $\exists r : p \xrightarrow{a} r \wedge r \xrightarrow{\sigma} q$. For any $q \in P$ such that $\exists \sigma : p \xrightarrow{\sigma} q$ we say that q is *reachable* from p. We write $I(p) \overset{def}{=} \{a \in Act \mid \exists q \in P : p \xrightarrow{a} q\}$ for the set of *initial actions (initials)* that process p can take in the first step.

Definition 2.2 (Properties of processes). *A process p is:*

- well-founded *if there is no infinite execution trace starting at p;*
- finitely branching *if for all r reachable from p the set $\{q \mid \exists a \in Act : r \xrightarrow{a} q\}$ is finite.*

We assume that all processes we consider are finitely branching, unless explicitly stated otherwise.

2.2. Processes as terms

As usual, process algebra terms [10] are defined over some signature.

Definition 2.3 (Signature, term). *A* signature Σ *is a finite set of function symbols, where to each $f \in \Sigma$ an arity $ar(f) \in \mathbb{N}$ is assigned. If $ar(f) = 0$, then f is called a* constant. *Var denotes a countably infinite set of variables. The set $\mathbb{T}(\Sigma)$ of* terms over Σ *is defined inductively:*
(1) $x \in \mathbb{T}(\Sigma)$ for every $x \in Var$,
(2) If $f \in \Sigma$ and $t_1, ..., t_{ar(f)} \in \mathbb{T}(\Sigma)$, then $f(t_1, ..., t_{ar(f)}) \in \mathbb{T}(\Sigma)$.
A term is closed *if it does not contain any variables. The set of all closed terms over Σ is denoted $\mathcal{T}(\Sigma)$. A term in which all variables belong to a vector of variables $\vec{x}$ is denoted $C[\vec{x}]$.*

A *substitution* is a function $\sigma : Var \rightarrow \mathbb{T}(\Sigma)$. If $\sigma(Var) \subseteq \mathcal{T}(\Sigma)$, then σ is called *closed*. Substitution extends to a function $\sigma : \mathbb{T}(\Sigma) \rightarrow \mathbb{T}(\Sigma)$ in the standard fashion, namely $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$.

2.3. Structural operational semantics

We consider a setting of labelled transition systems of the form $(\mathcal{T}(\Sigma), \rightarrow)$ for some signature Σ of operators on processes. These operators are defined using structural operational semantics [2], by means of a transition system specification.

Definition 2.4 (Transition system specification). *A* transition rule *over Σ is an inference rule of the form:*

$$\frac{H}{t \xrightarrow{a} t'}$$

where H is a (possibly empty) set of literals. A transition system specification (TSS) *is a set of transition rules.*

We will restrict ourselves to the widely studied class of GSOS rules [8]. GSOS stands for Structural Operational Semantics with Guarded recursion.

Definition 2.5 (GSOS format). *A transition rule is in* GSOS format *if it is of the form:*

$$\frac{\bigcup_{i=1}^{l}\{x_i \xrightarrow{a_{ij}} y_{ij} | 1 \leq j \leq m_i\} \cup \bigcup_{i=1}^{l}\{x_i \not\xrightarrow{b_{ik}} | 1 \leq k \leq n_i\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

where for $X = \{x_i\}$ and $Y = \{y_{ij}\}$ we have $X, Y \subseteq Var$ and $X \cap Y = \emptyset$, and $a_{ij}, b_{ik} \in Act$. If $m_i > 0$ then we say that the GSOS rule tests its i-th argument positively. A TSS is in GSOS format if all its rules are and there are finitely many of them.

Definition 2.6 (Transition relation). *The transition relation $\rightarrow$ generated by a TSS T in GSOS format contains all transitions $t \xrightarrow{a} t'$ such that t and t' are closed terms and there is a transition rule $\frac{H}{\gamma}$ and a closed substitution σ such that $\sigma(\gamma) = t \xrightarrow{a} t'$, and all positive literals in $\sigma(H)$ are generated by T, whereas for all negative literals $u \not\xrightarrow{b}$ in $\sigma(H)$, no transition $u \xrightarrow{b} u'$ is generated by T for any closed term u'.*

The transition relation generated by a GSOS system exists, is unique and finitely branching [8].

2.4. *FINTREE* and one-step encapsulation

The process algebra *FINTREE* [1] consists of three basic operators, which can generate all well-founded LTSs. These operators are:

- *Action prefix a* for all $a \in Act$, a unary operator which represents execution of a single action followed by the rest of a process; for a process p, ap is a process that first executes a and afterwards proceeds with p. Action prefix is defined with the following GSOS rule, for each $a \in Act$:

$$\frac{}{ax \xrightarrow{a} x}$$

- *Alternative composition* $+$, a binary operator which represents a choice between two processes. If p and q are processes, then $p+q$ is a process that executes either p or q. Alternative composition is defined with two GSOS rules:

$$\frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$$

- A constant 0, which represents a process that cannot execute any action. There are no transition rules for 0.

We will use the notation $\Sigma_{i \in I} t_i$, for a finite set of indexes $I = \{i_1, ..., i_n\}$, to represent a choice $t_{i_1} + \cdots + t_{i_n}$. In particular, $\Sigma_{i \in \emptyset}$ denotes 0.

The family of *one-step encapsulation operators* ∂^1_B, which is outside *FINTREE*, was used in [1] to obtain a ground-complete axiomatisation for bisimulation equivalence (this issue will be covered in Section 3). Process $\partial^1_B(p)$, where $B \subseteq \mathit{Act}$, behaves like p except that in the first step it cannot take any action from the set B. The GSOS rule for ∂^1_B is, for any $a \notin B$:

$$\frac{x \xrightarrow{a} x'}{\partial^1_B(x) \xrightarrow{a} x'}$$

2.5. Process semantics

An important question is when we should consider two processes distinguishable. We give a short overview of the most common process semantics in the linear time-branching time spectrum from [13], to which we will refer as basic process equivalences.

- *Traces.* Processes are considered *trace equivalent* if they execute exactly the same traces of actions. Formally, $\sigma \in \mathit{Act}^*$ is a *trace* of a process p if $\exists q : p \xrightarrow{\sigma} q$. The set of all traces of p is denoted $T(p)$ and p and q are *trace equivalent* ($p =_T q$) iff $T(p) = T(q)$.
- *Completed traces.* In addition to trace equivalence, we distinguish processes according to execution paths that lead to termination. Let $CT(p) = \{\sigma \in \mathit{Act}^* \mid \exists q : p \xrightarrow{\sigma} q \wedge I(q) = \emptyset\}$. Processes p and q are *completed trace equivalent* ($p =_{CT} q$) iff $T(p) = T(q)$ and $CT(p) = CT(q)$.
- *Failures.* Apart from mere traces, we take into account all subsets of actions that cannot be taken after executing a certain trace. The set of *failure pairs* of p is defined as

  $$F(p) = \{(\sigma, X) \mid \sigma \in \mathit{Act}^* \wedge X \subseteq \mathit{Act} \wedge \exists q : (p \xrightarrow{\sigma} q \wedge I(q) \cap X = \emptyset)\}$$

  Processes p and q are *failures equivalent* ($p =_F q$) iff $F(p) = F(q)$.
  This equivalence plays a crucial role in a model for Hoare's CSP language which replaced an earlier one based on trace equivalence.
- *Readiness.* This notion is based on a similar idea as failures, but now we take into account the set of actions that can be taken after executing a certain trace. We define a set of *ready pairs* of p as:

  $$R(p) = \{(\sigma, X) \mid \sigma \in \mathit{Act}^* \wedge X \subseteq \mathit{Act} \wedge \exists q : (p \xrightarrow{\sigma} q \wedge I(q) = X\ )\ \}.$$

  Processes p and q are *readiness equivalent*, notation $p =_R q$, iff $R(p) = R(q)$.
  The real difference between failures and readiness equivalence is that in case of failures we allow all subsets of $\mathit{Act} \setminus I(q)$, and therefore the presence of a failure pair (σ, X) does not imply that there is a state q with $p \xrightarrow{\sigma} q$ and $I(q) = \mathit{Act} \setminus X$. This is the reason why readiness equivalence is strictly finer than failures equivalence ($=_R \subset =_F$).
- *Failure traces.* This notion strengthens failures, by taking into account a subset of forbidden actions in all steps of a trace. Therefore, a *failure trace* is an alternating sequence of failure subsets and actions: $FT(p) = \{X_0 a_1 X_1 ... a_n X_n \mid \exists p_0, ..., p_n : p = p_0 \wedge p_{i-1} \xrightarrow{a_i} p_i$ for $i = 1, ..., n \wedge \forall i \in \{0, ..., n\} : I(p_i) \cap X_i = \emptyset\ \}$.
  Processes p and q are *failure trace equivalent* ($p =_{FT} q$) iff $FT(p) = FT(q)$.
- *Ready traces.* A combination of ideas from the previous two notions is known as ready traces. Actions are interleaved with sets of initial actions of a state:
  $RT(p) = \{X_0 a_1 X_1 ... a_n X_n \mid \exists p_0, ..., p_n : p = p_0 \wedge p_{i-1} \xrightarrow{a_i} p_i$ for $i = 1, ..., n \wedge \forall i \in \{0, ..., n\} : X_i = I(p_i)\}$. Processes p and q are *ready trace equivalent* ($p =_{RT} q$) iff $RT(p) = RT(q)$.
- *Simulation.* Previous equivalences are examples of decorated trace semantics, where we take into account traces of actions, possibly interleaved with information about initial or forbidden action. A different approach is based on the notion of simulation. We say that $S \subseteq P \times P$ is a *simulation relation* iff:

  $$pSq \wedge p \xrightarrow{a} p' \Rightarrow \exists q' : q \xrightarrow{a} q' \wedge p'Sq'$$

  Processes p and q are *simulation equivalent*, or *similar* ($s =_S q$), if there is a simulation relation S such that pSq and a simulation relation R with qRp. Simulation relation is finer than trace equivalence and independent of all decorated trace semantics.
- *Ready simulation.* This equivalence is finer than all the aforementioned semantics. This time, we define *ready simulation* relation which is a simulation that satisfies the following additional condition: $pSq \Rightarrow I(p) = I(q)$. Processes p and q are *ready simulation equivalent*, notation $p =_{RS} q$ iff there exists a ready simulation S with pSq and a ready simulation R with qRp.
- *Bisimulation.* The finest and most widely used behavioural equivalence. $R \subseteq P \times P$ is a

bisimulation relation iff:
$pRq \wedge p \xrightarrow{a} p'$ then $\exists q' : q \xrightarrow{a} q' \wedge p'Rq'$ and
$pRq \wedge q \xrightarrow{a} q'$ then $\exists p' : p \xrightarrow{a} p' \wedge p'Rq'$.
Processes p and q are *bisimilar*, notation $p \leftrightarrow q$, iff there exists a bisimulation relation R such that pRq.

2.6. Axiomatisation of the basic operators

We now present axiomatisations of *FINTREE* operators for several process equivalences. Proofs of the following soundness and ground-completeness theorems can be found in [13] (and [6] for ready simulation).

THEOREM 2.1. *The following four equations are a sound and ground-complete axiomatisation of FINTREE operators for bisimulation equivalence.*

$$\begin{array}{llcl} (A1) & (x+y)+z & = & x+(y+z) \\ (A2) & x+y & = & y+x \\ (A3) & x+x & = & x \\ (A4) & x+0 & = & x \end{array}$$

THEOREM 2.2. *Sound and ground-complete axiomatisations for FINTREE operators with respect to some basic process equivalences are presented below. For each of the equivalences, its axiomatisation contains axioms A1-A4 and one or two equations given in the table.*

trace	$a(x+y) = ax + ay$
completed trace	$a(bx+u) + a(cy+v)$ $= a(bx+cy+u+v)$
failures	$a(bx+u) + a(by+v)$ $= a(bx+by+u) + a(by+v)$ $ax + a(y+z) = ax + a(x+y) + a(y+z)$
readiness	$a(bx+u) + a(by+v)$ $= a(bx+by+u) + a(by+v)$
failure trace	$I(x) = I(y) \Rightarrow ax + ay = a(x+y)$ $ax + ay = ax + ay + a(x+y)$
ready trace	$I(x) = I(y) \Rightarrow$ $ax + ay = a(x+y)$
simulation	$a(x+y) = a(x+y) + ay$
ready simulation	$a(x+by+bz) + a(x+by)$ $= a(x+by+bz)$

FIGURE 2. Additional axioms for different process equivalences

3. AXIOMATISATION STRATEGY FOR BASIC PROCESS EQUIVALENCES

Given a GSOS system G and a process equivalence $=_N$, we are after an algorithm that generates a sound and ground-complete axiomatisation $\mathcal{T}_G$ for all operators in Σ_G modulo $=_N$. That is, for all closed terms s, t:

$$\mathcal{T}_G \vdash s = t \;\Leftrightarrow\; s =_N t$$

3.1. Axiomatisation strategy for bisimulation equivalence

The axiomatisation strategy discussed here is the one presented in [1] for bisimulation equivalence. It has two variants, the basic and the alternative strategy. Strategies for other process equivalences that we will discuss later are slight variations of it with a few axioms added, depending on the equivalence. We will present briefly the main idea of the algorithm, as much as it is necessary for general understanding and the forthcoming proofs.

First, let us recall the basic definitions concerning TSS formats used in the axiomatisation strategy. It is not necessary to digest the exact definitions of smooth, distinctive and discarding operators; the important thing is how an arbitrary operator that does not satisfy these additional restrictions is expressed with the "good" operators in the axiomatisation.

DEFINITION 3.1 (Smooth operator). *A GSOS rule is* smooth *if it is in the form:*

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\} \,\cup\, \{x_i \overset{b_{ij}}{\not\longrightarrow} \mid i \in K, 1 \le j \le n_i\}}{f(x_1,\ldots,x_l) \xrightarrow{c} C[\vec{x},\vec{y}]}$$

where I and K constitute a partition of $\{1,\ldots,l\}$ and $C[\vec{x},\vec{y}]$ contains only variables from $\{x_k \mid k \in K\} \cup \{y_i \mid i \in I\}$. An operator is smooth if all its transition rules in the defining TSS are.

An example of a non-smooth operator is the priority operator [5].

DEFINITION 3.2 (Distinctive operator). *An operator f from a GSOS system is* distinctive *if:*

- *it is smooth,*
- *for each argument x_i, either all GSOS rules for f test x_i positively or none of them does, and*
- *for each pair of different GSOS rules for f there is an argument for which both rules have a positive antecedent, but with a different action.*

Alternative and parallel composition are smooth but not distinctive. On the other hand, action prefix and the so called "left merge" are examples of distinctive operators, we refer the interested reader to [1] for more details.

The alternative strategy uses the notion of discarding operator in order to introduce an additional peeling law.

DEFINITION 3.3 (Discarding rule). *A GSOS rule is* discarding *if:*

- *it is smooth, and*
- *for no argument x_i that is tested negatively, x_i occurs in the target.*

An operator is discarding if all of its transition rules in the defining TSS are.

The axiomatisation strategy from [1] takes as input an arbitrary TSS G in GSOS format and produces an axiomatisation $\mathcal{T}$. It proceeds in the following steps:

1. If G does not contain all *FINTREE* operators, then they are added to G. Axioms A1-A4 are included in $\mathcal{T}$.
2. For each non-smooth operator f a new smooth operator f^c with a higher arity is introduced, so that the following axiom is sound:

$$f(x_1, ..., x_{ar(f)}) = f^c(x'_1, ..., x'_{ar(f^c)})$$

   with $\forall i\, \exists j : x'_i = x_j$.
   This equation (copying axiom) is included in the axiomatisation.

2a. In the alternative strategy the same kind of axiom is added for each operator f that is not both smooth and discarding, where f^c is a smooth discarding operator.

3. For each smooth (smooth and discarding in the alternative strategy) but not distinctive operator f add distinctive (and discarding - alternative strategy) operators $f_1, ..., f_n$ such that for each $\vec{x}$:

$$f(\vec{x}) = \sum_{i=1}^{n} f_i(\vec{x})$$

   Functions f_i are obtained by simply partitioning the set of transition rules of f. Each subset from the partition gives rise to a new distinctive function. Thus, f is equal to the choice between the f_i. The above equation (distinctifying axiom) is added to the axiomatisation.

4. Once all the fresh auxiliary operators with the corresponding axioms have been added to G and the output axiomatisation, axioms for distinctive operators are added according to the rules specified in [1]. These include:
   - distributivity laws of the form:
     $f(x_1, ..., x_i + y_i, ..., x_{ar(f)}) =$
     $f(x_1, ..., x_i, ..., x_{ar(f)}) + f(x_1, ..., y_i, ..., x_{ar(f)})$
   - action laws (recall Section 2.4 and the ∂^1_B operator):

$$f(P_1, ..., P_{ar(f)}) = aC[x_1, ..., x_{ar(f)}]$$

     where $P_i \in \{\partial^1_{B_i}(x_i), a_i x_i, x_i, 0\}$ for $\emptyset \subset B_i \subset Act$.
   - inaction laws:

$$f(P_1, ..., P_{ar(f)}) = 0$$

     where $P_i \in \{a_i x_i, b_i x_i + y_i, x_i, 0\}$.

4a. The alternative strategy introduces the same distributivity and inaction axioms as above. The action axioms are of the same form except that one-step encapsulation is not used, so $P_i \in \{a_i x_i, x_i, 0\}$. Furthermore, an additional peeling law is introduced which takes the form:

$$f(P_1, ..., b_i x_i + y_i, ..., P_{ar(f)}) = f(P_1, ..., y_i, ..., P_{ar(f)})$$

   where $P_j \in \{a_j x_j, x_j\}$.

3.2. Approximation Induction Principle

For each natural number n we define a *projection operator* π_n, which mimics the behaviour of its argument up to n steps and then terminates. The behaviour of an application of the projection operator to a process is given by the following transition rules:

$$\frac{x \xrightarrow{a} x'}{\pi_{n+1}(x) \xrightarrow{a} \pi_n(x')}$$

DEFINITION 3.4 (Approximation Induction Principle). AIP *[4, 12] states that if two processes are equal up to any finite depth, then the processes themselves are equal.*

If $\pi_n(x) = \pi_n(y)$ for all $n \in N$, then $x = y$.

AIP is sound modulo all process equivalences in the linear time-branching time spectrum [11].

3.3. Ground-completeness of generated axiomatisations

In [1] it was proven that the axiomatisation obtained from a GSOS system in Section 3.1, together with AIP, provides a sound and ground-complete axiomatisation for the process algebra corresponding to the GSOS system, modulo bisimulation equivalence. It will come as no surprise that this result extends to other process semantics in the linear time-branching time spectrum, if the axiomatisation is extended with the corresponding axioms given in Figure 2.

We consider equivalences $=_N$ with the following two properties:

1. $=_N$ is a congruence for all operators in $\Sigma \cup$ *FINTREE* $\cup \{\pi_n \mid n \in \mathbb{N}\}$, and
2. AIP is sound modulo $=_N$.

DEFINITION 3.5 (Head normalisation). *A term $t \in \mathcal{T}(\Sigma)$ is in* head normal form *if it is of the form $\Sigma_{i \in I} a_i t_i$. An axiomatisation $\mathcal{T}$ over Σ is* head normalising *if for each $t \in \mathcal{T}(\Sigma)$ there exists a $t' \in \mathcal{T}(\Sigma)$ in head normal form such that $\mathcal{T} \vdash t = t'$.*

THEOREM 3.1. *Given a head normalising axiomatisation $\mathcal{T}$ over a signature $\Sigma \cup$ FINTREE $\cup \{\pi_n \mid n \in \mathbb{N}\}$ which is sound modulo the process equivalence $=_N$, if $\mathcal{T}$ contains a ground-complete axiomatisation of FINTREE modulo $=_N$, then $\mathcal{T}$ + AIP is ground-complete for $=_N$.*

Proof. Let $s, t \in \mathcal{T}(\Sigma)$ with $s =_N t$. Since $=_N$ is compositional for each projection π_n, we have $\pi_n(s) =_N$

$\pi_n(t)$ for each $n \in \mathbb{N}$. Take any $n \in \mathbb{N}$. Since $\mathcal{T}$ is head normalising, it is not hard to see that $\mathcal{T} \vdash \pi_n(s) = s_n$ and $\mathcal{T} \vdash \pi_n(t) = t_n$ for some $s_n, t_n \in \mathit{FINTREE}$. From the transitivity of $=_N$ and the soundness of $\mathcal{T}$ modulo $=_N$ we obtain $s_n =_N t_n$. Since $\mathcal{T}$ contains a ground-complete axiomatisation of *FINTREE* modulo $=_N$, $\mathcal{T} \vdash s_n = t_n$. So $\mathcal{T} \vdash \pi_n(s) = s_n = t_n = \pi_n(t)$. Hence, for any $n \in \mathbb{N}$, $\mathcal{T} \vdash \pi_n(s) = \pi_n(t)$. Thus $\mathcal{T} + \text{AIP} \vdash s = t$. □

3.4. Congruence formats

Together with the head normalising property of axiomatisations generated by the presented strategy and the result from [11], which yields soundness of AIP for all basic process equivalences, the above theorem implies that in order to obtain a sound and complete axiomatisation for other basic process equivalences we only need now to have a TSS format that would ensure that the equivalence in question is a congruence for all the generated operators.

We briefly recall known congruence formats for basic equivalences; they are summarised in Figure 3 (note that, since the axiomatisation strategy works for GSOS specifications, we consider subformats of GSOS only). GSOS is a congruence format for bisimulation equivalence and ready simulation. Its positive variant (without negative premises) generates operations that respect trace and simulation equivalences.

Bloom, Fokkink and van Glabbeek [7] have obtained congruence formats for ready trace, failure trace, readiness and failures. The following theorem comes from [7].

Theorem 3.2. *If a TSS is in ready trace/readiness format, then the operations that it defines respect ready trace/readiness congruence. If a TSS is in failure trace format, then its operations respect failure trace and failures equivalence.*

In the following lemma, we show that the abovementioned formats are "safe" for the axiomatisation strategy from [1]. The proof is conceptually straightforward, but it requires to digest several technical notions, we therefore refer the interested reader to the appendix for the detailed proof.

Lemma 3.1. *Let $P = (\Sigma, R)$ be a TSS in GSOS format and also in N format for $N \in \{$ ready simulation, ready trace, readiness, failure trace $\}$. Then the TSS $P' = (\Sigma', R')$ with $\Sigma \subset \Sigma'$ and $R \subset R'$ which contains extra operations introduced by both strategies from [1] is also in GSOS N format.*

As a consequence, the axiomatisation strategy can be applied to almost all basic process equivalences.

Theorem 3.3. *For $P = (\Sigma, R)$ a TSS in GSOS N format for some $N \in \{$ ready simulation, ready trace, readiness, failure trace $\}$, there exists an algorithm (strategy) to produce a sound and complete axiomatisation. It is a variation of the axiomatisation strategy from [1], which adds to the set of produced axioms a finite number of specific axioms for each equivalence, according to Theorem 1.2.*

Process equivalence	**Input SOS format**
trace	positive failure trace GSOS
completed trace	-
simulation	positive GSOS
failures	failure trace GSOS
readiness	readiness GSOS
failure trace	failure trace GSOS
ready trace	ready trace GSOS
ready simulation	GSOS
bisimulation	GSOS

FIGURE 3. Formats of TSSs for the axiomatisation strategies

4. GENERATED AXIOMATISATIONS AS TERM REWRITING SYSTEMS

Having extended the axiomatisation strategy to a number of process semantics, the next question is of more computational nature, namely do those axiomatisations give rise to term rewriting systems [3] with "good" properties such as normalisation and confluence?

This issue has already been addressed by Bosscher [9] for the alternative version of the original axiomatisation strategy for bisimulation [1]. He provided a rulified axiomatisation, i.e. a term rewriting system (TRS) based on the generated axioms. This TRS is not strongly normalising; however, a rewriting strategy has been given such that well-founded bisimilar terms (representing well-founded processes) are rewritten to the same normal form, modulo associativity and commutativity (AC) of the $+$. This yields weak normalisation. We use this approach to obtain TRSs for other process equivalences, which are head normalising and possibly confluent. Note that, since our TRSs contain all rules of Bosscher's original rulified axiomatisation, neither of them is strongly normalising.

4.1. Term rewriting system for bisimulation equivalence

First we present a summary of the TRS and a rewriting strategy for bisimulation equivalence from [9].

Definition 4.1 (Rulified axiomatisation). *Suppose we have an axiomatisation $\mathcal{T}$ generated by the alternative strategy for some GSOS system G. A* rulified axiomatisation $\rightarrow$ *consists of rewrite rules given in Table 1.*

The rewrite rules in Table 1 are mostly directed versions of equations from $\mathcal{T}$. There are some

TABLE 1. Rulified axiomatisation

(1)	$x + x \to x$
(2)	$x + 0 \to x$
(3)	$f(x_1, ..., x_{ar(f)}) \to f^c(x'_1, ..., x'_{ar(f^c)})$ for each copying axiom in $\mathcal{T}$
(4)	$f(x_1, ..., x_{ar(f)}) \to \Sigma_{i=1}^{p} f_i(x_1, ..., x_{ar(f)})$ for each distinctifying axiom in $\mathcal{T}$
(5)	$f(P_1, ..., P_{ar(f)}) \to aC[x_1, ..., x_{ar(f)}]$ where $P_i \in \{a_i x_i, x_i, 0\}$ for each action axiom in $\mathcal{T}$
(6)	$f(x_1, ..., x_i + y_i, ..., x_{ar(f)}) \to f(x_1, ..., x_i ..., x_{ar(f)}) + f(x_1, ..., y_i, ..., x_{ar(f)})$ for each distributivity axiom in $\mathcal{T}$
(7)	$f(P_1, ..., P_{ar(f)}) \to 0$ where $P_i \in \{a_i x_i, b_i x_i + y_i, x_i, 0\}$ for each inaction axiom in $\mathcal{T}$
(8)	$f(P_1, ..., b_i x_i + y_i, ..., P_{ar(f)}) \to f(P_1, ..., y_i, ..., P_{ar(f)})$ where $P_j \in \{a_j x_j, x_j\}$ for each peeling axiom in $\mathcal{T}$
(9)	$f(P_1, ..., b_i x_i, ..., P_{ar(f)}) \to 0$ whenever there exists a rewrite rule $f(P_1, ..., b_i x_i + y_i, ..., P_{ar(f)}) \to 0$
(10)	$f(P_1, ..., b_i x_i, ..., P_{ar(f)}) \to f(P_1, ..., 0, ..., P_{ar(f)})$ whenever there exists a rewrite rule $f(P_1, ..., b_i x_i + y_i, ..., P_{ar(f)}) \to f(P_1, ..., y_i, ..., P_{ar(f)})$

exceptions, though. As usual, the rewrite rules for associativity and commutativity of the + are excluded, as they are non-terminating. Thus the rewriting is done for equivalence classes modulo these two laws for +. Furthermore, rewrite rules (9) and (10) are added to obtain a confluent TRS in situations when we would need a rewrite rule $x \to x + 0$ to be able to use the inaction axiom. For the rulified axiomatisation obtained in this way, Bosscher proposed a rewriting strategy that works as follows:

1. Contract all non-action redexes; repeat this step until no more non-action redexes are left.
2. Contract the outermost redex surrounded by the least number of action prefixing operators.
3. Repeat the whole procedure from point 1.

This rewriting strategy is head normalising, and rewrites bisimilar well-founded closed terms to the same normal form, which is a closed *FINTREE* term. This normal form is unique modulo AC of the +.

4.2. Term rewriting systems for other process equivalences

A simple variation of the approach from [9] for bisimulation equivalence provides us with a terminating TRS for well-founded terms with respect to other process equivalences in the linear time-branching time spectrum. Namely, when the rewriting strategy above terminates on a well-founded term, it produces a normal form in *FINTREE*. We introduce a subsequent step, depending on the chosen process equivalence $=_N$:

4. Contract the redex according to rewrite rules based on the additional one or two axioms in Figure 2 for the equivalence $=_N$, or two of the rewrite rules for *FINTREE* ($x + x \to x$, $x + 0 \to x$). Repeat until no more redexes are left.

Definition 4.2 (Rulified axiomatisations for basic process equivalences). *For an equivalence $=_N$ where $N \in \{T, CT, F, R, FT, RT, S, RS\}$, its* rulified axiomatisation *$\to_N$ consists of the rewrite rules (1)-(10) of $\to$, together with one or two extra rewrite rules, given in Figure 4.*

Process equivalence	**Additional rewrite rules**
trace	$ax + ay \to_T a(x + y)$
completed trace	$a(bx + u) + a(cy + v) \to_{CT} a(bx + cy + u + v)$
failures	$a(bx + by + u) + a(by + v) \to_F a(bx + u) + a(by + v)$, $ax + a(x + y) + a(y + z) \to_F ax + a(y + z)$
readiness	$a(bx + by + u) + a(by + v) \to_R a(bx + u) + a(by + v)$
failure trace	$I(x) = I(y) \Rightarrow ax + ay \to_{FT} a(x + y)$, $ax + ay + a(x + y) \to_{FT} ax + ay$
ready trace	$I(x) = I(y) \Rightarrow ax + ay \to_{RT} a(x + y)$
simulation	$a(x + y) + ay \to_S a(x + y)$
ready simulation	$a(x + by + bz) + a(x + by) \to_{RS} a(x + by + bz)$

FIGURE 4. Additional rewrite rules for rulified axiomatisations

We include completed trace equivalence in the considerations, even though no axiomatisation strategy has been defined for this equivalence. Since it is a congruence with respect to *FINTREE*, and has a ground-complete axiomatisation for these basic operators, we can analyse term rewriting properties of the rulified axioms for *FINTREE* only. While the resulting TRSs are always terminating modulo AC of the +, some of the obtained TRSs are not confluent.

We call a TSS well-founded if it generates only well-founded processes.[1] Note that any well-founded process is bisimilar to a process term in *FINTREE*. Equality modulo AC of + is denoted with $=_{AC}$.

Proposition 4.1. *For each equivalence $=_N$, where $N \in \{T, CT, F, R, FT, RT, S, RS\}$, $\to_N$ is terminating modulo AC of the + for FINTREE.*

[1]In [9] there are notions of syntactic and semantic well-foundedness. In this paper we do not deal with syntactic well-foundedness, and abbreviate semantically well-founded to well-founded.

Proof. We observe that each of the additional rewrite rules strictly decreases the number of summands at some level, possibly increasing their number at an outer (higher) level (by level we mean the number of nested action prefixing operators). However, the number of levels remains constant. Therefore, the summands disappear or are "pushed" to the outer term structure. This cannot take forever, since the number of levels is constant. □

COROLLARY 4.1. *For each equivalence $=_N$, where $N \in \{T, S, CT, F, R, FT, RT, RS\}$, $\rightarrow_N$ is terminating modulo AC of the + for well-founded terms.*

THEOREM 4.1. *The rulified axiomatisations of well-founded GSOS systems are confluent for trace, completed trace, ready trace and bisimulation equivalence.*

In the other cases, namely simulation, failures, readiness, failure trace and ready simulation, we obtain a non-confluent TRS.

Proof. Traces: Any rewriting strategy with regard to the TRS for trace equivalence leads to a normal form

$$\textstyle\sum_{j \in J} a_j t_j$$

where $i \neq j \Rightarrow a_i \neq a_j$ and all t_j are normal forms. It suffices to show that any two trace equivalent normal forms s and t are equal modulo AC of the +. We prove this by induction on depth. Let $s = \sum_{i \in I} a_i s_i$ and $t = \sum_{j \in J} a_j t_j$ (with I and J disjoint). Since $s =_T t$, clearly $I(s) = I(t)$. Pick any $a \in I(s)$; since s and t are normal forms, $a_i = a$ and $a_j = a$ for a unique $i \in I$ and $j \in J$. So by induction, $s_i =_{AC} t_j$. Since this holds for any $a \in I(s)$, we conclude that $s =_{AC} t$.

Completed traces: In this case normal forms are of the form

$$\textstyle\sum_{j \in J} a_j t_j$$

where $i \neq j \Rightarrow (a_i \neq a_j$ or $(t_i =_{AC} 0$ and $t_j \neq_{AC} 0)$ or $(t_i \neq_{AC} 0$ and $t_j =_{AC} 0))$, and all t_j are normal forms. The proof is similar to the case of trace equivalence. The only difference is that a normal form can now contain two summands with the same prefix, $a0$ and at where $t \neq_{AC} 0$.

Ready traces: In this case normal forms are of the form

$$\textstyle\sum_{j \in J} a_j t_j$$

where $i \neq j \Rightarrow (a_i \neq a_j$ or $I(t_i) \neq I(t_j))$, and all t_j are normal forms. Again we use induction on depth. Clearly the sets of ready traces that the summands $a_j t_j$ contribute to the overall set of ready traces of the normal form above are disjoint (except for the empty trace). This implies that, given two ready trace equivalent normal forms $s = \sum_{i \in I} a_i s_i$ and $t = \sum_{j \in J} a_j t_j$, for each $i \in I$ there is a unique $j \in J$ with $a_i = a_j$ and $s_i =_{RT} t_j$, and vice versa. By induction, $s_i =_{AC} t_j$. Hence, $s =_{AC} t$.

Bisimulation: Confluence of the TRS for bisimulation equivalence has already been proven in [9].

Now we prove for other process equivalences that the corresponding TRS is not confluent.

Simulation: The normal forms

$$b0 + a(b0 + a0) \text{ and } b0 + a(b0 + a0) + aa0$$

are simulation equivalent. Thus the TRS for simulation equivalence is not confluent.

We could try to overcome this obstacle by involving sequential composition instead of action prefixing in the rewrite rules; this would allow us to compare sequences of arbitrary length. For example, for the two normal forms above a rewrite rule $x; (y + z) + x; y \rightarrow_S x; (y + z)$ would work (provided that we would define sensible axioms and rewrite rules for the sequential composition operator ;). However, there could be an arbitrary number of alternating alternative composition and action prefixing/sequential composition operators in a path to a leaf of a term. This is exemplified by the two simulation equivalent normal forms t and $t + a^n 0$ for $n \geq 2$, where t denotes:

$$\underbrace{b0 + a(b0 + a(b0 + a(\ldots(b0 + a0)\ldots)))}_{n \text{ summands}}$$

We present a proof sketch to argue that there exists no TRS involving action prefixing and/or sequential composition, alternative composition and 0 that would rewrite each of the above terms to a unique form for each $n \in \mathbb{N}$. In a finite set of rewrite rules there is a boundary on the number of nested sequential composition operators in a term (we call this value a depth), say K. Now suppose that we have two terms as above with $n = K + 1$. Suppose further that they are rewritten to a unique form $\sum_{i=1}^{p} t_i$ with $p \geq 1$. We can distinguish two cases; either there is only one summand t_i of depth $K + 1$ or there are at least two of them. In the first case the second term would be rewritten to a form with only one summand of depth $K + 1$, so the second summand a^{K+1} would disappear at some point of the reduction. Let $l \rightarrow_S r$ be the rewrite rule applying which we can rewrite the term $\sum_{i=1}^{p} t'_i$ with two or more summands of depth $K + 1$ to a term with only one summand of depth $K + 1$ and other possible summands with lower depth. It is easy to see that it cannot be applied at a position within the scope of a sequential composition operator; this is because no term is similar to a term of lower depth. Since we rewrite modulo AC, we can in fact assume that we apply the rule in position ϵ (root position). Consider the case when $l \rightarrow_S r$ is applied at the root position and let t'_{i_0} be a term of depth $K + 1$ that would disappear by applying $l \rightarrow_S r$. Take a leaf of

t'_{i_0} with depth $K+1$ an add a term $a(a+b)$ at the end of it. Call the resulting term t'' and consider a term obtained from $\sum_{i=1}^{p} t'_i$ by replacing t'_{i_0} with t''. Its depth is $K+2$ and thus is obviously not similar to our redex $\sum_{i=1}^{p} t'_i$. However, this term would be rewritten with $l \to_S r$ to the same term as the result of applying $l \to_S r$ to $\sum_{i=1}^{p} t'_i$. The same argument can be used in the second case where the unique normal form would have two or more summands of depth $K+1$.

Ready simulation: The counterexample is a slightly modified version of the one that we used in the case of simulation. The normal forms t and $t + a^n 0$, where t denotes

$$\underbrace{ab0 + a(ab0 + a(ab0 + a(\ldots(ab0 + aa0)\ldots)))}_{n-1 \text{ summands}}$$

are ready simulation equivalent for $n \geq 0$.

With a slight variation of the proof sketch for simulation equivalence, one can argue that there exists no TRS involving action prefixing and/or sequential composition, alternative composition and 0 that would rewrite each of the above terms to a unique form for each $n \in \mathbb{N}$.

For the remaining equivalences, we provide counterexamples for the rulified axiomatisations defined in this paper. However, we do not know of a proof that there cannot exist a confluent TRS.

Readiness and failures: The terms

$$a(bc0 + d0) + a(b0 + e0) \text{ and } a(b0 + d0) + a(bc0 + e0)$$

are readiness equivalent, and so also failure equivalent. Moreover, they are normal forms for the TRSs for readiness equivalence as well as failures equivalence. So these TRSs are not confluent.

The following more general definition of such normal forms shows that developing confluent TRSs for readiness and failures equivalence is a difficult task, if at all possible:

$$\begin{aligned} s_{n+1} &\overset{def}{=} a(bt_n + d0) + a(b0 + e0) \\ t_{n+1} &\overset{def}{=} a(b0 + d0) + a(bs_n + e0) \\ s_0, t_0 &\overset{def}{=} c0 \end{aligned}$$

$s_n =_R t_n$ for all $n \geq 0$.

Failure traces: The normal forms

$$a(a^n 0 + a0) + ab0 \text{ and } a(a^n 0 + a0) + ab0 + a(a^n 0 + b0)$$

are failure trace equivalent, but not equal modulo AC of the +, for $n \geq 1$. Thus the TRS for failure trace equivalence is not confluent.

□

The results are summarized in Figure 5.

Process equivalence	**Term rewriting system**
trace	confluent
completed trace	confluent
failures	not confluent / no confluent TRS known
readiness	not confluent / no confluent TRS known
failure trace	not confluent / no confluent TRS known
ready trace	confluent
simulation	not confluent / no confluent TRS exists
ready simulation	not confluent / no confluent TRS exists
bisimulation	confluent

FIGURE 5. Term rewriting properties of the rulified axiomatisations

REFERENCES

[1] Aceto, L., Bloom, B. and Vaandrager, F.W. (1994) Turning SOS Rules into Equations. Information and Computation, 111(1), 1-52.

[2] Aceto, L., Fokkink, W.J. and Verhoef, C. (2001) Structural Operational Semantics. In Bergstra, J.A., Ponse, A. and Smolka, S.A. (eds), Handbook of Process Algebra. Elsevier.

[3] Baader, F. and Nipkov, T. (1998) Term Rewriting and All That. Cambridge University Press.

[4] Baeten, J.C.M., Bergstra, J.A. and Klop, J.W. (1987) On the Consistency of Koomen's Fair Abstraction Rule. Theoretical Computer Science, 51(1/2), 129-176.

[5] Baeten, J.C.M., Bergstra, J.A. and Klop, J.W. (1987) Ready-trace Semantics for Concrete Process Algebra with the Priority Operator. Computer Journal 30(6), 498-506.

[6] Blom, S.C.C., Fokkink, W.J., Nain,S. (2003) On the Axiomatizability of Ready Traces, Ready Simulation and Failure Traces. Proceedings of ICALP'03, Eindhoven, 30 June - 4 July, pp. 109-118, LNCS, Springer.

[7] Bloom, B., Fokkink, W.J. and van Glabbeek, R.J. (2004) Precongruence Formats for Decorated Trace Semantics. ACM Transactions on Computational Logic, 5(1), 26-78.

[8] Bloom, B., S. Istrail and Meyer, A.R. (1995) Bisimulation can't be traced. Journal of the ACM, 42(1), 232-268.

[9] Bosscher, D.J.B. (1994) Term Rewriting Properties of SOS Axiomatisations. In Proceedings of TACS'94, Sendai, 19-21 April, pp. 425-439, LNCS, Springer.

[10] Fokkink, W.J. (2000) Introduction to Process Algebra. Texts in Theoretical Computer Science. Springer.

[11] Gazda, M. and Fokkink, W.J. (2012) Modal Logic and the Approximation Induction Principle. Mathematical Structures in Computer Science, 22, 175-201.

[12] van Glabbeek, R.J. (1987) Bounded Nondeterminism and the Approximation Induction Principle in Process Algebra. Proceedings of STACS, Passau, Germany, 19-21 February, pp. 336-347, Springer.

[13] van Glabbeek, R.J. (2001) The Linear Time - Branching Time Spectrum I. In Bergstra, J.A., Ponse, A. and Smolka, S.A. (eds), Handbook of Process Algebra, Elsevier.

APPENDIX A. PROOFS OF SECTION 3.4

Bloom, Fokkink and van Glabbeek have obtained congruence formats for ready trace, failure trace, readiness and failures in [7]. The congruence formats for decorated trace equivalences are here adapted into the setting of the GSOS format.

Definition A.1 (Propagation, polling). *An occurrence of a variable in a GSOS rule is* propagated *if the occurrence is either in the target or in the left-land side of a positive premise whose right-hand side occurs in the target. An occurrence of a variable in a GSOS rule is* polled *if the variable occurs on the left-hand side of a positive premise and is not propagated (does not occur in the target).*

The congruence formats use the notion of a floating variable, which may represent a running process. To this end we need to introduce a predicate Λ on arguments of function symbols. The interested reader is referred to [7] for details underlying these concepts.

Definition A.2 (Liquid and frozen arguments, floating variables). *Assume a predicate Λ on arguments of function symbols. If $\Lambda(f,i)$, then we say that an argument i of f is liquid (w.r.t Λ), otherwise it is frozen. A variable in a GSOS rule is Λ-floating if either it occurs as a right-hand side of a positive premise or it occurs in the source at a Λ-liquid position.*

Definition A.3 (Decorated trace safe GSOS rules and decorated trace formats). *Assume a predicate Λ on arguments of function symbols. A GSOS rule is:*
- Λ-ready trace safe if each Λ-floating variable is propagated at most once, and at a liquid position,
- Λ-readiness safe if it is Λ-ready trace safe and no Λ-floating variable is both propagated and polled,
- Λ-failure trace safe if it is Λ-readiness safe and each Λ-floating variable is polled at most once, at a liquid position in a positive premise.

A TSS is in ready trace, readiness or failure trace format if all its rules are Λ-ready trace safe, Λ-readiness safe or Λ-failure trace safe respectively for some predicate Λ.

In order to adapt the decorated trace formats in our axiomatisation strategy, we only need to make sure that the operations introduced by copying and distinctifying axioms are definable with decorated trace rules, provided that the whole input TSS is in one of the above formats.

Lemma A.1. *Let $P = (\Sigma, R)$ be a TSS in GSOS format and also in N format for $N \in \{$ ready simulation, ready trace, readiness, failure trace $\}$. Then the TSS $P' = (\Sigma', R')$ with $\Sigma \subset \Sigma'$ and $R \subset R'$ which contains extra operations introduced by both strategies from [1] is also in GSOS N format.*

Proof. P' is obviously in GSOS format. Let $f' \in \Sigma' \setminus \Sigma$ be an operation introduced by one of the strategies. Then f' is used as an auxiliary operator for either a distinctifying or a copying axiom.

In the first case, the rules for f' are a subset of rules for some $g \in \Sigma$ which are given in N format. Therefore, rules that define f' are also in N format.
In the latter case, f' is obtained from some $f \in \Sigma$ by introducing extra variables so that $f(x_1, ..., x_m) = f'(x'_1, ..., x'_n)$ where $n > m$ and for each i there exists a j with $x'_i = x_j$. The rules for f' are obtained by replacing occurrences of x-variables with the corresponding x'-variables. For $i \neq j$, x_i and x_j are replaced by disjoint sets of corresponding x'_{k_i}, x'_{k_j}.

Consider a rule from R' defining f':

$$\frac{\bigcup_{i\in I}\{x'_i \xrightarrow{a_i} y_i\} \cup \bigcup_{i\in K}\{x'_i \overset{b_{i_j}}{\not\to}\}}{f'(x'_1, ..., x'_n) \xrightarrow{a} C[x', y]} \tag{A.1}$$

In both strategies, there is a surjective mapping $\sigma : \{1, ..., n\} \to \{1, ..., m\}$ and a corresponding rule from R of the form:

$$\frac{\bigcup_{i\in\sigma(I)}\{(x_{\sigma(i)}) \xrightarrow{a_i} y_i\} \cup \bigcup_{i\in\sigma(K)}\{x_{\sigma(i)} \overset{b_{i_j}}{\not\to}\}}{f(x_1, ..., x_m) \xrightarrow{a} C[\sigma(x'), y]} \tag{A.2}$$

where $\sigma(x')$ denotes the vector $x_{\sigma(1)}, ..., x_{\sigma(n)}$.

Let us also define $\Lambda(f', i) \Leftrightarrow \Lambda(f, \sigma(i))$, so that an argument of f is liquid if and only if the corresponding argument of f' is also liquid.

Below we present the proofs that the first rule preserves the syntactic restrictions of the second rule:

1) Ready trace:
Suppose rule 2 is in ready trace format. We have to prove that each Λ-floating variable has at most one propagated occurrence at a Λ-liquid position.
Let z be a Λ-floating variable. There are two possible cases. First, z can occur at the right-hand side of a positive premise, so it is one of y_i for some i. The target in rule 1 does not change occurrences of y-variables as compared to the rule 2, in which it occurred at most once. In the second case $z = x_i$ for some i, and it occurs exactly once in the source, at a Λ-liquid position. Therefore $\Lambda(f, x_{\sigma(i)}) = \Lambda(f', i) = 1$ and $x_{\sigma(i)}$ has at most one propagated occurrence in rule 2. We can view the target of rule 2 as the same open term as the target of rule 1 with variables x_i substituted with $x_{\sigma(i)}$. So if $x_{\sigma(i)}$ has at most one occurrence in rule 2, then x_i must have at most one occurrence in rule 1 as

well.

2) Readiness:
The ready trace format has already been proven. We have to show that no Λ-floating variable has both propagated and polled occurrences.
If z is a Λ-floating variable that has a polled occurrence, then $z = x_i$ for some i (this is because the rule does not contain any lookahead). Since $x_{\sigma(i)}$ is also Λ-floating in rule 2, it does not have a propagated occurrence there. Thus, neither has x_i.

3) Failure trace:
If rule 2 is in failure trace format, then rule 1 is in readiness format. What remains to prove is that each Λ-floating variable has at most one polled occurrence, which must be at a Λ-liquid position in a positive premise. Actually, the second fact is immediate since the rule is in GSOS format, so the left-hand sides of premises are single variables.
As before, we derive the desired property from the fact that for a Λ-floating variable x_i, $x_{\sigma(i)}$ is also Λ-floating and therefore has at most one polled occurrence in rule 2. Since x_i cannot have more occurrences than $x_{\sigma(i)}$, we have proven that rule 1 is in failure trace format. □