On Cool Congruence Formats for Weak Bisimulations
(extended abstract)

Rob van Glabbeek

National ICT Australia
and School of Computer Science and Engineering
The University of New South Wales
`rvg@cs.stanford.edu`

Abstract. In TCS 146, Bard Bloom presented rule formats for four main notions of bisimulation with silent moves. He proved that weak bisimulation equivalence is a congruence for any process algebra defined by *WB cool rules*, and established similar results for rooted weak bisimulation (Milner's "observational congruence"), branching bisimulation and rooted branching bisimulation. This study reformulates Bloom's results in a more accessible form and contributes analogues for (rooted) η-bisimulation and (rooted) delay bisimulation. Moreover, finite equational axiomatisations of rooted weak bisimulation equivalence are provided that are sound and complete for finite processes in any RWB cool process algebra. These require the introduction of auxiliary operators with lookahead. Finally, a challenge is presented for which Bloom's formats fall short and further improvement is called for.

Introduction

Structural Operational Semantics [8, 10] is one of the main methods for defining the meaning of operators in system description languages like CCS [8]. A system behaviour, or *process*, is represented by a closed term built from a collection of operators, and the behaviour of a process is given by its collection of (outgoing) transitions, each specifying the action the process performs by taking this transition, and the process that results after doing so. For each n-ary operator f in the language, a number of *transition rules* are specified that generate the transitions of a term $f(p_1, \ldots, p_n)$ from the transitions (or the absence thereof) of its arguments $p_1, \ldots, p_n$.

For purposes of representation and verification, several behavioural equivalence relations have been defined on processes, of which the most well-known is *strong bisimulation equivalence* [8], and its variants *weak* and *branching* bisimulation equivalence [8, 7], that feature abstraction from internal actions. In order to allow compositional system verification, such equivalence relations need to be *congruences* for the operators under consideration, meaning that the equivalence class of an n-ary operator f applied to arguments $p_1, \ldots, p_n$ is completely determined by the equivalence classes of these arguments. Although strong bisimulation equivalence is a congruence for the operators of CCS and many other languages found in the literature, weak bisimulation equivalence fails to be a congruence for the *choice* or *alternative composition* operator $+$ of CCS. To bypass this problem one uses the coarsest congruence relation for $+$ that is finer than weak bisimulation equivalence, characterised as *rooted weak bisimulation equivalence* [8, 3], which turns out to be a minor variation of weak bisimulation equivalence, and a congruence for all of CCS and many other languages. Analogously, *rooted branching bisimulation* is the coarsest congruence for CCS and many other languages that is finer than branching bisimulation equivalence [7].

In order to streamline the process of proving that a certain equivalence is a congruence for certain operators, and to guide sensible language definitions, syntactic criteria (*rule formats*) for the transition rules in structural operational semantics have been developed, ensuring that the equivalence is a congruence for any operator specified by rules that meet these criteria. One of these is the *GSOS format* of Bloom, Istrail & Meyer [5], generalising an earlier format by De Simone [11]. When adhering to this format, all processes are computably finitely branching, and strong bisimulation equivalence is a congruence [5]. Bloom [4] defines congruence formats for (rooted) weak and branching bisimulation equivalence by imposing additional restrictions on the GSOS format. As is customary in this field, finer equivalences have wider formats, so Bloom's *BB cool* GSOS format, which guarantees that branching bisimulation equivalence is a congruence, is more general than his *WB cool* GSOS format, which suits weak bisimulation equivalence; also his *RWB cool* GSOS format, suiting rooted weak bisimulation, is more general than the WB cool GSOS format, and his *RBB cool* GSOS format, guaranteeing that rooted branching bisimulation equivalence is a congruence, is the finest of all. The prime motivating example for these formats is the structural operational semantics of CCS [8]. All CCS operators are RWB cool, and the CCS operators other than the $+$ are even WB cool.

Bloom's formats involve a fast bookkeeping effort of names of variables, used to precisely formulate the *bifurcation rules* that his formats require. To make his work more accessible, Bloom also presents simpler but less general versions of his formats, obtained by imposing an additional syntactic restriction. This restriction makes it possible to simplify the bifurcation rules to *patience rules*, which do not require such an extensive bookkeeping. Fokkink [6] generalises Bloom's *simply RBB cool* format to a format he calls *RBB safe*, and writes "The definition of bifurcation rules is deplorably complicated, and we do not know of any examples from the literature that are RBB cool but not simply RBB cool. Therefore, we refrain from this generalisation here." Ulidowski [12–14] studies congruence formats for variations of the semantic equivalences mentioned above with a different treatment of divergence. Ulidowski's formats form the counterparts of Bloom's simply cool formats only.

The main aim of the present study is to simplify and further clarify Bloom's work, so as to make it more accessible for the development of applications, variations and extensions. In passing, analogous results are obtained for two equivalences, and their rooted variants, that bridge the gap between weak and branching bisimulation. Moreover, the method of Aceto, Bloom & Vaandrager [1] to extract from any GSOS language a finite equational axiomatisation that is sound and complete for strong bisimulation equivalence on finite processes, is adapted to rooted weak bisimulation equivalence. In the construction fresh function symbols may need be added whose transition rules have *lookahead* and thereby fall outside the GSOS format.

One of the simplifications of Bloom's formats presented here stems from the observation that the operators in any of the cool formats can be partitioned in *principal operators* and *abbreviations*, such that the abbreviations can be regarded as syntactic sugar, adding nothing that could not be expressed with principal operators. For any abbreviation f there exists a principal operator $f^\star$ that typically takes more arguments. For instance, $f(x_1, x_2)$ could be an abbreviation of $f^\star(x_1, x_1, x_2)$. The rules for the abbreviations are completely determined by the rules for the principal operators, and for principal operators patience rules suffice, i.e. one does not need the full generality of bifurcation rules. Moreover, the simply cool formats can be characterised by the requirement that all operators are principal. These observations make it possible to define the cool formats of Bloom without mentioning bifurcation rules altogether. It also enables a drastic simplification of the congruence proofs, namely by establishing the congruence results for the simply cool formats first, and reducing the general case to the simple case by means of some general insights in abbreviation expansion.

Even though any operation that fits the cool formats can also be defined using merely the simply cool formats, in practice it may be handy to work with the full generality of the cool formats. The unary copying operator cp of [5] (page 257) for instance does not fit the cool formats directly, but can be made to fit by adding an auxiliary binary copying operator to the language, of which the unary one is an abbreviation. Dumping the abbreviation from the language would appear unnatural here, as the unary operator motivates the rules for both itself and its binary expansion, the latter being needed merely to make it work.

Another simplification contributed here is in the description of the RWB cool format. Bloom requires for every operational rule with target t the existence of two terms t_1 and t_2, and seven types of derived operational rules. I show that without limitation of generality it is always possible to choose $t_2 = t$, thereby making four of those seven types of rules redundant. Thus, the same format is obtained by requiring only t_1 and two types of derived rules (the third being a patience rule, that was already required for its own sake).

After defining the basic concepts in Section 1, I present the simply cool congruence formats in Section 2. Section 3 presents the theory of abbreviations that lifts the results from the simple to the general formats, and Sect. 4 deals with the rooted congruence formats. Section 5 compares my definitions of the cool formats with the ones of Bloom. Section 6 recapitulates the method of [1] to provide finite equational axiomatisations of strong bisimulation equivalence that are sound and complete for finite processes on an augmentation of any given GSOS language, and Sect. 7 extends this work to the rooted weak equivalences. Finally, Sect. 8 presents a fairly intuitive GSOS language for which the existing congruence formats fall short and further improvement is called for.

Acknowledgement. My thanks to Simone Tini for inspiration.

1 Preliminaries

In this paper $V = \{x_1, x_2, \ldots\}$ and *Act* are two sets of *variables* and *actions*.

Definition 1. A *signature* is a collection Σ of *function symbols* $f \notin V$ equipped with a function $ar : \Sigma \rightarrow \mathbb{N}$. The set $\mathbb{T}(\Sigma)$ of *terms* over a signature Σ is defined recursively by:

- $V \subseteq \mathbb{T}(\Sigma)$,
- if $f \in \Sigma$ and $t_1, \ldots, t_{ar(f)} \in \mathbb{T}(\Sigma)$ then $f(t_1, \ldots, t_{ar(f)}) \in \mathbb{T}(\Sigma)$.

A term $c()$ is abbreviated as c. For $t \in \mathbb{T}(\Sigma)$, $var(t)$ denotes the set of variables that occur in t. $T(\Sigma)$ is the set of closed terms over Σ, i.e. the terms $p \in \mathbb{T}(\Sigma)$ with $var(p) = \emptyset$. A *Σ-substitution* σ is a partial function from V to $\mathbb{T}(\Sigma)$. If σ is a substitution and S is any syntactic object, then $\sigma(S)$ denotes the object obtained from S by replacing, for x in the domain of σ, every occurrence of x in S by $\sigma(x)$. In that case $\sigma(S)$ is called a *substitution instance* of S. A Σ-substitution is *closed* if it is a total function from V to $T(\Sigma)$.

Definition 2. Let Σ be a signature. A *positive Σ-literal* is an expression $t \xrightarrow{a} t'$ and a *negative Σ-literal* an expression $t \overset{a}{\not\longrightarrow}$ with $t, t' \in \mathbb{T}(\Sigma)$ and $a \in Act$. A *transition rule* over Σ is an expression of the form $\frac{H}{\alpha}$ with H a set of Σ-literals (the *premises* of the rule) and α a positive Σ-literal (the *conclusion*). The left- and right-hand side of α are called the *source* and the *target* of the rule, respectively. A rule $\frac{H}{\alpha}$ with $H = \emptyset$ is also written α. A *transition system specification* (*TSS*), written (Σ, R), consists of a signature Σ and a set R of transition rules over Σ. A TSS is *positive* if the premises of its rules are positive.

Definition 3. [5] A *GSOS* rule is a transition rule such that

- its source has the form $f(x_1, \ldots, x_{ar(f)})$ with $f \in \Sigma$ and $x_i \in V$,
- the left-hand sides of its premises are variables x_i with $1 \leq i \leq ar(f)$,
- the right-hand sides of its positive premises are variables that that are all distinct, and that do not occur in its source,
- its target only contains variables that also occur in its source or premises.

A *GSOS language*, or TSS in GSOS format, is a TSS whose rules are GSOS rules.

Definition 4. A *transition* over a signature Σ is a closed positive Σ-literal. With structural recursion on p one defines when a GSOS language $\mathcal{L}$ generates a transition $p \xrightarrow{a} p'$ (notation $p \xrightarrow{a}_{\mathcal{L}} p'$):

$f(p_1, \ldots, p_n) \xrightarrow{a}_{\mathcal{L}} q$ iff $\mathcal{L}$ has a transition rule $\frac{H}{f(x_1,\ldots,x_n) \xrightarrow{a} t}$ and there is a closed substitution σ with $\sigma(x_i) = p_i$ for $i = 1, ..., n$ and $\sigma(t) = q$, such that $p_i \xrightarrow{c}_{\mathcal{L}} \sigma(y)$ for $(x_i \xrightarrow{c} y) \in H$ and $\neg\exists r(p_i \xrightarrow{c}_{\mathcal{L}} r)$ for $(x_i \overset{c}{\not\longrightarrow}) \in H$.

Henceforth a GSOS language $\mathcal{L}$ over a signature Σ is assumed, and closed Σ-terms will be called *processes*. The subscript $\mathcal{L}$ will often be suppressed. Moreover, $Act = A \stackrel{\bullet}{\cup} \{\tau\}$ with τ the *silent move* or *hidden action*.

Definition 5. Two processes t and u are *weak bisimulation equivalent* or *weakly bisimilar* ($t \leftrightarrows_w u$) if $t\mathcal{R}u$ for a symmetric binary relation $\mathcal{R}$ on processes (a *weak bisimulation*) satisfying, for $a \in Act$,

if $p\mathcal{R}q$ and $p \xrightarrow{a} p'$ then $\exists q_1, q_2, q'$ such that $q \Longrightarrow q_1 \xrightarrow{(a)} q_2 \Longrightarrow q' \wedge p'\mathcal{R}q'$. (*)

Here $p \Longrightarrow p'$ abbreviates $p = p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_n = p'$ for some $n \geq 0$, whereas $p \xrightarrow{(a)} p'$ abbreviates $(p \xrightarrow{a} p') \vee (a = \tau \wedge p = p')$.

t and u are *η-bisimilar* ($t \leftrightarrows_\eta u$) if in (*) one additionally requires $p\mathcal{R}q_1$;

t and u are *delay bisimilar* ($t \leftrightarrows_d u$) if in (*) one additionally requires $q_2 = q'$;

t and u are *branching bisimilar* ($t \leftrightarrows_b u$) if in (*) one requires both;

t and u are *strongly bisimilar* ($t \leftrightarrows u$) if in (*) one simply requires $q \xrightarrow{a} q'$.

Two processes t and u are *rooted weak bisimulation equivalent* ($t \leftrightarrows_{rw} u$), if they satisfy

if $t \xrightarrow{a} t'$ then $\exists u_1, u_2, u$ such that $u \Longrightarrow u_1 \xrightarrow{a} u_2 \Longrightarrow u'$ and $t' \leftrightarrows_w u'$, and
if $u \xrightarrow{a} u'$ then $\exists t_1, t_2, t$ such that $t \Longrightarrow t_1 \xrightarrow{a} t_2 \Longrightarrow t'$ and $t' \leftrightarrows_w u'$.

They are *rooted η-bisimilar* ($t \leftrightarrows_{r\eta} u$) if above one additionally requires $u_1 = u$, $t_1 = t$, and $t' \leftrightarrows_\eta u'$, they are *rooted delay bisimilar* ($t \leftrightarrows_{rd} u$) if one requires $u_2 = u'$, $t_2 = t'$ and $t' \leftrightarrows_d u'$, and they are *rooted branching bisimilar* ($t \leftrightarrows_{rb} u$) if one requires $u_1 = u$, $u_2 = u'$, $t_1 = t$, $t_2 = t'$ and $t' \leftrightarrows_b u'$.

It is well known and easy to check that the nine relations on processes defined above are equivalence relations indeed [2, 7], and that, for $x \in$ {weak, η, delay, branching, strong}, x-bisimulation equivalence is the largest x-bisimulation relation on processes. Moreover, $p \leftrightarrows_{rx} q$ implies $p \leftrightarrows_x q$.

Definition 6. An equivalence relation $\sim$ on processes is a *congruence* if

$$p_i \sim q_i \text{ for } i = 1, \ldots, ar(f) \quad \Rightarrow \quad f(p_1, \ldots, p_{ar(f)}) \sim f(q_1, \ldots, q_{ar(f)})$$

for all $f \in \Sigma$. This is equivalent to the requirement that for all $t \in \mathbb{T}(\Sigma)$ and closed substitutions $\sigma, \nu : V \rightarrow T(\Sigma)$

$$\sigma(x) = \nu(x) \text{ for } x \in var(t) \quad \Rightarrow \quad \sigma(t) = \nu(t).$$

This note, and Bloom [4], deal with syntactic conditions on GSOS languages that guarantee that the equivalence notions of Definition 5 are congruences.

2 Simply Cool GSOS Languages

In this section I define *simply XB cool* rule formats, for X$\in$ {W,D,H,B}, such that on XB cool GSOS languages, X-bisimulation equivalence is a congruence. In [5] it is shown that strong bisimulation equivalence is a congruence on any GSOS language. The proof is pretty straightforward; it consists of showing that the congruence-closure of $\leftrightarrows$ is a bisimulation. The same idea can be applied almost verbatim to $\leftrightarrows_w$, $\leftrightarrows_d$, $\leftrightarrows_\eta$ and $\leftrightarrows_b$, once we have lemmas like Lemma 1 below. The simply XB cool formats contain the simplest syntactic requirements that guarantee these lemmas to hold.

Definition 7. Let $\mathcal{L}$ be a positive GSOS language. For an operator f in $\mathcal{L}$, the *rules of* f are the rules in $\mathcal{L}$ with source $f(x_1,...,x_{ar(f)})$.

- An operator in $\mathcal{L}$ is *straight* if it has no rules in which a variable occurs multiple times in the left-hand side of its premises. An operator is *smooth* if moreover it has no rules in which a variable occurs both in the target and in the left-hand side of a premise.
- An argument $i \in \mathbb{N}$ of an operator f is *active* if f has a rule in which x_i appears as left-hand side of a premise.
- A variable x occurring in a term t is *receiving* in t if t is the target of a rule in $\mathcal{L}$ in which x is the right-hand side of a premise. An argument $i \in \mathbb{N}$ of an operator f is *receiving* if a variable x is receiving in a term t that has a subterm $f(t_1,\dots,t_n)$ with x occurring in t_i.
- A rule of the form $\dfrac{x_i \xrightarrow{\tau} y}{f(x_1,\dots,x_n) \xrightarrow{\tau} f(x_1,\dots,x_n)[y/x_i]}$ with $1 \leq i \leq n$ is called a *patience rule* for the i^{th} argument of f. Here $t[y/x]$ denotes term t with all occurrences of x replaced by y.

Definition 8. A GSOS language $\mathcal{L}$ is *simply WB cool* if it is positive and

1. all operators in $\mathcal{L}$ are straight,
2. patience rules are the only rules in $\mathcal{L}$ with τ-premises,
3. every active argument of an operator has a patience rule,
4. every receiving argument of an operator has a patience rule,
5. all operators in $\mathcal{L}$ are smooth.

The formats *simply DB cool*, *simply HB cool* and *simply BB cool* are defined likewise, but skipping Clause 4 for DB and BB, and Clause 5 for HB and BB.

The simply WB and BB cool formats above coincide with the ones of [4], whereas the simply DB cool format coincides with the `eb` format of [13].

Lemma 1. *Let $\mathcal{L}$ be simply WB cool, let $\frac{H}{s \xrightarrow{a} t}$ be a rule in $\mathcal{L}$, and let ν be a closed substitution such that $\nu(x) \Longrightarrow\xrightarrow{(c)}\Longrightarrow \nu(y)$ for each premise $x \xrightarrow{c} y$ in H. Then $\nu(s) \Longrightarrow\xrightarrow{(a)}\Longrightarrow \nu(t)$.*

Similar lemmas can be obtained for the other three formats, and these yield the following congruence results. The proofs are in the full version of this paper.

Theorem 1. *On any simply WB cool GSOS language, $\leftrightarroweq_w$ is a congruence.*
On any simply DB cool GSOS language, $\leftrightarroweq_d$ is a congruence.
On any simply HB cool GSOS language, $\leftrightarroweq_\eta$ is a congruence.
On any simply BB cool GSOS language, $\leftrightarroweq_b$ is a congruence.

3 Cool GSOS Languages

In this section I will extend the simply XB cool rule formats to XB cool rule formats and establish the associated congruence theorems (X$\in$ {W,D,H,B}).

Definition 9. A GSOS language is *two-tiered* if its operators are partitioned into *abbreviations* and *principal operators*, and for every abbreviation f a principal operator $f^\star$ is specified, together with a substitution $\sigma_f : \{x_1, \ldots, x_{ar(f^\star)}\} \rightarrow \{x_1, \ldots, x_{ar(f)}\}$, such that the rules of f are

$$\left\{ \frac{\sigma_f(H)}{f(x_1, \ldots, x_{ar(f)}) \xrightarrow{a} \sigma_f(t)} \;\middle|\; \frac{H}{f^\star(x_1, \ldots, x_{ar(f^\star)}) \xrightarrow{a} t} \text{ is a rule of } f^\star \right\}.$$

Write $f(i)$ for the j such that $\sigma_f(x_i) = x_j$; take $f^\star = f$ and $f(i) = i$ in case f is a principal operator.

Trivially, any positive GSOS language can be extended (*straightened*) to a two-tiered GSOS language whose principal operators are straight and smooth [1].

Example 1. Let $\mathcal{L}$ have an operator f with rule $\dfrac{x_1 \xrightarrow{a} y,\quad x_1 \xrightarrow{b} z}{f(x_1, x_2) \xrightarrow{a} f(x_1, (f(y, x_2))}$. $\mathcal{L}$ is straightened by adding a operator $f^\star$ with

$$\frac{x_1 \xrightarrow{a} y,\quad x_2 \xrightarrow{b} z}{f^\star(x_1, x_2, x_3, x_4) \xrightarrow{a} f(x_3, f(y, x_4))}.$$

In this case $\sigma_f(x_1) = \sigma_f(x_2) = \sigma_f(x_3) = x_1$ and $\sigma_f(x_4) = x_2$.

Equally trivial, $f^\star(p_{f(1)}, ..., p_{f(n)}) \xrightarrow{a} t$ iff $f(p_1, ..., p_n) \xrightarrow{a} t$;
so $f^\star(p_{f(1)}, ..., p_{f(n)}) \leftrightarroweq f(p_1, ..., p_n)$.

Definition 10. A two-tiered GSOS language $\mathcal{L}$ is *WB cool* if it is positive and

1. all principal operators in $\mathcal{L}$ are straight,
2. patience rules are the only rules of principal operators with τ-premises,
3. every active argument of a principal operator has a patience rule,
4. if argument $f(i)$ of f is receiving, then argument i of $f^\star$ has a patience rule,
5. all principal operators in $\mathcal{L}$ are smooth.

The formats *DB cool*, *HB cool* and *BB cool* are defined likewise, but skipping Clause 4 for DB and BB, and Clause 5 for HB and BB. Clause 4 may be weakened slightly; see Sect. 3.1.

Note that the simply cool formats defined before are exactly the cool formats with the extra restriction that all operators are principal.

Theorem 2. *On any WB cool GSOS language, $\leftrightarroweq_w$ is a congruence.*
On any DB cool GSOS language, $\leftrightarroweq_d$ is a congruence.
On any HB cool GSOS language, $\leftrightarroweq_\eta$ is a congruence.
On any BB cool GSOS language, $\leftrightarroweq_b$ is a congruence.

Given that the cool GSOS languages differ from the simply cool GSOS language only by the addition of operators that can be regarded as syntactic sugar, the theorems above are a simple consequence of the corresponding theorems for simply cool GSOS languages. Details are in the full version of this paper.

3.1 A Small Extension

Say that an argument i of an operator f is *ignored* if $f^\star$ has no argument k with $f(k) = i$. In that case there can be no rule with source $f(x_1, \dots, x_{ar(f)})$ with x_i in its premises or in its target. A subterm u of a term t is *irrelevant* if occurs within an ignored argument t_i of a subterm $f(t_1, \dots, t_{ar(f)})$ of t. Now Definition 7 of an argument of an operator being receiving may be strengthened by replacing "a subterm $f(t_1, \dots, t_n)$ with x occurring in t_i" by "a relevant subterm $f(t_1, \dots, t_n)$ with x a relevant subterm of t_i". This yields a slight weakening of Clause 4 in Definition 10, still sufficient to obtain Theorem 2.

Example 2. Let $\mathcal{L}$ have a rule $\dfrac{x_1 \xrightarrow{a} y}{g(x_1) \xrightarrow{a} f(h(f(x_1, y)), k(y))}$. By Definition 7 both the arguments of h and k are receiving, so Clause 4 in Definition 10 demands patience rules for both $h^\star$ and $k^\star$. Now suppose that $h^\star = h$, $k^\star = k$, $ar(f^\star) = 1$ and $\sigma_f(x_1) = x_1$. This means that $f(x_1, x_2)$ is an abbreviation for $f^\star(x_1)$ and the second argument of f is ignored. In such a case $f(p, q) \leftrightarroweq f(p, r)$ for all closed terms p, q and r. Now the weakened Clause 4 does not demand a patience rule for either $h^\star$ or $k^\star$, since the arguments of h and k are no longer receiving.

4 Rooted Cool GSOS Languages

In this section I will define the (simply) RWB, RDB, RHB and RBB cool rule formats and establish the associated congruence theorems. In order to formulate the requirements for the RWB and RDB cool GSOS languages I need the concept of a *ruloid*, this being a kind of derived GSOS rule.

Definition 11. For r transition rule, let RHS(r) denote the set of right-hand sides of its premises. Let $\mathcal{L}$ be a positive GSOS language. The class of $\mathcal{L}$-*ruloids* is the smallest set of rules such that

- $\frac{x \xrightarrow{a} y}{x \xrightarrow{a} y}$ is an $\mathcal{L}$-ruloid, for every $x, y \in V$ and $a \in Act$;
- if σ is a substitution, $\mathcal{L}$ has a rule $\frac{H}{s \xrightarrow{a} t}$, and for every premise $x \xrightarrow{c} y$ in H there is an $\mathcal{L}$-ruloid $r_y = \frac{H_y}{\sigma(x) \xrightarrow{c} \sigma(y)}$ such that the sets RHS(r_y) are pairwise disjoint and each RHS(r_y) is disjoint with $var(\sigma(s))$, then $\frac{\bigcup_{y \in H} H_y}{\sigma(s) \xrightarrow{a} \sigma(t)}$ is an $\mathcal{L}$-ruloid.

Note that a transition α, seen as a rule $\frac{\emptyset}{\alpha}$, is an $\mathcal{L}$-ruloid iff it is generated by $\mathcal{L}$ in the sense of Definition 4. The left-hand sides of premises of a ruloid are variables that occur in its source, and the right-hand sides are variables that are all distinct and do not occur in its source. Its target only contains variables that also occur elsewhere in the rule.

Example 3. Let $\mathcal{L}$ contain the rule $\dfrac{x_1 \xrightarrow{a} y_1 \quad x_2 \xrightarrow{b} y_2}{f(x_1, x_2) \xrightarrow{a} g(x_1, y_1)}$. Then $\mathcal{L}$ has ruloids

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{f(x, y) \xrightarrow{a} g(x, x')} \quad \text{and} \quad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y' \quad z \xrightarrow{b} z'}{f(f(x, y), z) \xrightarrow{a} g(f(x, y), g(x, x'))}.$$

Definition 12. A GSOS language $\mathcal{L}$ is *RWB cool* if the operators can be partitioned in *tame* and *wild* ones, such that

1. the target of every rule contains only tame operations;
2. the sublanguage $\mathcal{L}^{tame}$ of tame operators in $\mathcal{L}$ is WB cool;
3. $\mathcal{L}$ is positive, and for each rule $\frac{H}{s \xrightarrow{a} t}$ there is a term u and a substitution $\sigma : var(u) \rightarrow var(s)$ such that
   – there is an $\mathcal{L}$-ruloid $\frac{K}{u \xrightarrow{a} v}$ with $\sigma(K) = H$ and $\sigma(v) = t$,
   – and for every premise $x \xrightarrow{c} y$ in K, $\mathcal{L}$ has a rule $\frac{\sigma(x) \xrightarrow{\tau} y}{s \xrightarrow{\tau} \sigma(u[y/x])}$;

(4. if argument $f(i)$ of f is receiving, then argument i of $f^\star$ has a patience rule.)

The formats *RDB cool*, *RHB cool* and *RBB cool* are defined likewise, adapting "WB cool" in the second clause appropriately, but skipping the third clause for RHB and RBB, and the last one for RDB and RBB. The last clause cannot be skipped for RHB. The *simply RXB cool* rule formats (X$\in$ {W,D,H,B}) are obtained by requiring the sublanguage of tame operators to be simply XB cool.

Note that in the third clause, u, σ and the ruloid can always be chosen in such a way that $v = t$. The instance of this clause with $s = f(x_1, \ldots, x_{ar(f)})$ for a tame operator f is (in the full version of this paper) easily seen to be redundant.

The last clause above appeared before as Clause 4 in Definition 10 of the WB and HB cool formats. Given that a term with a receiving variable cannot contain wild operators, this clause is almost implied by Clause 2 above. All it adds, is that the requirement of Clause 4 for the sublanguage of tame operators applies to "receiving in $\mathcal{L}$" instead of merely "receiving in $\mathcal{L}^{tame}$". Thus, the rules for the wild operators help determine which variables in a term t count as receiving. The following results are obtained in the full version of this paper.

Proposition 1. *In the definition of RWB cool above, Clause 4 is redundant.*

Theorem 3. *On any RWB cool GSOS language, $\leftrightarroweq_{rw}$ is a congruence.*
On any RDB cool GSOS language, $\leftrightarroweq_{rd}$ is a congruence.
On any RHB cool GSOS language, $\leftrightarroweq_{r\eta}$ is a congruence.
On any RBB cool GSOS language, $\leftrightarroweq_{rb}$ is a congruence.

Example 4. The following fragment of CCS has the constant 0, unary operators $a.\_$, binary operators $+$ and $\|$, and instances of the GSOS rules below. Here a ranges over $Act = \mathcal{N} \dot{\cup} \overline{\mathcal{N}} \dot{\cup} \{\tau\}$ with $\mathcal{N}$ a set of *names* and $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$ the set of *co-names*. The function $\bar{\cdot}$ extends to $\mathcal{N} \cup \overline{\mathcal{N}}$ (but not to *Act*) by $\overline{\overline{a}} = a$.

$$\frac{x_1 \xrightarrow{a} y_1}{x_1 + x_2 \xrightarrow{a} y_1} \qquad \frac{x_2 \xrightarrow{a} y_2}{x_1 + x_2 \xrightarrow{a} y_2} \qquad a.x_1 \xrightarrow{a} x_1$$

$$\frac{x_1 \xrightarrow{a} y_1}{x_1 \| x_2 \xrightarrow{a} y_1 \| x_2} \qquad \frac{x_2 \xrightarrow{a} y_2}{x_1 \| x_2 \xrightarrow{a} x_1 \| y_2} \qquad \frac{x_1 \xrightarrow{a} y_1 \quad x_2 \xrightarrow{\overline{a}} y_2}{x_1 \| x_2 \xrightarrow{\tau} y_1 \| y_2}$$

The sublanguage without the $+$ is simply WB cool, and the entire GSOS language is simply RWB cool. Clause 3 of Definition 12 applied to the i^{th} rule for the $+$ is satisfied by taking $u = x$, $\sigma(x) = x_i$, and the ruloid $\frac{x \xrightarrow{a} y_i}{x \xrightarrow{a} y_i}$.

5 Comparison with Bloom's Formats

Bloom's definitions of the cool formats differ in five ways from mine.

First of all Bloom requires *bifurcation rules* for all operators in $\mathcal{L}^{tame}$, whereas I merely require patience rules for the principal operators. As principal operators in $\mathcal{L}^{tame}$ are straight, and bifurcation rules for straight operators are exactly patience rules, the difference is that I dropped the bifurcation requirement for abbreviations (non-principal operators). This is possible, because by Definition 9, which corresponds to Definition 3.5.5 in [4], the rules for the abbreviations are completely determined by the rules for their straightenings, and it turns out that a bifurcation rule of an abbreviation f is exactly what is determined by the corresponding patience rule for its straightening $f^\star$.

Bloom requires the existence of bifurcation/patience ruloids for receiving variables in any term, whereas I require them for receiving arguments of operators, which is a more syntactic and easy to check requirement. The two approaches are shown equivalent in the full version of this paper when using the extension of my formats of Sect. 3.1, this being the reason behind that extension.

Bloom's WB and RWB cool formats use a so-called *ε-presentation.* This entails that rules may have premises of the form $x \xrightarrow{\varepsilon} y$. In terms of Definition 4, the meaning of such premises is given by the requirement that $\sigma(x) = \sigma(y)$ for $(x \xrightarrow{\varepsilon} y) \in H$. By using ε-premises, any rule can be given a form in which the target is a univariate term, having no variables in common with the source. This allows a simplification of the statement of the bifurcation ruloids. Any ε-presented GSOS language can be converted to ε-free form by substitution, in each rule r, x for y for every premise $x \xrightarrow{\varepsilon} y$ of r. I believe that my conventions for naming variables improve the ones of [4].

Bloom's rendering of the RWB cool format doesn't feature Clause 4 (and in view of Prop. 1, neither does mine), but Clause 3 is much more involved. For every rule with conclusion $s \xrightarrow{a} t$ Bloom requires the existence of two terms t_1 and t_2 and seven types of derived operational rules, such that the diagram on the right commutes. My Clause 3 stems from the observation that, given Bloom's other restrictions, t necessarily has the rules required for t_2, so that one may always choose $t_2 = t$. This leaves only t_1 (called u in Definition 12) and three types of rules, one of which (the t_1-loop in the diagram above) is in fact a bifurcation rule whose existence is already implied by the requirements of Definition 10.

$$\begin{tikzcd} s \arrow[r, "a"] \arrow[d, "\tau"'] \arrow[dr, "a"] & t \arrow[d, "\tau"] \\ t_1 \arrow[r, "a"'] \arrow[ur, "a"] \arrow[loop left, "\tau"] & t_2 \arrow[loop right, "\tau"] \end{tikzcd}$$

In Clause 3 of Definition 12, Bloom requires that

$$var(u) = \{y' \mid y \in var(t)\} \text{ and } \sigma(y') = \begin{cases} x \text{ if } H \text{ contains a premise } x \xrightarrow{c} y \\ y \text{ otherwise.} \end{cases} \qquad (1)$$

In order to match Bloom's format I could have done the same, but this condition is not needed in the proof and reduces the generality of the format.

Proposition 2. *A GSOS language is WB cool, respectively RWB, BB or RBB cool, as defined here, with the extension of Sect. 3.1 and the restriction* (1) *above, iff it is WB cool, resp. RWB, BB or RBB cool, as defined in* Bloom [4]*.*

Moreover, my proofs that cool languages are compositional for bisimulation equivalences greatly simplify the ones of Bloom [4] by using a reduction of the general case to the simple case, instead of treating the general formats directly.

6 Turning GSOS Rules into Equations

This section recapitulates the method of [1] to provide finite equational axiomatisations of $\underline{\leftrightarrow}$ on an augmentation of any given GSOS language.

Definition 13. A process p, being a closed term in a GSOS language, is *finite* if there are only finitely many sequences of transitions $p \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} p_n$. The length n of the longest sequence of this form is called the *depth* of p.

Definition 14. An *equational axiomatisation* Ax over a signature Σ is a set of equations $t = u$, called *axioms*, with $t, u \in \mathbb{T}(\Sigma)$. It *respects* an equivalence relation $\sim$ on $T(\Sigma)$ if $\sigma(t) \sim \sigma(u)$ for any closed substitution $\sigma : V \rightarrow T(\Sigma)$.

An *instance* of axiom $t = u$ is an equation $\sigma(C[t/x]) = \sigma(C[u/x])$ where σ is a substitution and C a term with $var(C)=\{x\}$, and x occurring only once in C. An equation $p = q$ is *derivable* from Ax, notation $p =_{\mathrm{Ax}} q$, if there is a sequence $p_0, \ldots, p_n$ of terms with $n \geq 0$ such that $p = p_0$, $q = p_n$ and for $i = 1, \ldots, n$ the equation $p_{i-1} = p_i$ is an instance of one of the axioms.

Ax is *sound* for $\sim$ if $p =_{\mathrm{Ax}} q$ implies $p \sim q$ for $p, q \in T(\Sigma)$. Ax is *complete for $\sim$ on finite processes* if $p \sim q$ implies $p =_{\mathrm{Ax}} q$ for finite processes p and q.

Note that Ax is sound for $\sim$ iff Ax respects $\sim$ and $\sim$ is a congruence.

Definition 15. A GSOS language $\mathcal{L}$ *extends BCCS* (*basic* CCS) if it contains the operators 0, $a.\_$ and $+$ of Example 4.
A *basic process* is a closed term build from the operators mentioned above only.
A *head normal form* is a closed term of the form $0+a_1.p_1+\cdots+a_n.p_n$ for $n \geq 0$.
An axiomatisation on $\mathcal{L}$ is *head normalising* if any term $f(p_1, \ldots, p_{ar(f)})$ with the p_i basic processes can be converted into head normal form.

Proposition 3. *Let $\mathcal{L}$ be a GSOS language extending BCCS, and* Ax *a head normalising equational axiomatisation, respecting $\underline{\leftrightarrow}$, and containing the axioms* A1–4 *of Table 1. Then* Ax *is sound and complete for $\underline{\leftrightarrow}$ on finite processes.*

$x + (y + z) = (x + y) + z$	A1	$x \| y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \mid y$	CM1
$x + y = y + x$	A2	$a.x \mathbin{\lfloor\!\lfloor} y = a.(x \| y)$	CM2
$x + x = x$	A3	$0 \mathbin{\lfloor\!\lfloor} y = 0$	CM3
$x + 0 = x$	A4	$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$	CM4
		$a.x \mid \overline{a}.y = \tau.(x \| y)$	CM5
$a.(\tau.(x + y) + x) = a.(x + y)$	T1	$a.x \mid b.y = 0 \quad (\text{if } b \neq \overline{a})$	CM6
$\tau.x + x = \tau.x$	T2	$0 \mid x = x \mid 0 = 0$	CM7
$a.(\tau.x + y) + a.x = a.(\tau.x + y)$	T3	$(x + y) \mid z = x \mid z + y \mid z$	CM8
		$x \mid (y + z) = x \mid y + x \mid z$	CM9

Table 1. Complete equational axiomatisations of BCCS and the parallel composition

Proof. Using induction on the depth of p and a nested structural induction, the axioms can convert any finite process p into a basic process. Here one uses that strongly bisimilar processes have the same depth. Now apply the well-known fact that the axioms A1–4 are sound and complete for $\leftrightarrows$ on basic processes [8].

For the parallel composition operator $\|$ of CCS no finite equational head normalising axiomatisation respecting strong bisimulation equivalence exists [9]. However, Bergstra & Klop [3] gave such an axiomatisation on the language obtained by adding two auxiliary operators, the *left merge* $\mathbin{\|\!\_}$ and the *communication* merge $|$, with rules $\frac{x_1 \xrightarrow{a} y_1}{x_1 \mathbin{\|\!\_} x_2 \xrightarrow{a} y_1 \| x_2}$ and $\frac{x_1 \xrightarrow{a} y_1 \quad x_2 \xrightarrow{\overline{a}} y_2}{x_1 | x_2 \xrightarrow{\tau} y_1 \| y_2}$, provided the alphabet *Act* of actions is finite. The axioms are CM1–9 of Table 1, in which $+$ binds weakest and $a.\_$ strongest, and a, b range over *Act*.

Aceto, Bloom & Vaandrager [1] generalise this idea to arbitrary GSOS languages with finitely many rules, each with finitely many premises, and assuming a finite alphabet *Act*. I recapitulate their method for positive languages only. A smooth operator (Definition 7) only has rules of the form $\frac{\{x_i \xrightarrow{c_i} y_i \mid i \in I\}}{f(x_1,\ldots,x_n) \xrightarrow{a} t}$. The *trigger* of such a rule is the partial function $\uparrow_r$: $\{i,\ldots,n\} \rightharpoonup Act$ given by $\uparrow_r(i) = c_i$ if $i \in I$, and $\uparrow_r(i)$ is undefined otherwise.

Definition 16. [1] A smooth GSOS operator f is *distinctive*, if no two rules of f have the same trigger, and the triggers of all rules of f have the same domain.

All operators of CCS, as well as $\mathbin{\|\!\_}$ and $|$, are smooth. The operators 0, $a.\_$, $\mathbin{\|\!\_}$ and $|$ are distinctive, but $\|$ is not. Its triggers have domains $\{1\}$, $\{2\}$ and $\{1,2\}$.

For every smooth and distinctive operator f, Aceto, Bloom & Vaandrager declare four types of axioms. First of all, for every rule r as above there is an axiom $f(\sigma(x_1),\ldots,\sigma(x_n)) = a.\sigma(t)$, where $\sigma : \{x_1,\ldots,x_n\} \rightarrow \mathbb{T}(\Sigma)$ is the substitution given by $\sigma(x_i) = c_i.y_i$ for $i \in I$ and $\sigma(x_i) = x_i$ for $i \notin I$. Such an axiom is called an *action law*. Examples are CM2 and CM5 in Table 1.

Secondly, whenever I is the set of active arguments of f, but f has no rule of the form above (where the name of the variables y_i is of no importance), there is an axiom $f(\sigma(x_1),\ldots,\sigma(x_n)) = 0$, with σ as above (for an arbitrary choice of distinct variables y_i). Such an axiom is an *inaction law*. An example is CM6. If f has k active arguments, in total there are $|Act|^k$ action and inaction laws for f, one for every conceivable trigger with as domain the active arguments of f.

Finally, for any active argument i of f, there are laws

$$f(x_1,\ldots x_{i-1},0,x_{i+1},\ldots,x_n) = 0 \qquad \text{and}$$
$$f(x_1,\ldots,x_i+x_i',\ldots,x_n) = f(x_1,\ldots,x_i,\ldots,x_n) + f(x_1,\ldots,x_i',\ldots,x_n).$$

Examples for the second type of inaction law are CM3 and CM7, and examples of *distributivity laws* are CM4, CM8 and CM9.

It is not hard to see that all axioms above respect $\leftrightarrows$ and that together they bring any term $f(p_1,\ldots,p_{ar(f)})$ with the p_i basic processes in head normal form.

The method of [1] makes three types of additions to a given finite GSOS language $\mathcal{L}$, and provides an equational head normalising axiomatisation on the resulting language, that respects strong bisimulation.

First of all, the operators 0, $a.\_$ and $+$ are added, if not already there. The corresponding axioms are A1–4 of Table 1. If all other operators are smooth and distinctive, for each of them the axioms just described are taken, which finishes the job. (In the presence of negative premises, this step is slightly more complex.)

In case there are operators f that are smooth but not distinctive, the set of operational rules of f is partitioned into subsets D such that no two rules in D have the same trigger, and the triggers of all rules in D have the same domain. Note that such a partition can always be found—possibly by taking exactly one rule in each subset D. Now for any subset D in the partition, an operator f_D with $ar(f_D) = ar(f)$ is added to the language, whose rules are exactly the rules in that subset, but with f_D in the source. By definition, f_D is distinctive. Now add an axiom $f(x_1, \ldots, x_{ar(f)}) = \sum f_D(x_1, \ldots, x_{ar(f)})$, where the sum is taken over all subsets in the partition, and apply the method above to the operators f_D. Again, it is trivial to check that the axioms respect $\leftrightarroweq$ and are head normalising. Applied to the $\|$ of CCS, this technique yields the left merge and communication merge as auxiliary operators, as well as a right merge, and the axiom CM1.

In case of operators f that are not smooth, a smooth operator $f^\star$ is added to $\mathcal{L}$, of which f is an abbreviation in the sense of Definition 9 (cf. Example 1). The treatment of $f^\star$ proceeds as above, and the project is finished by the axiom

$$f(p_1, ..., p_n) = f^\star(p_{f(1)}, ..., p_{f(n)}).$$

Besides completeness for finite processes, using an infinitary induction principle the method of [1] even yields completeness for arbitrary processes. I will not treat this here, as it does not generalise to weak equivalences.

7 Turning Cool GSOS Rules into Equations

The method of [1] does not apply to $\leftrightarroweq_w$, $\leftrightarroweq_d$, $\leftrightarroweq_\eta$, and $\leftrightarroweq_b$, because these equivalences fail to be congruences for the $+$. However, Bloom [4] shows that the method applies more or less verbatim to $\leftrightarroweq_{rb}$. This section observes that the same holds for $\leftrightarroweq_{r\eta}$, and finds an adaptation to yield finite equational axiomatisations of $\leftrightarroweq_{rw}$ (resp. $\leftrightarroweq_{rd}$) that are sound and complete for finite processes on an augmentation of any RWB cool (resp. RDB cool) GSOS language.

On basic processes, the axioms A1–4 together with T1–T3 are complete for $\leftrightarroweq_{rw}$ [8], whereas complete axiomatisations for $\leftrightarroweq_{rd}$, $\leftrightarroweq_{r\eta}$ and $\leftrightarroweq_{rb}$ are obtained by dropping T3, T2 or both, respectively [7]. So in order to get axiomatisations of these equivalences that are complete for finite processes, all that is needed is head normalisation. The simplest approach is to use the same head normalising axioms as in the previous section, reasoning that axioms that respect $\leftrightarroweq$ surely respect a coarser equivalence like $\leftrightarroweq_{rb}$ or $\leftrightarroweq_{rw}$. The only way this approach could fail is when the auxiliary operators generated by [1] fail to be congruences for the equivalence relation at hand. The operators 0, $a.\_$ and $+$ are WB cool, and thus unproblematic. As observed in [4], for any RBB cool GSOS language, the augmented language is also RBB cool. Namely, the new operators do not show up in targets of new rules, so classifying all auxiliary operators as wild is sufficient. Since the auxiliary operators do not increase the collection of receiving arguments of operators either, it follows likewise that for any RHB cool GSOS language, the augmented language is also RHB cool. Hence one obtains

Proposition 4. *The method of [1], together with axiom T1 (and T3), yields finite equational axiomatisations of $\leftrightarroweq_{rb}$ (resp. $\leftrightarroweq_{r\eta}$) that are sound and complete for finite processes on an augmentation of any RBB cool (resp. RHB cool) GSOS language.*

For $\leftrightarroweq_{rw}$ and $\leftrightarroweq_{rd}$ this approach fails. In particular, these equivalences fail to be congruences for the communication merge: one has $\tau.a.0 \leftrightarroweq_{rd} \tau.a.0 + a.0$ but

$$0 \leftrightarroweq (\tau.a.0|\overline{a}.b.0) \not\leftrightarroweq_{rd} ((\tau.a.0 + a.0)|\overline{a}.b.0) \leftrightarroweq \tau.b.0.$$

Conjecture. *There exists no GSOS language including the parallel composition of CCS and ≥ 2 visible actions that admits a finite equational axiomatisation of weak bisimulation equivalence that is sound and complete for finite processes.*

Nevertheless, such an axiomatisation was found by Bergstra & Klop [3], using a variant of the communication merge that is not a GSOS operator. Their axiomatisation of $\|$ is obtained from the one in Table 1 by requiring $a, b \neq \tau$ in CM6, and adding the axioms $\tau.x|y = x|\tau.y = x|y$. Here I generalise their approach to arbitrary RWB cool (or RDB cool) GSOS languages.

The RWB cool format can be extended by allowing wild operators f, besides GSOS rules satisfying Clause 3 of Definition 12, also to have rules of which all premises have the form $x \Longrightarrow\xrightarrow{c} y$ with $c \in A$. For such rules Clause 3 is not required, but in fulfilling Clause 4, they do count in determining which arguments are receiving. A similar extension applies to the RDB cool format.

Theorem 4. *On any extended-RWB cool TSS, $\leftrightarroweq_{rw}$ is a congruence.*
On any extended-RDB cool TSS, $\leftrightarroweq_{rd}$ is a congruence.

In an RWB (or RDB) cool language, the smooth operators $f^\star$ that are needed to axiomatise a non-smooth operator f are unproblematic. For tame operators f, they are already in the language, and for a wild f it is not hard to define them in such a way that the augmented language remains RWB (or RDB) cool. Of the operators f_D needed to axiomatise a non-distinctive operator f, those that have exactly one active argument can be made to satisfy Clause 3 of Definition 12 by including the relevant τ-rule in D. All operators f_D with another number of active arguments cannot have τ-premises, by Definitions 12 and 10. These operators f_D are replaced by counterparts f'_D, obtained by replacing each premise $x \xrightarrow{c} y$ in a rule for f_D by $x \Longrightarrow\xrightarrow{c} y$. By Theorem 4, $\leftrightarroweq_{rw}$ (or $\leftrightarroweq_{rd}$) is a congruence for f'_D. Furthermore, $f(x_1,\ldots,x_{ar(f)}) \leftrightarroweq_{rw} \sum f'_D(x_1,\ldots,x_{ar(f)})$. Now the required axiomatisation is obtained by omitting all inaction laws for the modified operators f'_D with $\sigma(x_i) = \tau.y_i$ for some active argument i, and instead adding *τ-laws* $f'_D(x_1,\ldots,\tau.x_i,\ldots,x_n) = f'_D(x_1,\ldots,x_i,\ldots,x_n)$.

8 A Challenge

All equivalences of Definition 5 are congruences of the GSOS language with rules

$$\frac{x_1 \xrightarrow{a} y}{f(x_1) \xrightarrow{a} g(y)} \qquad \frac{x_1 \xrightarrow{\tau} y}{g(x_1) \xrightarrow{\tau} g(y)} \qquad g(x_1) \xrightarrow{\tau} !x_1$$

$$\frac{x_1 \xrightarrow{a} y}{!x_1 \xrightarrow{a} y \| !x_1} \qquad \frac{x_1 \xrightarrow{a} y_1}{x_1 \| x_2 \xrightarrow{a} y_1 \| x_2} \qquad \frac{x_2 \xrightarrow{a} y_2}{x_1 \| x_2 \xrightarrow{a} x_1 \| y_2}$$

for $a \in Act$. Here, the operator $!x$ can be understood as a parallel composition of infinitely many copies of x. The rules for f, g and $\|$ are WB cool, but the one for ! is not. It is not even RBB safe in the sense of [6].

Open problem. Find a congruence format that includes the language above.

References

1. L. Aceto, B. Bloom & F.W. Vaandrager (1994): *Turning SOS rules into equations. Information and Computation* 111(1), pp. 1–52.
2. T. Basten (1996): *Branching bisimulation is an equivalence indeed! Information Processing Letters* 58(3), pp. 141–147.
3. J.A. Bergstra & J.W. Klop (1985): *Algebra of communicating processes with abstraction. Theoretical Computer Science* 37(1), pp. 77–121.
4. B. Bloom (1995): *Structural operational semantics for weak bisimulations. Theoretical Computer Science* 146, pp. 25–68.
5. B. Bloom, S. Istrail & A.R. Meyer (1995): *Bisimulation can't be traced. Journal of the ACM* 42(1), pp. 232–268.
6. W.J. Fokkink (2000): *Rooted branching bisimulation as a congruence. Journal of Computer and System Sciences* 60(1), pp. 13–37.
7. R.J. van Glabbeek & W.P. Weijland (1996): *Branching time and abstraction in bisimulation semantics. Journal of the ACM* 43(3), pp. 555–600.
8. R. Milner (1990): *Operational and algebraic semantics of concurrent processes.* In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, chapter 19, Elsevier Science Publishers B.V. (North-Holland), pp. 1201–1242. Alternatively see *Communication and Concurrency*, Prentice-Hall International, Englewood Cliffs, 1989, or *A Calculus of Communicating Systems*, LNCS 92, Springer-Verlag, 1980.
9. F. Moller (1990): *The nonexistence of finite axiomatisations for CCS congruences.* In *Proceedings 5th Annual Symposium on Logic in Computer Science,* Philadelphia, USA, IEEE Computer Society Press, pp. 142–153.
10. G.D. Plotkin (2004): *A structural approach to operational semantics. The Journal of Logic and Algebraic Programming* 60–61, pp. 17–139. First appeared in 1981.
11. R. de Simone (1985): *Higher-level synchronising devices in* Meije-*SCCS. Theoretical Computer Science* 37, pp. 245–267.
12. I. Ulidowski (1992): *Equivalences on observable processes.* In *Proceedings 7th Annual Symposium on Logic in Computer Science,* Santa Cruz, California, IEEE Computer Society Press, pp. 148–159.
13. I. Ulidowski & I. Phillips (2002): *Ordered SOS rules and process languages for branching and eager bisimulations. Information & Computation* 178, pp. 180–213.
14. I. Ulidowski & S. Yuen (2000): *Process languages for rooted eager bisimulation.* In C. Palamidessi, editor: Proceedings of the 11th International Conference on *Concurrency Theory*, CONCUR 2000, LNCS 1877, Springer, pp. 275–289.