Weak bisimulation as a congruence in MSOS

Technical Report

Version 0.9

Peter D. Mosses and Ferdinand Vesely

Swansea University, Swansea, SA2 8PP, United Kingdom
p.d.mosses@swansea.ac.uk, csfvesely@swansea.ac.uk

Abstract. MSOS is a variant of structural operational semantics with a natural representation of unobservable transitions. To prove various desirable laws for programming constructs specified in MSOS, bisimulation should disregard unobservable transitions, and it should be a congruence. One approach, following Van Glabbeek, is to add abstraction rules and use strong bisimulation with MSOS specifications in an existing congruence format. Another approach is to use weak bisimulation with specifications in an adaptation of Bloom's WB Cool congruence format to MSOS. We compare the two approaches, and relate unobservable transitions in MSOS to equations in Rewriting Logic.

Table of Contents

1 Introduction

Rewriting logic comes with a built-in "abstraction dial" [26]. The least abstract position of the dial is when the specification consists entirely of rules, with no equations (apart from structural axioms such as associativity, etc.). Turning rules into equations increases the degree of abstraction: the remaining rules specify rewrites on equivalence classes of terms. The set of equations is required to be confluent, so semantic rules for nondeterministic or concurrent constructs cannot be replaced by equations.

In structural operational semantics (SOS) [1,35,36], specifications usually consist entirely of transition rules. Abstraction can be introduced by regarding particular transitions as *unobservable* (conventionally using the label τ) and ignoring unobservable transitions when defining behavioural equivalences. Confluence of unobservable transitions is not required: nondeterministic choice can itself be unobservable. So-called 'structural congruence' equations such as associativity and commutativity are used in the SOS of the π-calculus [28], but their introduction has a non-trivial impact on the meta-theory of SOS [33]; moreover, they are not as general as equations in Rewriting Logic, nor do they subsume the use of unobservable transitions in SOS.

Modular structural operational semantics (MSOS) [29,30,31] is a variant of SOS with a particularly natural representation of unobservable transitions: labels are the morphisms of a category, and unobservable transitions are those labelled by identity morphisms. This lets us specify unobservable transitions in MSOS without introducing extra labels such as τ.

For compositional reasoning about behavioural equivalence in SOS and MSOS, bisimulation needs to be a congruence. Various rule formats ensuring congruence have been established for SOS [34]. Recently, a rule format was given for MSOS such that strong bisimulation is a congruence [7]. MSOS was there extended with an unlabelled rewriting relation, which was required to be a precongruence, and used to represent unobservable evaluation steps in the definition of strong bisimulation. That is sufficient for proving some simple context-independent absorption laws, such as the null command being a left and right unit for command sequencing. However, we want to be able to prove also that various program optimisations preserve observable behaviour, and the precongruence is too restrictive for that purpose.

This led us to investigate how to ignore unobservable transitions in connection with bisimulation in MSOS, while ensuring that it remains a congruence. After recalling established concepts, notation and results (Sect. 2), we adopt a technique proposed by Van Glabbeek [10,12] (Sect. 3): we add abstraction rules for the unobservable transitions, and use strong bisimulation with MSOS specifications in an existing congruence format [7]. However, that approach requires the introduction of stuttering rules, which might be seen as a drawback (e.g., in connection with executability) and motivates an alternative approach (Sect. 4): to use weak bisimulation with MSOS specifications in an adaptation of Bloom's WB Cool congruence format. We conclude (Sect. 5) by comparing the

two approaches. Our main contribution is establishing adequate techniques for disregarding unobservable transitions in MSOS.

This paper was written for the Festschrift in honor of José Meseguer. PDM would like to express the following personal appreciation of José and his research.

As recalled in [23], José and I first met almost 40 years ago, in 1976, in Oxford. I soon became keenly interested in the initial algebra approach to specification of abstract data types, and followed the development of the seminal OBJ system for executing algebraic specifications, initiated by Joseph Goguen [14] and continued through the 1980s in a fruitful collaboration between Joseph, José and their colleagues [9,13,19]. Order-sorted algebra [15,16,17,25] was a further topic of intense common interest. José is also a founding member of IFIP WG 1.3 (Foundations of System Specification); in that connection he chaired an expert panel that assessed the design of CASL [2,32].

The breadth and depth of José's research is clearly reflected by the hundreds of carefully written journal and conference papers that he has published. A particularly prominent topic since the beginning of the 1990s is Rewriting Logic [21,22] and its implementation in Maude [8]. When I took a sabbatical in 1998–99, José invited me to visit him at SRI International, and obtained funding (with Carolyn Talcott) for a joint research project during the visit. José's group was a stimulating environment for the initial development of MSOS [29,30,31]. In particular, I recall that José made crucial observations concerning the possibility of treating labels in MSOS as morphisms of categories, and we subsequently co-authored two papers (together with Christiano Braga and Hermann Haeusler) [5,6] on an embedding of MSOS in Rewriting Logic. The recent progress report by José (with Grigore Roşu) on the Rewriting Logic Semantics Project [26] and his review of 20 years of Rewriting Logic [24] reflect his inspiration to a multitude of colleagues, as well as his own contributions. I look forward to following his ongoing research, and to many future meetings.

2 Background

This section serves as an overview of preliminaries regarding SOS, rule formats and MSOS. Everywhere in this paper we assume a set of variables $\mathcal{X}$ with typical elements x, y and z, optionally subscripted.

2.1 SOS

Structural operational semantics (SOS) [1,35,36] is a framework for giving operational semantics of specification and programming languages.

In the most general case, SOS computations are modelled by *labelled terminal transition systems*.

Definition 1 (labelled terminal transition system). *A* labelled terminal transition system (LTTS) *is a tuple* $\langle \Gamma, L, \longrightarrow, T \rangle$*, where* $\gamma \in \Gamma$ *are configurations (or states),* $l \in L$ *are labels,* $\longrightarrow \subseteq \Gamma \times L \times \Gamma$ *is a transition relation, and* $T \subseteq \Gamma$ *a set of final configurations. An assertion of an* l*-transition* $\langle \gamma, l, \gamma' \rangle \in \longrightarrow$ *is written* $\gamma \xrightarrow{l} \gamma'$ *and we say that* γ *is the* source *and* γ' *the* target*, or that* γ' *is an* l*-derivative of* γ*. If* $\gamma \in T$ *then there is no* $\gamma' \in \Gamma$ *and* $l \in L$ *such that* $\gamma \xrightarrow{l} \gamma'$*.*

A computation *in an LTTS is a finite or infinite sequence of transitions* $\gamma_0 \xrightarrow{l_0} \gamma_1 \xrightarrow{l_1} \cdots$ *such that if the sequence terminates with* γ_n*, then* $\gamma_n \in T$*.*

In SOS definitions of process calculi, states are usually just closed terms generated over an algebraic signature Σ, which is a collection of function symbols (operators) and their arities. For programming languages, the terms may contain computed values, and states usually contain additional *auxiliary entities* such as stores or environments.

The transition relation of an LTTS is generated inductively by a set of SOS rules of the form $\frac{H}{c}$, where H is a (possibly empty) set of premises and c is the conclusion. If H is empty, the rule is an *axiom*, otherwise it is a *conditional rule*. Premises and the conclusion are formulas $t \xrightarrow{l} t'$, where t and t' are configuration terms optionally containing variables. The intuitive meaning of a rule is that if all premises in H are satisfied by some closing substitution then so is the conclusion. Usually, premises will assert transitions for subterms of t, giving the rules a structural character.

2.2 Bisimulation and Rule Formats

For labelled transition systems, the finest useful notion of equivalence between states [37] is *strong bisimulation*. Briefly, a strong bisimulation relates two states if they both can make transitions with the same label to states that are again related; final states must be equal.

Definition 2 (Strong bisimulation on LTTS). *Given an LTTS* $\langle \Gamma, L, \longrightarrow, T \rangle$*, a* symmetric *relation* $\mathcal{R} \subseteq \Gamma \times \Gamma$ *is a* strong bisimulation *if whenever* $\gamma_1 \mathcal{R} \gamma_2$ *then* $\gamma_1 \xrightarrow{l} \gamma_1'$ *implies that there is a* γ_2' *such that* $\gamma_2 \xrightarrow{l} \gamma_2'$ *and* $\gamma_1' \mathcal{R} \gamma_2'$*; and* $\gamma_1 \in T$ *implies* $\gamma_1 = \gamma_2$*.*

Two configurations γ_1 *and* γ_2 *are* strongly bisimilar*, written* $\gamma_1 \sim \gamma_2$ *iff* $\gamma_1 \mathcal{R} \gamma_2$ *for some strong bisimulation* $\mathcal{R}$*.*

For equational reasoning, an important property of bisimulation is *congruence*. A bisimulation is a congruence if it is preserved by all operators in the language. In general, strong bisimulation is not necessarily a congruence: terms whose subterms are bisimilar might not be bisimilar themselves. While for small languages one can check the congruence property by checking all constructs in the language, a more principled way is to make all rules adhere to a particular *congruence format*. The *positive GSOS* format [4] is a congruence format for SOS; we adapt it to MSOS in Sect. 4. A survey of congruence formats and associated meta-theory can be found in [34].

Definition 3 (Positive GSOS). *A rule for a construct* f *with arity* n *is in the positive GSOS format if it has the form*

$$\frac{\{x_i \xrightarrow{l_j} y_j \mid i \in I, j \in J_i\}}{f(x_1, \ldots, x_n) \xrightarrow{l} t}$$

where $I \subseteq \{1, \ldots, n\}$, J_i is a finite index set for each i, and all x_i and y_j are distinct variables. Only variables x_i and y_j may appear in t. A language definition is in the positive GSOS format if all its rules are.

The full GSOS format allows also rules with negative premises. A congruence theorem for GSOS states that if all rules in a specification are in the GSOS format, then strong bisimulation is guaranteed to be a congruence. This theorem was first proved by Bloom et al. [4]; a more succinct proof was subsequently provided by Van Glabbeek [11].

Traditionally, a designated *silent* label 'τ' is used for transitions, such as selecting a branch in a conditional statement, that correspond to internal housekeeping actions, and are of no inherent interest.

Strong bisimulation does not treat a τ-label differently from other labels: a τ-transition of one state has to be matched with a τ-transition of the other state. This results in a notion of equivalence which is too strong for many purposes.

One approach is to modify the transition relation by adding *abstraction rules* to a specification. This approach is explored by Van Glabbeek in [10,12] by adding special τ-rules. These rules allow hiding of silent transitions, or allow an arbitrary number of silent transitions *after* an observable transition has been made. We adapt this approach to our needs in Sect. 3.

The more common approach is to weaken bisimulation itself: *weak bisimulation* is a notion of bisimulation that permits ignoring τ-transitions.

Definition 4 (Weak bisimulation on LTTS). *Assume an LTTS $\langle \Gamma, L, \longrightarrow, T \rangle$. A* symmetric *relation $\mathcal{R} \subseteq \Gamma \times \Gamma$ is a* weak bisimulation *if whenever $\gamma_1 \mathcal{R} \gamma_2$ then $\gamma_1 \xrightarrow{l} \gamma_1'$ implies that there is a γ_2' such that $\gamma_2 \longrightarrow^* \xrightarrow{(l)} \longrightarrow^* \gamma_2'$ and $\gamma_1' \mathcal{R} \gamma_2'$; and $\gamma_1 \in T$ implies $\gamma_1 = \gamma_2$. Here $\gamma \xrightarrow{(l)} \gamma'$ iff $l = \tau$ and $\gamma = \gamma'$, or $l \neq \tau$ and $\gamma \xrightarrow{l} \gamma'$; and $\longrightarrow^*$ is the reflexive transitive closure of $\xrightarrow{\tau}$.*

Two configurations γ_1 and γ_2 are weakly bisimilar*, written $\gamma_1 \approx \gamma_2$ iff $\gamma_1 \mathcal{R} \gamma_2$ for some weak bisimulation $\mathcal{R}$.*

Weak bisimulation has less desirable congruence properties than the strong variant. Even when a language is in the positive GSOS format, weak bisimulation is not guaranteed to be a congruence. The Cool formats [3] impose further restrictions on positive GSOS to guarantee that weak bisimulation is a congruence. These resetrictions prevent constructs in a language from distinguishing between states based on the number of silent transitions they can make. We spell out the restrictions, adjusted for MSOS, in Sect. 4.

2.3 Modular SOS

Modular SOS (MSOS) [31] improves the modularity of SOS definitions. This is achieved by moving auxiliary entities from configurations to labels of transitions, and letting labels be morphisms of a category. Label components, each corresponding to an auxiliary entity such as environments and stores, are combined using an indexed product, which can also be regarded as a finite map or record.

Labels on adjacent transitions are required to be composable. This, together with a record pattern notation akin to that of Standard ML, allows rules to only mention entities that are required in specifying a particular transition, while other entities are left unmentioned.

An MSOS specification consists of an algebraic signature, a label profile and a set of rules. The signature specifies how terms can be formed from a set of constructors, the label profile specifies the label entities used in the specification, and rules define the behaviour of terms formed from constructors. We provide an example set of rules in Table 1. Since information about auxiliary entities is moved to the labels, configurations in MSOS are limited to program terms and computed values (value-added syntax). The rules define a transition system with closed terms as configurations and composites of entity morphisms as labels. We give more details on the facets of MSOS specifications, how they specify computations, and how equalities between terms can be established, in the following subsections.

2.4 MSOS Terms

A basic single-sorted signature is a collection of function symbols (operators, constructors) together with a function assigning an arity to each of these symbols. The terms in the language form a freely generated algebra over the signature. In this paper we adopt an extension of the basic signature to a *value-computation signature* [7], which distinguishes a set of value constructors. This provides an alternative to specifying a set of final states of a terminal transition system – the set of final states (values) is the set of terms where the outermost constructor is a value constructor. In the example of Table 1, the boolean values **true** and **false** are nullary value constructors and so is **skip**, which represents the successful termination of a command. Further values assumed by the rules are identifiers and locations, which are used as indices for environments and stores, respectively, and which we have left unspecified in this example.

Definition 5 (value-computation signature). *A* value-computation signature (vc-signature) Σ *consists of a set of constructors* C_Σ*, each with an arity* $\mathrm{ar}_\Sigma : C_\Sigma \to \mathbb{N}$*, and a set of value constructors* $VC_\Sigma \subseteq C_\Sigma$*.*

For a signature Σ*,* $\mathbb{T}_\Sigma$ *is the set of open terms generated by the set* $\mathcal{X}$ *of variables,* $T_\Sigma \subseteq \mathbb{T}_\Sigma$ *the set of closed terms,* $\mathrm{vars}(t) \subseteq \mathcal{X}$ *is the set of variables in a term* t*, and* $V_\Sigma \subseteq T_\Sigma$ *is the set of* value terms *whose outermost constructor is in* VC_Σ*.*

A Σ-substitution *is a partial function* $\sigma : \mathcal{X} \to \mathbb{T}_\Sigma$ *mapping variables to terms. The domain of a substitution* σ *is written* $\mathrm{dom}(\sigma)$*. A substitution is* closing *for a term* $t \in \mathbb{T}_\Sigma$ *if* $\mathrm{vars}(t) \subseteq \mathrm{dom}(\sigma)$ *and its image is a subset of the closed terms* T_Σ*.*

2.5 MSOS Labels

MSOS transition labels are aggregates of label components which are morphisms of their respective categories. The aggregate is formed as an indexed product or

$$\frac{s_1 \xrightarrow{\{\ldots\}} s_1'}{\mathsf{if}(s_1, s_2, s_3) \xrightarrow{\{\ldots\}} \mathsf{if}(s_1', s_2, s_3)} \quad (1) \qquad \frac{}{\mathsf{if}(\mathsf{true}, s_2, s_3) \xrightarrow{\{-\}} s_2} \quad (2)$$

$$\frac{}{\mathsf{if}(\mathsf{false}, s_2, s_3) \xrightarrow{\{-\}} s_3} \quad (3) \qquad \frac{s_1 \xrightarrow{\{\ldots\}} s_1'}{\mathsf{let}(i, s_1, s_2) \xrightarrow{\{\ldots\}} \mathsf{let}(i, s_1', s_2)} \quad (4)$$

$$\frac{s_2 \xrightarrow{\{\mathbf{env}=\rho[i\mapsto v_1],\ldots\}} s_2'}{\mathsf{let}(i, v_1, s_2) \xrightarrow{\{\mathbf{env}=\rho,\ldots\}} \mathsf{let}(i, v_1, s_2')} \quad (5)$$

$$\frac{}{\mathsf{let}(i, v_1, v_2) \xrightarrow{\{-\}} v_2} \quad (6) \qquad \frac{\rho(i) = v}{\mathsf{bound}(i) \xrightarrow{\{\mathbf{env}=\rho,-\}} v} \quad (7)$$

$$\frac{s \xrightarrow{\{\ldots\}} s'}{\mathsf{print}(s) \xrightarrow{\{\ldots\}} \mathsf{print}(s')} \quad (8) \qquad \frac{}{\mathsf{print}(v) \xrightarrow{\{\mathbf{out}'=v,-\}} \mathsf{skip}} \quad (9)$$

$$\frac{s_1 \xrightarrow{\{\ldots\}} s_1'}{\mathsf{assign}(s_1, s_2) \xrightarrow{\{\ldots\}} \mathsf{assign}(s_1', s_2)} \quad (10) \qquad \frac{s_2 \xrightarrow{\{\ldots\}} s_2'}{\mathsf{assign}(s_1, s_2) \xrightarrow{\{\ldots\}} \mathsf{assign}(s_1, s_2')} \quad (11)$$

$$\frac{}{\mathsf{assign}(v_1, v_2) \xrightarrow{\{\mathbf{sto}=\sigma,\mathbf{sto}'=\sigma[v_1\mapsto v_2],-\}} \mathsf{skip}} \quad (12)$$

$$\frac{s \xrightarrow{\{\ldots\}} s'}{\mathsf{stored}(s) \xrightarrow{\{\ldots\}} \mathsf{stored}(s')} \quad (13) \qquad \frac{\sigma(v_1) = v_2}{\mathsf{stored}(v_1) \xrightarrow{\{\mathbf{sto}=\sigma,\mathbf{sto}'=\sigma,-\}} v_2} \quad (14)$$

$$\frac{s_1 \xrightarrow{\{\ldots\}} s_1'}{\mathsf{seq}(s_1, s_2) \xrightarrow{\{\ldots\}} \mathsf{seq}(s_1', s_2)} \quad (15) \qquad \frac{}{\mathsf{seq}(\mathsf{skip}, s_2) \xrightarrow{\{-\}} s_2} \quad (16)$$

Table 1: An example language defined in MSOS. Meta-variables v range over values and s over arbitrary terms. Environments ρ and stores σ are partial maps.

finite map from label indices (for example, **sto** for stores or **env** for environments) to morphisms. The full label is composable only if it is composable componentwise. The morphisms-as-labels aspect of MSOS means that it has a very natural representation of *unobservable transitions*: instead of having a special label, conventionally designated as silent, unobservable transitions in MSOS are simply those labelled by *identity morphisms*.

The usual auxiliary entities used in programming language semantics are covered by three kinds of label categories:

- A category for *read-only* entities, such as environments ρ, will have just an identity morphism for each object; then two morphisms are composable only if they are identical.
- Store, being a *read-write* entity, is represented by a category with individual stores σ as objects and pairs of stores as morphisms. The morphism $\langle \sigma, \sigma' \rangle$ represents an update of store σ (readable component) into σ' (writeable component). The morphisms $\langle \sigma, \sigma' \rangle$ and $\langle \sigma'', \sigma''' \rangle$ are composable only if $\sigma' = \sigma''$; the result of their composition is $\langle \sigma, \sigma''' \rangle$.
- *Write-only* entities, such as emitted signals, are represented by a category with a single object. Morphisms are finite (possibly empty) sequences generated over a set of actions (signals); their composition is concatenation. Individual signals are singleton sequences and there is a single identity morphism: the empty sequence.

Label indices are determined by a *label profile* $\mathcal{L}$, which divides them according to what kind of entity they represent. In the example of Table 1 we only have one label index for each kind of entity: **env** is a read-only label index for environments, **sto** is a read-write label index for stores, and **out** is a write-only label index for the output stream. By convention, indices for write-only entities and writeable components of read-write entities are *always* primed (**out′**, **sto′**), while read-only and readable components of read-write entities are always unprimed (**env**, **sto**). The label profile is instantiated by a set of terms T to $\mathcal{L}(T)$, a set of finite maps from label indices to label terms. For instance, for any transition label $L \in \mathcal{L}(T)$ that contains a store entity as its component at index **sto**, $L(\mathbf{sto})$ is the store at the beginning of the transition and $L(\mathbf{sto'})$ is the store at the end of the transition.

Definition 6 (MSOS labels [7]). *A* label profile *is a triple of disjoint sets* $\mathcal{L} = \langle \mathcal{L}_{RO}, \mathcal{L}_{RW}, \mathcal{L}_{WO} \rangle$. *The set* $\mathrm{reads}(\mathcal{L})$ *consists of the* unprimed indices $\mathbf{l} \in \mathcal{L}_{RO} \uplus \mathcal{L}_{RW}$. *The set* $\mathrm{writes}(\mathcal{L})$ *consists of the* primed indices $\{\mathbf{l'} : \mathbf{l} \in \mathcal{L}_{WO} \uplus \mathcal{L}_{RW}\}$. *For any set* T, *the* label set $\mathcal{L}(T)$ *is the set of maps* $(\mathrm{reads}(\mathcal{L}) \uplus \mathrm{writes}(\mathcal{L})) \to T$. *For a label* $L \in \mathcal{L}(T)$, *we write* $\mathrm{reads}(L)$ *and* $\mathrm{writes}(L)$ *for the restriction of* L *to* $\mathrm{reads}(\mathcal{L})$ *and* $\mathrm{writes}(\mathcal{L})$ *respectively.*

2.6 MSOS Rules and Specifications

The form of MSOS rules is much the same as for SOS rules, except that configurations are now simply terms (possibly with computed values as sub-terms) and labels contain multiple entities. Although a rule might specify entity morphisms explicitly via side-conditions as in [29], a more readable approach is using the *record pattern* notation introduced in [31]. Record patterns allow to specify labels of transitions directly above the arrows, as illustrated in Table 1. Label patterns usually only mention entities directly used by the rule, while the remaining components of the label are propagated throughout the rule using

special meta-variables '...' matching any label components, and '−' matching only label components that are identity morphisms. For example, in Rule 8 the label pattern '$\{\mathbf{out}' = v, -\}$' matches any transition label L where $L(\mathbf{out}')$ is the value v (representing the morphism v) and any other component is an identity morphism. The pattern '$\{-\}$' matches labels that are completely unobservable, while '$\{\ldots\}$' matches any label.

Definition 7 (MSOS label patterns). *Given a label profile $\mathcal{L}$ and a signature Σ, a* label pattern *is a sequence of equations '$\mathbf{l} = t$', enclosed in '$\{\}$' and (optionally) terminated by label meta-variables '...' or '−'. In '$\mathbf{l} = t$', $\mathbf{l}$ is a label index from* $\mathrm{reads}(\mathcal{L}) \uplus \mathrm{writes}(\mathcal{L})$ *and t is an open term over Σ. The meta-variable '...' matches* any *label components, except those already mentioned in the sequence, and '−' matches all unmentioned components of the label only if they are identity morphisms. At most one of '...' and '−' may appear.*

An explicit label pattern *is a label pattern that contains no '...' or '−'.*

In practice, label patterns in rules *always* end with the meta-variables '...' or '−' as this ensures that the rules are modular. In the remainder of this paper we will use the meta-variable P for label patterns as defined above, and L for labels (maps), as specified in Definition 6.

Definition 8 (MSOS rules). *Let Σ be a signature and $\mathcal{L}$ a label profile. An MSOS rule is a pair $\langle H, c\rangle$, usually written $\frac{H}{c}$, where H is the set of premises and c is the conclusion. Premises and the conclusion are formulas of the form $t \xrightarrow{P} t'$, where t and t' are open terms over Σ, and P is an MSOS label pattern over $\mathcal{L}$ and Σ.*

Definition 9 (MSOS specifications). *An* MSOS specification *is a tuple $\langle \Sigma, \mathcal{L}, D\rangle$, where Σ is a vc-signature (Definition 5), $\mathcal{L}$ a label profile (Definition 6) and D a set of MSOS rules (Definition 8).*

2.7 Models of MSOS Specifications

The most general model of computation in MSOS is a *generalised transition system (GTS)* $\langle \Gamma, \mathbb{L}, \longrightarrow, T\rangle$, where $\mathbb{L}$ is a category with morphisms L, such that $\langle \Gamma, L, \longrightarrow, T\rangle$ is an LTTS (Definition 1). Computations in a GTS are computations in the underlying LTTS, with the restriction that labels on adjacent transitions have to be composable.

Since a value-computation signature distinguishes value terms syntactically, we can have a transition system without a distinguished set of final states. We simply disallow transitions from value terms. The following notion of a value-computation transition system is adapted from [7].

Definition 10 (value-computation transition systems). *A* value-computation transition system (VCTS) *is a tuple $\langle \Sigma, L, \longrightarrow\rangle$, where Σ is a vc-signature, L a set of labels, and $\longrightarrow \subseteq T_\Sigma \times L \times T_\Sigma$ a transition relation. If $s \xrightarrow{l} s'$, then $s \notin V_\Sigma$.*

An MSOS specification $\langle \Sigma, \mathcal{L}, D \rangle$ generates a VCTS $\langle \Sigma, \mathcal{L}(T), \longrightarrow \rangle$ for some T (usually T_Σ) by letting $\longrightarrow$ be the least relation closed under rules in D. Since we do not allow negative premises in our specifications, there is always a model and it is unique.

The provability of a transition assertion is established by finding a rule in D and a closing substitution for the rule, such that the conclusion matches the transition assertion and the premises are also provable in this way.

Definition 11 (provable transitions). *A transition assertion $s \xrightarrow{L} s'$ is* provable *in an MSOS specification $\langle \Sigma, \mathcal{L}, D \rangle$ iff there is a rule $\langle H, t \xrightarrow{P} t' \rangle \in D$ and a closing substitution σ, such that, $s = t[\sigma]$, $s' = t'[\sigma]$, $L = P[\sigma]$ and for all $t_i \xrightarrow{P_i} t'_i \in H$, $t_i[\sigma] \xrightarrow{P_i[\sigma]} t'_i[\sigma]$ is also a provable transition in the specification.*

In this definition and in the remainder of the paper we lift substitutions to terms and label patterns in the obvious way by letting $s[\sigma]$ denote the term obtained by replacing all occurrences of variables $x \in \mathrm{dom}(\sigma)$ in s by $\sigma(x)$. $P[\sigma]$ for label patterns P is defined similarly.

2.8 Equivalence in MSOS

Just like with ordinary SOS, bisimulation can be used to prove equivalence between program terms. As indicated in [31], the usual notion of strong bisimulation can be used on MSOS directly and weak bisimulation just needs to be modified by replacing τ-transitions with the notion of unobservable transitions in MSOS. A higher-order *MSOS bisimulation* theory was developed in [7] for value-computation transition systems extended with rewriting. In addition to the usual transition relation (as in Definition 10), such a transition system also has a *rewrite* relation '$\Rightarrow$' used for value-computations that are entirely independent of any auxiliary entities. The relation is a *precongruence*, that is, a reflexive and transitive relation which is preserved by all constructors. MSOS bisimulation allows bisimilarities between label components and uses the rewriting relation to achieve some of the advantages of weak bisimulation regarding transitions that are independent of label components.

A reformulation of the *tyft/tyxt* format [18] for MSOS guarantees that this notion of MSOS bisimulation is a congruence. In [7] only the tyft part of the format is considered but extension to tyxt is straightforward as long as value terms are not allowed to make any transitions in accordance with Definition 10.

It is simple to identify context-independent transitions in a specification: they are transitions labelled with '$\{-\}$' (e.g. Rules 2, 3, 6 and 16 in Table 1). Replacing those transition with rewrites (e.g. $\mathsf{seq}(\mathsf{skip}, s) \Rightarrow s$) allows us, for example, to prove unit laws for sequencing, $\mathsf{seq}(s, \mathsf{skip}) \equiv s \equiv \mathsf{seq}(\mathsf{skip}, s)$, which wouldn't hold using strong bisimulation with MSOS. In this paper we are, however, also interested in equivalences between terms that involve *unobservable* but *context-dependent* transitions, such as $\mathsf{let}(i, v, \mathsf{bound}(i)) \equiv v$. These terms are not MSOS bisimilar due to $\mathsf{let}(i, v, \mathsf{bound}(i))$ requiring a transition for looking up the bound

value in the environment according to Rule 5. The rest of the paper discusses two approaches to proving such equivalences.

3 Absorbing Unobservable Transitions

We would like to prove equivalences involving context-dependent unobservable transitions. These can justify simple optimisations which replace a program term with a term that takes fewer computation steps but still has the same observable behaviour and results in the same value. For example, given the specification in Table 1, we want the following equivalences to hold.

Example 12. $\mathsf{let}(i, v, \mathsf{bound}(i)) \equiv v$

Example 13. $\mathsf{let}(i, v, \mathsf{assign}(l, \mathsf{bound}(i))) \equiv \mathsf{assign}(l, v)$

In this section we explore an approach where we allow the unobservable transitions in a VCTS to be ignored by adding *abstraction* rules. These rules are inspired by Van Glabbeek [10] and are similar to built-in rules for the rewriting relation in [7].

Absorption rules permit hiding of unobservable transitions occurring before or after a transition:

$$\text{AbsL}\ \frac{s \xrightarrow{\{-\}} s' \qquad s' \xrightarrow{L} s''}{s \xrightarrow{L} s''} \qquad\qquad \text{AbsR}\ \frac{s \xrightarrow{L} s' \qquad s' \xrightarrow{\{-\}} s''}{s \xrightarrow{L} s''}$$

Either of these rules also gives us transitivity of unobservable transitions.

These rules allow us to prove strong MSOS *simulations* (as bisimulations but not symmetric) between some terms that were excluded before. For example, we now have that $\mathsf{let}(i, v, \mathsf{assign}(l, \mathsf{bound}(i)))$ strongly simulates $\mathsf{assign}(l, v)$. They are still not bisimilar: the former term can make an unobservable transition that cannot be matched by '$\mathsf{assign}(l, v)$'. We do not get any (bi)similarity between $\mathsf{let}(i, v, \mathsf{bound}(i))$ and v.

To achieve (strong) bisimilarity in such cases, we add a *stuttering* (or *waiting*) rule, allowing a term to make an unobservable transition to itself.

$$\frac{}{s \xrightarrow{\{-\}} s}$$

A stuttering transition under this rule is similar to stuttering steps in TLA [20]. In bisimulation proofs, this rule will allow terms to stutter and thus to match an unobservable transition of the challenger with a transition that has no observable effect at all.

The rule is less natural than absorption rules since now any term can make an unobservable transition. It also prevents specification of constructs whose behaviour may depend on the number of unobservable transitions their components make.

In the setting of programming languages we are mainly interested in computations resulting in a value. From this perspective, a general stuttering rule still has two issues. The first is that now value terms have transitions. As noted in [7], values should only be inspected. To address this, we argue that stuttering transitions are not proper computation steps. The term that is the target of the transition is equal to the source and the label is an identity morphism, thus there is nothing that can be observed about the transition. Additionally, a value-computation signature distinguishes value terms (final states), so we can always formulate a condition for when a computation is finished.

The second issue is that now any computation term can perform an infinite sequence of transitions without making any progress. All computations are potentially non-terminating. Furthermore, computations which do not terminate in the original system (without abstraction rules) are now identified with stuck terms ('deadlock = livelock'). We can address this by limiting stuttering to terms that either can progress by making an observable transition, or are value terms. We express this as two stuttering rules. We assume that v ranges over value terms and the predicate unobs holds if its argument is an unobservable label.

$$\text{Wait}\ \frac{s \xrightarrow{L} s' \qquad \neg\,\mathrm{unobs}(L)}{s \xrightarrow{\{-\}} s} \qquad\qquad \text{Stutter}\ \frac{}{v \xrightarrow{\{-\}} v}$$

These two rules, together with the two absorption rules introduced before, can be used to abstract from unobservable transitions and prove the desired equivalences in Examples 12 and 13 using MSOS bisimulation as illustrated by Fig. 1.

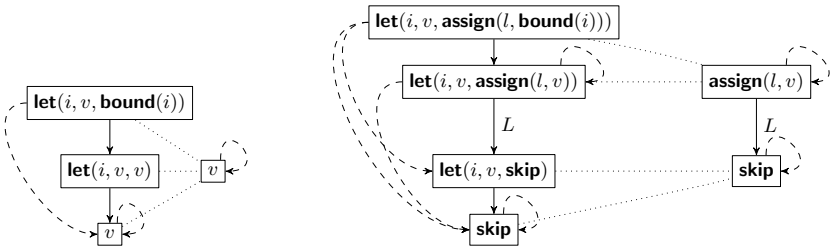

Fig. 1. Transitions of Examples 12 and 13 with abstraction rules added to the system. Unlabelled arrows represent unobservable transitions. Dashed arrows are new transitions due to abstraction rules. Dotted lines connect states related by the bisimulation relation. ($L = \{\mathbf{sto} = \sigma, \mathbf{sto}' = \sigma[l \mapsto v], -\}$).

For our stuttering rule to be admissible, we need to modify the condition in Definition 10 which prohibits transitions from value terms. The condition becomes the following: *If there is a transition* $s \xrightarrow{L} s'$*, then* $s \notin V_\Sigma$*, unless* $s = s'$ *and* unobs(L). With this restriction relaxed, all our rules are in the MSOS tyft/tyxt format and thus bisimilarity stays a congruence in the 'abstracted'

system. The bisimilarity achieved in this way also avoids the usual 'deadlock = livelock' problem.

We stress that the proposed absorption and stuttering rules are to be added to a MSOS specification as *abstraction* rules for reasoning about weak equivalences between terms. They are not useful as *computation* rules (e.g., when animating the semantics), since the two stuttering rules may add many potentially infinite computations.

4 A WB Cool MSOS format

This section presents a version of the Simply WB Cool format [3] for systems specified in MSOS. To this end we first present an MSOS version of the positive GSOS format (GSOS for MSOS). Then we give a definition of *weak MSOS bisimulation* and place further restrictions on the GSOS for MSOS format to guarantee that weak MSOS bisimulation is a congruence.

4.1 Positive GSOS for MSOS

Our notion of *strong MSOS bisimulation* allows bisimilarities between labels of transitions as well as configurations. To this end we implicitly lift the relation $\mathcal{R}$ to finite maps: $m_1\ \mathcal{R}\ m_2$ holds for two finite maps m_1 and m_2 with the same domain, whenever $m_1(i)\ \mathcal{R}\ m_2(i)$ for all i in the domain of the two maps.

In the following definition the readable label components of the two transitions are required to be the same. This is due to the composability condition of categories for read-only and read-write entities, which requires readable components to be the same. The possibility of transitions with bisimilar readable components is a consequence of the format, as shown by Lemma 18 below.

Definition 14 (strong MSOS bisimulation). *A* strong MSOS bisimulation *over a given VCTS $\langle \Sigma, \mathcal{L}(T_\Sigma), \longrightarrow \rangle$ is a symmetric relation $\mathcal{R} \subseteq T_\Sigma \times T_\Sigma$ such that, whenever $s\ \mathcal{R}\ t$,*

1. *if $s \xrightarrow{L} s'$, then $\exists t', L'.t \xrightarrow{L'} t'$,* $\mathrm{reads}(L) = \mathrm{reads}(L')$, $\mathrm{writes}(L)\ \mathcal{R}\ \mathrm{writes}(L')$, *and $s'\ \mathcal{R}\ t'$; and*
2. *if $s = v(s_1, \ldots, s_n)$ with $v \in VC$, then $t = v(t_1, \ldots, t_n)$ and $s_i\ \mathcal{R}\ t_i$ for $1 \leq i \leq n$.*

Two terms s and t are strongly MSOS bisimilar*, written $s \sim_{msos} t$, iff $s\ \mathcal{R}\ t$ for some strong MSOS bisimulation $\mathcal{R}$.*

This version allows us to prove associativity laws for constructs. For example, we have $\mathbf{seq}(p, \mathbf{seq}(r, q)) \sim_{msos} \mathbf{seq}(\mathbf{seq}(p, r), q)$, since both terms have the same transitions (observable or unobservable). But unit laws for sequencing, provable by MSOS bisimulation with rewrites, do not hold under strong MSOS bisimulation.

Next, we define a positive GSOS format for MSOS. Following [7] we extend the original GSOS format with value patterns and cater for the data flow discipline between the conclusion and premises of an MSOS rule.

Definition 15 (patterns [7]). *A pattern u is an open term constructed from value constructors and variables.*

The format definitions use new auxiliary functions ins and outs, in addition to reads and writes lifted to patterns and a selection operation to denote terms in label patterns. These are defined for a label pattern P (over a profile $\mathcal{L}$) as follows:

- $\mathrm{reads}(P) = \mathrm{reads}(\mathcal{L})$ and $\mathrm{writes}(P) = \mathrm{writes}(\mathcal{L})$;
- $P(\mathbf{l})$ denotes the term at label $\mathbf{l}$; and
- $\mathrm{ins}(P)$ denotes the set of all $P(\mathbf{l})$ for $\mathbf{l} \in \mathrm{reads}(P)$ and $\mathrm{outs}(P)$ the set of all $P(\mathbf{l})$ for $\mathbf{l} \in \mathrm{writes}(P)$.

Definition 16 (positive GSOS for MSOS). *A rule for a construct $f \in (C_\Sigma \setminus VC_\Sigma)$ with $n = \mathrm{ar}(f)$ is in the positive GSOS for MSOS format if it has the following form:*

$$\frac{\{x \xrightarrow{P_j} y_j \mid x \in X_i, j \in J_x\}}{f(u_1, \ldots, u_n) \xrightarrow{P} t}$$

where for $1 \leq i \leq n$, $X_i \subseteq \mathrm{vars}(u_i)$ and each u_i is a pattern; J_x is a finite set indexing premises for each x; P and each P_j is a label pattern; and all $\mathrm{ins}(P)$ and $\mathrm{outs}(P_j)$ are patterns. Variables in all u_i, $\{y_j \mid j \in J\}$ (where J is the union of all J_x), $\mathrm{ins}(P)$ and $\mathrm{outs}(P_j)$ are all distinct. No variables from u_i or $\mathrm{ins}(P)$ may appear in any $\mathrm{outs}(P_j)$. Variables in $\mathrm{ins}(P_j)$ must appear in the source of the conclusion or in $\mathrm{ins}(P)$. Finally, variables in t and in each $\mathrm{outs}(P)$ must appear in the source of the conclusion, as targets of premises, in $\mathrm{ins}(P)$, or in $\mathrm{outs}(P_j)$.

An MSOS specification is in the positive GSOS for MSOS format if all its rules are.

The specification in Table 1 is in the positive GSOS for MSOS format, if we consider the mathematical notation for maps and sequences as syntactic sugar for value terms, and we allow side conditions such as $\rho(i) = v$ in Rule 7. In general, side conditions that restrict instantiations of variables with value terms (such as the environment ρ in Rule 7) do not affect the format. They can be understood as generating sets of positive GSOS for MSOS rules. We could also express data operations (such as lookup in maps) using further constructs in the language which would only have unobservable transitions. This approach is used in [7] with rewriting instead of transitions.

We can use the following lemma to obtain bisimilar substitutions for bisimilar value terms.

Lemma 17. *Let $\mathcal{R}$ be the congruence closure of $\sim_{msos}$, u a pattern, and σ a substitution with $\mathrm{dom}(\sigma) = \mathrm{vars}(u)$. Assume $u[\sigma]\ \mathcal{R}\ t$. Then there is a substitution π such that $\mathrm{dom}(\pi) = \mathrm{vars}(u)$, $t = u[\pi]$, and for each $x \in \mathrm{vars}(u)$, $\sigma(x)\ \mathcal{R}\ \pi(x)$.*

Proof. By induction on the structure of u. □

The following lemma shows that if a term makes a transition with the label L, the same term can make a transition with a label that is *point-wise bisimilar* to L. The target terms of the transitions will also be bisimilar. Note that as a consequence of the positive GSOS for MSOS format, the lemma holds for any congruence relation, not just bisimulation.

Lemma 18. *Given an MSOS specification in the positive GSOS for MSOS format, let $\mathcal{R}$ be a congruence relation and P an explicit label pattern. Assume $s \xrightarrow{L} s'$ and let σ be a closing substitution such that* $\mathrm{reads}(P[\sigma])\ \mathcal{R}\ \mathrm{reads}(L)$. *Then there is a term s'' and a substitution σ' such that $s \xrightarrow{P[\sigma']} s''$,* $\mathrm{reads}(P[\sigma']) = \mathrm{reads}(P[\sigma])$, $\mathrm{writes}(P[\sigma'])\ \mathcal{R}\ \mathrm{writes}(L)$ *and $s'\ \mathcal{R}\ s''$.*

Proof. By induction on the height of the derivation tree of $s \xrightarrow{L} s'$. □

Theorem 19. *Given an MSOS specification, if all rules are in the positive GSOS for MSOS format, then $\sim_{msos}$ is a congruence for the specified language.*

Proof. We show that the congruence closure of $\sim_{msos}$ is also a strong MSOS bisimulation. The congruence closure $\mathcal{R}$ is formed by the following rules:

1. $s \sim_{msos} t$ implies $s\ \mathcal{R}\ t$
2. $s_i\ \mathcal{R}\ t_i$ implies $s = f(s_1, \ldots, s_n)\ \mathcal{R}\ f(t_1, \ldots, t_n) = t$, where $n = \mathrm{ar}(f)$ and $1 \leq i \leq n$.

The proof proceeds by induction on the number of applications of clause 2. The base case is immediate since $\sim_{msos}$ is a strong MSOS bisimulation.

In the inductive step we assume that the bisimulation property holds for all s_i and t_i, derived by fewer applications of clause 2, to show that it also holds for s and t. For condition 1 of Definition 14, we assume $s \xrightarrow{L} s'$ and show that there exists a transition $t \xrightarrow{L'} t'$ that satisfies the condition. We use the known format of the rule used to derive $s \xrightarrow{L} s'$ together with Lemmas 17 and 18 to find a substitution for the same rule so that it can be used to prove $t \xrightarrow{L'} t'$, as required by Definition 11. Condition 2 of Definition 14 follows immediately from the induction hypothesis. □

For a detailed proof see Appendix A.1.

4.2 Simply WB Cool MSOS

Now we adapt the WB Cool format as presented in [11] to MSOS. The format is a further restriction of positive GSOS for MSOS. We want the following notion of weak bisimilarity to be a congruence.

Definition 20 (weak MSOS bisimulation). *A* weak MSOS bisimulation *over a given VCTS $\langle \Sigma, \mathcal{L}(T_\Sigma), \longrightarrow \rangle$ is a symmetric relation $\mathcal{R} \subseteq T_\Sigma \times T_\Sigma$ such that, whenever $s\ \mathcal{R}\ t$,*

1. *if* $s \xrightarrow{L} s'$, *then* $\exists t', L'.t \longrightarrow^* \xrightarrow{(L')} \longrightarrow^* t'$, $\mathrm{reads}(L) = \mathrm{reads}(L')$, $\mathrm{writes}(L)\ \mathcal{R}\ \mathrm{writes}(L')$, *and* $s'\ \mathcal{R}\ t'$; *and*
2. *if* $s = v(s_1, \ldots, s_n)$ *with* $v \in VC$, *then* $\exists t', t \longrightarrow^* t'$ *and* $t' = v(t_1, \ldots, t_n)$ *with* $s_i\ \mathcal{R}\ t_i$ *for* $1 \leq i \leq n$;

where $t \xrightarrow{(L')} t'$ *is* $t \xrightarrow{L'} t'$ *if* L' *is observable, and* $t \xrightarrow{L'} t'$ *or* $t = t'$ *if it is unobservable.*

Two terms s *and* t *are* weakly MSOS bisimilar*, written* $s \approx_{msos} t$*, iff* $s\ \mathcal{R}\ t$ *for some weak MSOS bisimulation* $\mathcal{R}$*.*

Under this definition, we now have for example, $\mathsf{seq}(s, \mathsf{skip}) \approx_{msos} s \approx_{msos} \mathsf{seq}(\mathsf{skip}, s)$ and we can also justify the equivalences in Examples 12 and 13.

In Cool formats, *patience rules* allow an argument to perform all its τ-transitions before an observable transition can be performed. Since a VCTS usually has many meaningful final states and arguments may be inspected even after performing all their (observable) transitions, we generalise patience rules to *congruence* rules.

Definition 21 (congruence rules). *A* congruence rule *for a construct* f *with* $n = \mathrm{ar}(f)$ *is a rule of the following form:*

$$\frac{x_i \xrightarrow{P_i} y_i}{f(u_1, \ldots, u_{i-1}, x_i, u_{i+1}, \ldots u_n) \xrightarrow{P} f(u_1, \ldots, u_{i-1}, y_i, u_{i+1}, \ldots, u_n)}$$

where u_j $(1 \leq j \leq n)$ *are patterns for value terms (Definition 15);* x_i *and* y_i *are term variables and for all* $\mathbf{l} \in \mathrm{writes}(P) \cup \mathrm{writes}(P_i)$, $P(\mathbf{l}) = P_i(\mathbf{l})$.

In MSOS, a patience rule *is a congruence rule with only unobservable labels.*

Similarly, our notion of an *active argument* has to cater for patterns in rules.

Definition 22 (active and receiving arguments). *The* i*th argument of a construct* f*, with* $1 \leq i \leq \mathrm{ar}(f)$*, is* active *if* f *has a GSOS rule, where* x_i *is a variable on the left-hand side of a premise or it appears in* $\mathrm{ins}(P_j)$ *of a premise; or* u_i *is a pattern* $v(w_1, \ldots, w_{\mathrm{ar}(v)})$*, for* $v \in VC_\Sigma$*, in the source of the conclusion.*

A variable y *is* receiving *in* t *if* t *is the target of a rule in which* y *appears in the right-hand side of a premise. The* i*th argument of* f *is* receiving *if a variable* y *is receiving in a term* t *that has a subterm* $f(t_1, \ldots, t_{\mathrm{ar}(f)})$ *and* y *appears in* t_i*.*

Definition 23 (Simply WB Cool MSOS). *An MSOS specification is in the Simply WB Cool MSOS format if it is in the positive GSOS for MSOS format and*

1. *no rule has a variable* x *occurring more than once among the sources of premises;*
2. *in any rule, no variable appears both in the target of the conclusion and in the source of a premise;*
3. *only congruence rules can have premises with unobservable transitions;*

4. *there is a congruence rule for every active argument;*
5. *there is a congruence rule for every receiving argument;*
6. *patterns in the source of the conclusion have the form* $v(z_1, \ldots, z_{\mathrm{ar}(v)})$*, where all* z_i *are all variables;*
7. *if a variable* x *appears in the premises, it must be an argument* $u_i = x$ *in the source of the conclusion; and*
8. *for all* $\mathbf{l} \in \mathrm{writes}(P_j)$*,* $P_j(\mathbf{l})$ *must a variable.*

Conditions 1-5 correspond to the original restrictions of the WB Cool format. Condition 6 forbids nested patterns. (Otherwise one would have to specify 'nested congruence rules' for all patterns with nested value constructors. This generalisation is straightforward, however it complicates the definition of the format while rarely being used in practice. Moreover, nested pattern matching can be split among auxiliary constructs.) Condition 7 ensures that congruence rules can be applied to all terms going into readable components of premise labels. Due to condition 8, a writeable component of a premise can only be pattern matched via an auxiliary construct. This again ensures that congruence rules can be applied to a term obtained this way.

Note that as a consequence of condition 8 the technique used in [31] for specifying failure and exceptions using write-only entities might become problematic. A rule cannot check for an exception flag directly, but has to use an auxiliary construct with intermediate unobservable transitions. This could be alleviated by restricting (writeable) label components to value terms and allowing patterns (in accordance with condition 6) for matching on writeable components in the premises.

All rules of Table 1 are in Simply WB Cool MSOS. Rules 1, 4, 5, 8, 10, 11, 13 and 15 are congruence rules.

Theorem 24. *Given an MSOS specification, if all rules are in the Simply WB Cool MSOS format, then* $\approx_{msos}$ *is a congruence for the specified language.*

Proof. We need to show that the congruence closure of $\approx_{msos}$ is also a weak MSOS bisimulation. The congruence closure $\mathcal{R}$ is formed by the following rules:

1. $s \approx_{msos} t$ implies $s \mathcal{R} t$
2. $s_i \mathcal{R} t_i$ implies $s = f(s_1, \ldots, s_n) \mathcal{R} f(t_1, \ldots, t_n) = t$, where $n = \mathrm{ar}(f)$ and $1 \leq i \leq n$.

The proof proceeds by induction on the number of applications of clause 2. The base case is immediate since $\approx_{msos}$ is a weak MSOS bisimulation.

In the induction step we assume that the bisimulation property holds for all s_i and t_i, derived by fewer applications of clause 2, and show that it also holds for s and t. For condition 1 of Definition 20, we assume that $s \xrightarrow{L} s'$. We use the known format of the rule d used to derive that transition, the conditions of WB Cool MSOS, and Lemma 18 to derive the (possibly empty) transition sequence

$$t = f(t_1, \ldots, t_n) \longrightarrow^* \xrightarrow{(L')} \longrightarrow^* t'.$$

Congruence rules for active arguments of f can be used to derive the first sequence of unobservable transitions $\longrightarrow^*$. Then rule d, used to derive $s \xrightarrow{L} s'$, can be used to derive the transition $\xrightarrow{(L')}$ if L is observable, otherwise there might be no transition. The last unobservable transition sequence $\longrightarrow^*$ can be derived using congruence rules for receiving variables in rule d. Because we find in each step a substitution related by $\mathcal{R}$ to the one used with rule d to derive $s \xrightarrow{L} s'$, we also have that $s' \mathcal{R} t'$, $\mathrm{reads}(L) = \mathrm{reads}(L')$ and $\mathrm{writes}(L) \mathcal{R} \mathrm{writes}(L')$ as required. Condition 2 of Definition 20 follows immediately from the assumption in this case. □

For a detailed proof see Appendix A.2.

5 Conclusion

By abstracting away from context-free transitions, MSOS bisimulation with precongruence from [7] allows us to prove many useful laws in an MSOS setting that do not hold under the usual notions of strong bisimulation. It retains the pleasant properties of strong bisimulation and comes with a liberal tyft congruence format. However, for many useful equivalences we need to ignore also context-dependent unobservable transitions.

We have investigated two approaches of abstracting away from unobservable transitions while ensuring the equivalence is also a congruence. One approach is based on weakening the transition system generated from a specification by adding absorption rules. The main advantage of this approach is that it allows us to use the existing MSOS-tyft congruence format of [7]. However, the cost of this is that computations become potentially diverging. Also, abstraction rules modify the semantics of some constructs in unintended ways. A typical example would be the sum operator from CCS [27].

The other approach we investigated was to define a notion of weak bisimulation for MSOS and equip it with a congruence format. We have chosen the Simply WB Cool format based on the well-studied (positive) GSOS. Although checking whether rules in a specification are in that format could be tedious, it should be straightforward to implement such checks in tool support for MSOS (this is left as future work). While not as powerful as MSOS tyft/tyxt, the Simply WB Cool MSOS format seems to be sufficient for specifying most common programming constructs. The approach based on absorption currently has the advantage of allowing more general premises of rules. This also includes a more straightforward specification of exception handling as mentioned in the previous section.

Finally, let us compare our approach with abstraction in Rewriting Logic. Turning up the "abstraction dial" in a Rewriting Logic semantics for a programming language involves changing some of its rules into equations, and checking that these equations are confluent (modulo structural axioms such as associativity and/or commutativity). This can dramatically reduce the state space. In an MSOS, in contrast, there is no such dial to turn: unobservable transitions arise naturally as a special case of general labelled transitions, and they are not

subject to any confluence conditions – although in practice, they are usually deterministic. We conjecture that by identifying the sources of unobservable deterministic transitions with their respective targets, we would obtain state space reductions in MSOS comparable to using the abstraction dial in Rewriting Logic. Unfortunately, we cannot simply add equations to (M)SOS specifications, as that could easily undermine bisimulation being a congruence: even associativity is dangerous [33, Example 9]. However, it should be possible to extend our bisimulation results to specifications in the MSOS tyft format [7], which would allow the use of a precongruence relation that enjoys similar properties to equations in Rewriting Logic.

References

1. Aceto, L., Fokkink, W., Verhoef, C.: Structural operational semantics. In: Smolka, J.B.P. (ed.) Handbook of Process Algebra, pp. 197–292. Elsevier (2001)
2. Bidoit, M., Mosses, P.D.: CASL User Manual – Introduction to Using the Common Algebraic Specification Language, LNCS, vol. 2900. Springer, Berlin Heidelberg (2004)
3. Bloom, B.: Structural operational semantics for weak bisimulations. Theor. Comput. Sci. 146(1–2), 25–68 (1995)
4. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. Journal of the ACM 42(1), 232–268 (1995)
5. Braga, C.d.O., Haeusler, E.H., Meseguer, J., Mosses, P.D.: Maude Action Tool: Using reflection to map action semantics to rewriting logic. In: AMAST '00. LNCS, vol. 1816, pp. 407–421. Springer, Berlin Heidelberg (2000)
6. Braga, C.d.O., Haeusler, E.H., Meseguer, J., Mosses, P.D.: Mapping modular SOS to rewriting logic. In: LOPSTR'02. LNCS, vol. 2664, pp. 262–277. Springer, Berlin Heidelberg (2003)
7. Churchill, M., Mosses, P.D.: Modular bisimulation theory for computations and values. In: FoSSaCS'13, LNCS, vol. 7794, pp. 97–112. Springer Berlin Heidelberg (2013)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude – A High-Performance Logical Framework, LNCS, vol. 4350. Springer, Berlin Heidelberg (2007)
9. Futatsugi, K., Goguen, J.A., Jouannaud, J.P., Meseguer, J.: Principles of OBJ2. In: POPL '85. pp. 52–66. ACM, New York, NY, USA (1985)
10. van Glabbeek, R.J.: Bounded nondeterminism and the approximation induction principle in process algebra. Tech. Rep. CS-R8634, CWI (1986)
11. van Glabbeek, R.J.: On cool congruence formats for weak bisimulations. Theor. Comput. Sci. 412(28), 3283–3302 (2011)
12. van Glabbeek, R.: Bounded nondeterminism and the approximation induction principle in process algebra. In: STACS '87. LNCS, vol. 247, pp. 336–347. Springer, Berlin Heidelberg (1987)
13. Goguen, J., Kirchner, C., Meseguer, J., Kirchner, H., Winkler, T., Megrelis, A.: An introduction to OBJ 3. In: 1st International Workshop on Conditional Term Rewriting Systems. LNCS, vol. 308, pp. 258–263. Springer, Berlin Heidelberg (1988)

14. Goguen, J.A.: Some design principles and theory for OBJ-O, a language to express and execute algebraic specification for programs. In: Mathematical Studies of Information Processing. LNCS, vol. 75, pp. 425–473. Springer, Berlin Heidelberg (1979)
15. Goguen, J.A., Jouannaud, J.P., Meseguer, J.: Operational semantics for order-sorted algebra. In: ICALP '85. LNCS, vol. 194, pp. 221–231. Springer, Berlin Heidelberg (1985)
16. Goguen, J.A., Meseguer, J.: Order-sorted algebra solves the constructor-selector, multiple representation and coercion problems. In: LICS '87. pp. 18–29. IEEE (1987)
17. Goguen, J.A., Meseguer, J.: Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theor. Comput. Sci. 105(2), 217–273 (1992)
18. Groote, J.F., Vaandrager, F.: Structured operational semantics and bisimulation as a congruence. Inf. Comput. 100(2), 202–260 (1992)
19. Kirchner, C., Kirchner, H., Meseguer, J.: Operational semantics of OBJ-3 (extended abstract). In: ICALP '88. LNCS, vol. 317, pp. 287–301. Springer, Berlin Heidelberg (1988)
20. Lamport, L.: The temporal logic of actions. ACM Trans. Program. Lang. Syst. 16(3), 872–923 (1994)
21. Martí-Oliet, N., Meseguer, J.: Rewriting logic as a logical and semantic framework. Electr. Notes Theor. Comput. Sci. 4, 190–225 (1996)
22. Meseguer, J.: Conditional rewriting logic as a united model of concurrency. Theor. Comput. Sci. 96(1), 73–155 (1992)
23. Meseguer, J.: Order-sorted parameterization and induction. In: Semantics and Algebraic Specification. LNCS, vol. 5700, pp. 43–80. Springer, Berlin Heidelberg (2009)
24. Meseguer, J.: Twenty years of rewriting logic. J. Log. Algebr. Program. 81(7-8), 721–781 (2012)
25. Meseguer, J., Goguen, J.A.: Order-sorted algebra solves the constructor-selector, multiple representation, and coercion problems. Inf. Comput. 103(1), 114–158 (1993)
26. Meseguer, J., Roşu, G.: The rewriting logic semantics project: A progress report. Inf. Comput. 231, 38–69 (2013)
27. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
28. Milner, R.: Communicating and Mobile Systems: The π-calculus. Cambridge University Press, New York, NY, USA (1999)
29. Mosses, P.D.: Foundations of modular SOS (extended abstract). In: MFCS'99. LNCS, vol. 1672, pp. 70–80. Springer, Berlin Heidelberg (1999)
30. Mosses, P.D.: Pragmatics of Modular SOS. In: AMAST'02. LNCS, vol. 2422, pp. 21–40. Springer, Berlin Heidelberg (2002)
31. Mosses, P.D.: Modular structural operational semantics. J. Log. Algebr. Program. 60-61, 195–228 (2004)
32. Mosses, P.D. (ed.): CASL Reference Manual, LNCS, vol. 2960. Springer, Berlin Heidelberg (2004)
33. Mousavi, M., Reniers, M.A.: Congruence for structural congruences. In: FoSSaCS'05, LNCS, vol. 3441, pp. 47–62. Springer Berlin Heidelberg (2005)
34. Mousavi, M., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. Theor. Comput. Sci. 373(3), 238–272 (2007)
35. Plotkin, G.D.: A Structural Approach to Operational Semantics. Tech. Rep. DAIMI FN-19, University of Aarhus (1981)

36. Plotkin, G.D.: A structural approach to operational semantics. J. Log. Algebr. Program. 60-61, 17–139 (2004)
37. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press, New York, NY, USA (2011)

A Proofs of congruence results

For all proofs we (implicitly) convert the rules in the Positive GSOS for MSOS format into an explicit version, following the technique in [7] (Proposition 16) which is also applicable for our current format. That makes label patterns in all rules *explicit*, i.e., we are always assuming explicit label patterns (without label meta-variables).

We also lift vars to sets of terms and to label patterns in the obvious ways.

A.1 Positive GSOS for MSOS

Lemma 17. *Let $\mathcal{R}$ be the congruence closure of $\sim_{msos}$, u a pattern, and σ a substitution with* $\mathrm{dom}(\sigma) = \mathrm{vars}(u)$. *Assume $u[\sigma]\ \mathcal{R}\ t$. Then there is a substitution π such that* $\mathrm{dom}(\pi) = \mathrm{vars}(u)$, *$t = u[\pi]$, and for each $x \in \mathrm{vars}(u)$, $\sigma(x)\ \mathcal{R}\ \pi(x)$.*

Proof. By induction on the structure of u.

Base case 1. Assume $u = x$, with $x \in \mathcal{X}$. Set $\pi(x) = t$ and we are done.

Inductive step 1. Assume $u = v(u_1, \ldots, u_n)$ for a value constructor v with $n = \mathrm{ar}(v)$ and patterns $u_1, \ldots, u_n$. We proceed by induction on the definition of $\mathcal{R}$.

> **Base case 2.** Assume $u[\sigma] \sim_{msos} t$. Then by Definition 14, $t = v(t_1, \ldots, t_n)$ with $u_i[\sigma \restriction_{\mathrm{vars}(u_i)}] \sim_{msos} t_i$. By the main induction hypothesis there is a substitution π_i such that $\mathrm{dom}(\pi_i) = \mathrm{vars}(u_i)$, $t_i = u_i[\pi_i]$ and for all $x \in \mathrm{vars}(u_i)$, $(\sigma \restriction_{\mathrm{vars}(u_i)})(x)\ \mathcal{R}\ \pi_i(x)$. Let π be the disjoint union of all such π_i and we are done.
>
> **Inductive step 2.** Assume $t = v(t_1, \ldots, t_n)$ and for all t_i, $u_i[\sigma \restriction_{\mathrm{vars}(u_i)}]\ \mathcal{R}\ t_i$. Then we can proceed as in the base case, obtaining π_i for each t_i using the induction hypothesis and letting π denote the disjoint union of all such π_i. Then we are done.

This concludes the proof. □

Lemma 18. *Given an MSOS specification in the Positive GSOS for MSOS format, let $\mathcal{R}$ be a congruence relation and P an explicit label pattern. Assume $s \xrightarrow{L} s'$ and let σ be a closing substitution such that* $\mathrm{reads}(P[\sigma])\ \mathcal{R}\ \mathrm{reads}(L)$. *Then there is a term s'' and a substitution σ' such that $s \xrightarrow{P[\sigma']} s''$,* $\mathrm{reads}(P[\sigma']) = \mathrm{reads}(P[\sigma])$, $\mathrm{writes}(P[\sigma'])\ \mathcal{R}\ \mathrm{writes}(L)$ *and $s'\ \mathcal{R}\ s''$.*

Proof. We assume $s \xrightarrow{L} s'$ and proceed by induction on the height of the derivation tree.

Base case. The tree is $\dfrac{}{s \xrightarrow{L} s'}$. Then there must be a GSOS for MSOS axiom $\dfrac{}{f(u_1,\ldots,u_n) \xrightarrow{P} t}$ with $n = \mathrm{ar}(f)$ and a closing substitution π such that

$s = f(u_1[\pi], \ldots, u_n[\pi])$, $L = P[\pi]$ and $s' = t[\pi]$. Let σ' be a substitution such that for all $x \in \text{vars}(\{u_1, \ldots, u_n\})$, $\sigma'(x) = \pi(x)$ and for all $x \in \text{vars}(\text{ins}(P))$, $\sigma'(x) = \sigma(x)$. Now we have for all $x \in \text{dom}(\sigma')$, $\sigma'(x)\ \mathcal{R}\ \pi(x)$. By Definition 16, all variables in $\text{outs}(P)$ and in t are already in σ'. Let $s'' = t[\sigma']$. Thus we have $s \xrightarrow{P[\sigma']} s''$; and by $\mathcal{R}$ being a congruence, $\text{reads}(P[\sigma']) = \text{reads}(P[\sigma])$, $\text{writes}(P[\sigma'])\ \mathcal{R}\ \text{writes}(L)$ and $s'\ \mathcal{R}\ s''$ as required.

Inductive step. The root of the tree with its immediate children is $\frac{Z}{s \xrightarrow{L} s'}$ where Z is the set of roots of immediate subtrees. There must be a Positive GSOS for MSOS rule $\frac{\{x \xrightarrow{P_j} y_j \mid x \in X_i, j \in J_x\}}{f(u_1,\ldots,u_n) \xrightarrow{P} t}$ and a closing substitution π such that $s = f(u_1, \ldots, u_n)[\pi]$ and $s' = t[\pi]$, $L = P[\pi]$ and Z is exactly the set of all $\pi(x) \xrightarrow{P_j[\pi]} \pi(y_j)$.

Our induction hypothesis is that for each $\pi(x) \xrightarrow{P_j[\pi]} \pi(y_j)$ if given a σ_j such that $\text{reads}(P_j[\sigma_j])\ \mathcal{R}\ \text{reads}(P_j[\pi])$, there is an s_j and a substitution σ'_j such that $\pi(x) \xrightarrow{P_j[\sigma'_j]} s_j$, $\text{reads}(P_j[\sigma'_j]) = \text{reads}(P_j[\sigma_j])$, $\text{writes}(P_j[\sigma'_j])\ \mathcal{R}\ \text{writes}(P_j[\pi])$, and $\pi(y_j)\ \mathcal{R}\ s_j$.

We construct a substitution μ^{in} such that for all $x \in \text{vars}(\{u_1, \ldots, u_n\})$, $\mu^{in}(x) = \pi(x)$ and for all $x \in \text{vars}(\text{ins}(P))$, $\mu^{in}(x) = \sigma(x)$. We now have for all $x \in \text{dom}(\mu^{in})$, $\mu^{in}(x)\ \mathcal{R}\ \pi(x)$ and for all P_j, $\text{reads}(P_j[\mu^{in}])\ \mathcal{R}\ \text{reads}(P_j[\pi])$. By the induction hypothesis, for all $\pi(x) \xrightarrow{P_j[\pi]} \pi(y_j)$ we obtain s_j and σ'_j such that $\text{reads}(P_j[\sigma'_j]) = \text{reads}(P_j[\mu^{in}])$, $\text{writes}(P_j[\sigma'_j])\ \mathcal{R}\ \text{writes}(P_j[\pi])$ and $\pi(y_j)\ \mathcal{R}\ s_j$. Now we construct for all j, a substitution μ^{out}_j such that for all $x \in \text{vars}(\text{outs}(P_j))$, $\mu^{out}_j(x) = \sigma'_j(x)$ and $\mu^{out}_j(y_j) = s_j$. Let σ' be the disjoint union of μ^{in} and all μ^{out}_j. We have $\sigma'(x)\ \mathcal{R}\ \pi(x)$ for all $x \in \text{dom}(\sigma')$. By Definition 16 all variables in $\text{writes}(P)$ and t are already in σ'. Let $s'' = t[\sigma']$. Thus we have $s \xrightarrow{P[\sigma']} s''$, $\text{reads}(P[\sigma']) = \text{reads}(P[\sigma])$ (since all $x \in \text{ins}(P)$ are from μ^{in} and thus σ), and by $\mathcal{R}$ being a congruence we have $\text{writes}(P[\sigma'])\ \mathcal{R}\ \text{writes}(L)$ and $s' = t[\pi]\ \mathcal{R}\ t[\mu] = s''$ as required.

This concludes the proof. □

Theorem 19. *Given an MSOS specification, if all rules are in the Positive GSOS for MSOS format, then $\sim_{msos}$ is a congruence for the specified language.*

Proof. We need to show that the congruence closure of $\sim_{msos}$ is also a strong MSOS bisimulation. Let $\mathcal{R}$ be the congruence closure of $\sim_{msos}$, i.e., the smallest relation such that

1. $p \sim_{msos} q$ implies $p\ \mathcal{R}\ q$
2. $p_i\ \mathcal{R}\ q_i$ implies $f(p_1, \ldots, p_n)\ \mathcal{R}\ f(q_1, \ldots, q_n)$, where $n = \text{ar}(f)$ and $1 \leq i \leq n$.

Since $\sim_{msos}$ is symmetric and transitive, so is $\mathcal{R}$. We proceed by induction on the number of applications of clause 2 and show that in each case the conditions of Definition 14 are satisfied.

Base case. Assume $p \sim_{msos} q$. Then all conditions of Definition 14 are satisfied by $\sim_{msos}$ being the largest strong MSOS bisimulation.

Inductive step. Let $p = f(p_1, \ldots, p_n)$ and $q = f(q_1, \ldots, q_n)$, where $n = \mathrm{ar}(f)$ and $1 \leq i \leq n$, $p_i \mathrel{\mathcal{R}} q_i$, with each being derived in fewer steps than $p \mathrel{\mathcal{R}} q$. As the induction hypothesis (i.h.) we assume the following for all $1 \leq i \leq n$:

Induction hypothesis

1. if $p_i \xrightarrow{L_i} p'_i$ then $\exists q'_i.q_i \xrightarrow{L'_i} q'_i$, $\mathrm{reads}(L_i) = \mathrm{reads}(L'_i)$, $\mathrm{writes}(L_i) \mathrel{\mathcal{R}} \mathrm{writes}(L'_i)$, and $p'_i \mathrel{\mathcal{R}} q'_i$
2. if $p_i = v(p_{i1}, \ldots, p_{im})$ then $q_i = v(q_{i1}, \ldots, q_{im})$ and $p_{ij} \mathrel{\mathcal{R}} q_{ij}$ for $1 \leq j \leq m$

For case 2 we also assume a corresponding induction hypothesis for all $p_{ij} \mathrel{\mathcal{R}} q_{ij}$. We will now show that under these assumptions, the conditions of $\mathcal{R}$ being a strong MSOS bisimulation are satisfied.

Condition 1. Assume $p \xrightarrow{L} p'$. Then there must be a rule $\dfrac{H}{f(u_1,\ldots,u_n) \xrightarrow{P} t}$ and a closing substitution σ, such that

- $f(u_1, \ldots, u_n)[\sigma] = p$ and so for all u_i we have $u_i[\sigma] = p_i$,
- $t[\sigma] = p'$,
- for all $x_{ij} \xrightarrow{P_{ijk}} y_{ijk} \in H$ we have $x_{ij} \in \mathrm{vars}(u_i)$, $x_{ij}, y_{ijk} \in \mathrm{dom}(\sigma)$ and there is a transition $\sigma(x_{ij}) \xrightarrow{P_{ijk}[\sigma]} \sigma(y_{ijk})$.

Using the assumption $u_i[\sigma_i] = p_i \mathrel{\mathcal{R}} q_i$, where $\sigma_i = \sigma \restriction_{\mathrm{vars}(u_i)}$, and Lemma 17 we can find for each q_i a substitution π_i such that $q_i = u_i[\pi_i]$, $\mathrm{dom}(\pi_i) = \mathrm{vars}(u_i)$ and for all variables $x \in \mathrm{vars}(u_i)$, $\sigma_i(x) \mathrel{\mathcal{R}} \pi_i(x)$.

By induction hypothesis we have that for all $\sigma_i(x_{ij}) \xrightarrow{P_{ijk}[\sigma]} \sigma(y_{ijk})$ there is a q'_{ijk} such that $\pi_i(x_{ij}) \xrightarrow{L'_{ijk}} q'_{ijk}$, $\mathrm{reads}(P_{ijk}[\sigma]) = \mathrm{reads}(L'_{ijk})$, $\mathrm{writes}(L_{ijk}[\sigma]) \mathrel{\mathcal{R}} \mathrm{writes}(L'_{ijk})$, and $\sigma(y_{ijk}) \mathrel{\mathcal{R}} q_{ijk}$.

Substitution π^{in}

Let π^{in} denote the disjoint union of $\sigma \restriction_{\mathrm{vars}(\mathrm{ins}(P))}$ and all π_i. We have for all $x \in \mathrm{dom}(\pi^{in})$, $\pi^{in}(x) \mathrel{\mathcal{R}} \sigma(x)$. Also, by Definition 16, $\mathrm{vars}(\mathrm{ins}(P_{ijk})) \subseteq \mathrm{vars}(\{u_1, \ldots, u_n\}) \uplus \mathrm{vars}(\mathrm{ins}(P)) = \mathrm{dom}(\pi^{in})$. Thus for all P_{ijk}, we have $\mathrm{reads}(P_{ijk}[\pi^{in}]) \mathrel{\mathcal{R}} \mathrm{reads}(L'_{ijk})$. Using Lemma 18, from $\pi^{in}(x_{ij}) \xrightarrow{L'_{ijk}} q'_{ijk}$ and the substitution π^{in} we obtain q''_{ijk} and a substitution π^P_{ijk} such that $\pi^{in}(x_{ijk}) \xrightarrow{P_{ijk}[\pi^P_{ijk}]} q''_{ijk}$, $\mathrm{reads}(P_{ijk}[\pi^P_{ijk}]) = \mathrm{reads}(P[\pi^{in}])$, $\mathrm{writes}(P_{ijk}[\pi^P_{ijk}]) \mathrel{\mathcal{R}} \mathrm{writes}(L'_{ijk})$ and $q'_{ijk} \mathrel{\mathcal{R}} q''_{ijk}$. By transitivity, we also have $P_{ijk}[\sigma] \mathrel{\mathcal{R}} P_{ijk}[\pi^P_{ijk}]$ and $\sigma(y_{ijk}) \mathrel{\mathcal{R}} q''_{ijk}$.

Substitution π

Now we will construct the final substitution π. For each $\pi^{in}(x_{ijk}) \xrightarrow{P_{ijk}[\pi^P_{ijk}]} q''_{ijk}$ obtained above, we construct a substitution π^{out}_{ijk} by letting $\pi^{out}_{ijk}(y_{ijk}) = q''_{ijk}$ and for all $x \in \mathrm{vars}(\mathrm{outs}(P_{ijk}))$, $\pi^{out}_{ijk}(x) = \pi^P_{ijk}(x)$. Let $\hat{\pi}$ be the disjoint union of π^{in} and all π^{out}_{ijk}. Then let $\pi(x) = \hat{\pi}(x)$ for all $x \in \mathrm{dom}(\hat{\pi})$ and for all other x, $\pi(x) = \sigma(x)$.

Finally, we have that for all x, $\pi(x)\ \mathcal{R}\ \sigma(x)$; $q = f(u_1[\pi], \ldots, u_n[\pi]) \xrightarrow{P[\pi]} t[\pi] = q'$; $\text{reads}(P[\sigma]) = \text{reads}(P[\pi])$ and, by $\mathcal{R}$ being a congruence, $\text{writes}(P[\sigma])\ \mathcal{R}\ \text{writes}(P[\pi])$ and $p' = t[\sigma]\ \mathcal{R}\ t[\pi] = q'$ as required.

Condition 2. If f is a value constructor v, then $p = v(p_1, \ldots, p_n)\ \mathcal{R}\ v(q_1, \ldots, q_n) = q$ by the assumption $p_i\ \mathcal{R}\ q_i$ for $1 \leq i \leq n$.

This concludes the proof that $\mathcal{R}$ is a strong MSOS bisimulation which shows that $\sim_{msos}$ is a congruence under an MSOS specification in the GSOS for MSOS format. □

A.2 Simply WB Cool MSOS

Theorem 24. *Given an MSOS specification $\langle \Sigma, \mathcal{L}, D \rangle$, if all rules are in the WB Cool MSOS format, then $\approx_{msos}$ is a congruence for the specified language.*

Proof. We need to show that the congruence closure of $\approx_{msos}$ is also a weak MSOS bisimulation. Let $\mathcal{R}$ be the congruence closure of $\approx_{msos}$, i.e., the smallest relation such that

1. $p \approx_{msos} q$ implies $p\ \mathcal{R}\ q$
2. $p_i\ \mathcal{R}\ q_i$ implies $f(p_1, \ldots, p_n)\ \mathcal{R}\ f(q_1, \ldots, q_n)$, where $n = \text{ar}(f)$ and $1 \leq i \leq n$.

Since $\approx_{msos}$ is symmetric and transitive, so is $\mathcal{R}$. We proceed by induction on the number of applications of clause 2 and show that in each case the conditions of Definition 20 are satisfied.

Base case. Assume $p \approx_{msos} q$. Then all conditions of Definition 20 are satisfied by $\approx_{msos}$ being a weak MSOS bisimulation.

Inductive step. Let $p = f(p_1, \ldots, p_n)$ and $q = f(q_1, \ldots, q_n)$, where $n = \text{ar}(f)$ and for $1 \leq i \leq n$, $p_i\ \mathcal{R}\ q_i$, with each being derived in fewer steps than $p\ \mathcal{R}\ q$. As the induction hypothesis (i.h.) we assume the following for all $1 \leq i \leq n$:

Induction hypothesis

1. if $p_i \xrightarrow{L_i} p_i'$ then $\exists q_i^1, q_i^2, q_i', L_i'.\ q_i \longrightarrow^* q_i^1 \xrightarrow{(L_i')} q_i^2 \longrightarrow^* q_i'$, $\text{reads}(L_i) = \text{reads}(L_i')$, $\text{writes}(L_i)\ \mathcal{R}\ \text{writes}(L_i')$, $p_i'\ \mathcal{R}\ q_i'$, also $p_i\ \mathcal{R}\ q_i^1$ and $p_i'\ \mathcal{R}\ q_i^2$; and
2. if $p_i = v(p_{i1}, \ldots, p_{im})$ then $q_i \longrightarrow^* v(q_{i1}, \ldots, q_{im})$ and $p_{ij}\ \mathcal{R}\ q_{ij}$

For case 2 we also assume a corresponding induction hypothesis for all $p_{ij}\ \mathcal{R}\ q_{ij}$. We will now show that under these assumptions, the conditions of $\mathcal{R}$ being a weak MSOS bisimulation are satisfied.

Condition 1. Assume $p \xrightarrow{L} p'$. Then there must be a rule $d \in D$

Rule d

$$d = \frac{H}{f(u_1, \ldots, u_n) \xrightarrow{P} t}$$

and a closing substitution σ, such that

- $f(u_1[\sigma], \ldots, u_n[\sigma]) = p$ and so for all u_i we have $u_i[\sigma] = p_i$,
- $t[\sigma] = p'$,
- for all $x \xrightarrow{P_x} y_x \in H$ with $x, y_x \in \mathrm{dom}(\sigma)$; for some u_i, $u_i = x$; and $\sigma(x) \xrightarrow{P_x[\sigma]} \sigma(y_x)$.

Following Definition 20, we must find a q' and L', such that

$$q \longrightarrow^* q^1 \xrightarrow{(L')} q^2 \longrightarrow^* q',$$

where q^1 and q^2 are intermediate states, $\mathrm{reads}(L) = \mathrm{reads}(L')$, $\mathrm{writes}(L)\ \mathcal{R}\ \mathrm{writes}(L')$, and $q\ \mathcal{R}\ q'$. We will thus find substitutions π and μ such that

$$f(q_1, \ldots, q_n) \longrightarrow^* f(u_1[\pi], \ldots, u_n[\pi]) \xrightarrow{(P[\pi])} t[\pi] \longrightarrow^* t[\mu]$$

with $t[\sigma]\ \mathcal{R}\ t[\mu]$, $\mathrm{reads}(P[\sigma]) = \mathrm{reads}(P[\pi])$ and $\mathrm{writes}(P[\sigma])\ \mathcal{R}\ \mathrm{writes}(P[\pi])$.

Substitution π^{in}

First we construct π^{in}, mapping variables in the source of the conclusion and readable components of the conclusion label:

1. For all non-active arguments i, we have $u_i = x_i$ so let $\pi^{in}(x_i) = q_i$.
2. By Definition 23, for all (active) arguments i, if $u_i = x_i$ (with $x_i \in \mathcal{X}$), then using the induction hypothesis,
   - if $p_i \xrightarrow{L_i} p_i'$, then let $\pi^{in}(x_i) = q_i^1$ from case 1; and
   - if $p_i = v(p_{i1}, \ldots, p_{im})$, then $\pi^{in}(x_i) = v(q_{i1}, \ldots, q_{im})$ from case 2.
3. If $u_i = v(z_1, \ldots, z_m)$ (and so $p_i = v(p_{i1}, \ldots, p_{im})$), let for all z_j, $\pi^{in}(z_j) = q_{ij}$ (case 2).
4. For all $x \in \mathrm{vars}(\mathrm{ins}(P))$, $\pi^{in}(x_i) = \sigma(x_i)$.

We now have for all $x \in \mathrm{dom}(\pi^{in})$, $\pi^{in}(x)\ \mathcal{R}\ \sigma(x)$. By Definition 16 all variables that appear on the left-hand side of a premise, or in $\mathrm{reads}(P_x)$ of a premise label, are already mapped in π^{in}. Moreover, for all $x_i \xrightarrow{P_i} y_i \in H$ (where $x_i = u_i$ in the source of the conclusion, following conditions 1 and 7 of Definition 23), $\sigma(x_i) \xrightarrow{P_i[\sigma]} \sigma(y_i)$ implies $\pi^{in}(x_i) = q_i^1 \xrightarrow{(L_i')} q_i^2$ with $\mathrm{reads}(L_i') = \mathrm{reads}(P_i[\sigma])$.

Since $\mathcal{R}$ is (by definition) a congruence and we know that $\mathrm{reads}(P_i[\pi^{in}])\ \mathcal{R}\ \mathrm{reads}(L_i')$, we can use Lemma 18 to find for each $\pi^{in}(x_i) \xrightarrow{L_i'} q_i^2$, a $q_i^{2\prime}$ and a substitution π_i' such that $\pi^{in}(x_i) \xrightarrow{P_i[\pi_i']} q_i^{2\prime}$, $\mathrm{reads}(P_i[\pi_i']) = \mathrm{reads}(P_i[\pi^{in}])$, $\mathrm{writes}(P_i[\pi_i'])\ \mathcal{R}\ \mathrm{writes}(L_i')$ and $q_i^2\ \mathcal{R}\ q_i^{2\prime}$.

Deriving $q \longrightarrow^* q^1$

For all active arguments i there must be a corresponding congruence rule, so whenever $q_i \longrightarrow^* q_i'$ (for some q_i') then we can also derive $f(q_1, \ldots, q_i, \ldots, q_n) \longrightarrow^* f(q_1, \ldots, q_i', \ldots, q_n)$. Using the induction hypothesis and repeatedly applying the congruence rules we can derive $q^1 = f(u_1[\pi^{in}], \ldots, u_n[\pi^{in}])$.

We proceed by case analysis on the rule d.

Deriving $q^1 \xrightarrow{(L')} q^2$

Case: d is a congruence rule. If d is a congruence rule

$$\frac{x_i \xrightarrow{P_i} y_i}{f(u_1, \ldots, x_i, \ldots, u_n) \xrightarrow{P} f(u_1, \ldots, y_i, \ldots, u_n)}$$

with writes(P) = writes(P_i) and $\sigma(x) \xrightarrow{P_i[\sigma]} \sigma(y)$, then $P_i[\sigma]$ may be observable or unobservable.

If $P_i[\sigma]$ is observable, then by induction hypothesis there is a transition $\pi^{in}(x) = q_i^1 \xrightarrow{L_i'} q_i^2$ with reads($P_i[\sigma]$) = reads(L_i'), writes($P_i[\sigma]$) $\mathcal{R}$ writes(L_i') and $\sigma(y)$ $\mathcal{R}$ q_i^2. Using Lemma 18 as above we obtain $q^{2\prime}$ and a substitution π_i', such that $\pi^{in}(x) \xrightarrow{P_i[\pi_i']} q_i^{2\prime}$ with reads($P_i[\pi_i']$) = reads($P_i[\pi^{in}]$), writes($P_i[\pi_i']$) $\mathcal{R}$ writes(L_i') and q_i^2 $\mathcal{R}$ $q_i^{2\prime}$. By transitivity of $\mathcal{R}$ also $P_i[\sigma]$ $\mathcal{R}$ $P_i[\pi_i']$ and $\sigma(x)$ $\mathcal{R}$ $q_i^{2\prime}$.

Substitution π

Let π^{out} be a substitution such that $\pi^{out}(y_i) = q_i^{2\prime}$ and for all $x \in$ vars(outs(P_i)), $\pi^{out}(x) = \pi_i'(x)$. Note that by Definition 16 vars(π^{in}) and vars(π^{out}) are disjoint. Let π be such that for all $x \in$ vars(π^{in}), $\pi(x) = \pi^{in}(x)$; for all $x \in$ vars(π^{out}), $\pi(x) = \pi^{out}(x)$ and for all other $x \in \mathcal{X}$ let $\pi(x) = \sigma(x)$. Now we have π $\mathcal{R}$ σ, and by $\mathcal{R}$ being a congruence also $t[\pi]$ $\mathcal{R}$ $t[\sigma]$, where t is the target of the conclusion in the congruence rule above.

Alternatively, $P_i[\sigma]$ is unobservable and we can either find a derivation like above, or $q_i^1 = q_i^2$ and then we are done.

Deriving $q^2 \longrightarrow^* q'$

For the remaining unobservable transitions, $q_i^2 \longrightarrow^* q_i'$, we can use the same congruence rule to derive $f(u_1[\pi], \ldots, \pi(y_i), \ldots, u_n[\pi])$ $\longrightarrow^*$ $f(u_1[\mu], \ldots, \mu(y_i), \ldots, u_n[\mu])$, where $\mu(y_i) = q_i'$ and for all other $x \in \mathcal{X}$, $\mu(x) = \pi(x)$.

Deriving $q^1 \xrightarrow{L'} q^2$

Case: d is not a congruence rule. If d is any other WB Cool MSOS rule, then for all $x_i \xrightarrow{P_i} y_i \in H$ (H is the set of premises and $x_i = u_i$ in the source of the conclusion), we have $\sigma(x_i) \xrightarrow{P_i[\sigma]} \sigma(y_i)$, where $P_i[\sigma]$ is observable and $\sigma(x_i) = p_i$. By induction hypothesis there is also a transition $\pi^{in}(x_i) = q_i^1 \xrightarrow{L_i'} q_i^2$, with reads($L_i'$) = reads($P_i[\sigma]$), writes($L_i'$) $\mathcal{R}$ writes($P_i[\sigma]$) and $\sigma(y_i)$ $\mathcal{R}$ q_i^2. Using Lemma 18 as above we obtain $q^{2\prime}$ and a substitution π_i' such that $\pi^{in}(x_i) \xrightarrow{P_i[\pi_i']} q_i^{2\prime}$ with reads($P_i[\pi_i']$) = reads($P_i[\pi^{in}]$), writes($P_i[\pi_i']$) $\mathcal{R}$ writes($P_i[\sigma]$) and q_i^2 $\mathcal{R}$ $q_i^{2\prime}$. By transitivity of $\mathcal{R}$ also $P_i[\sigma]$ $\mathcal{R}$ $P_i[\pi_i']$ and $\sigma(x_i)$ $\mathcal{R}$ $q_i^{2\prime}$.

Substitution π

For all $x_i \xrightarrow{P_i} y_i \in H$ with $\pi^{in}(x_i) = q_i^1$, let π_i^{out} be a substitution such that $\pi_i^{out}(y_i) = q_i^{2\prime}$ and for all $x \in$ vars(outs(P_i)), $\pi_i^{out}(x) = \pi_i'(x)$. Let π^{out} be the disjoint union of all such π_i^{out}. Let π be such that for all $x \in$ vars(π^{in}), $\pi(x) = \pi^{in}(x)$; for all $x \in$ vars(π^{out}), $\pi(x) = \pi^{out}(x)$ and for all other $x \in \mathcal{X}$ let $\pi(x) = \sigma(x)$. Now we have derived $f(u_1[\pi], \ldots, u_n[\pi]) \xrightarrow{P[\pi]} t[\pi]$, reads($P_x[\sigma]$) = reads($P_x[\pi]$) and writes($P_x[\sigma]$) $\mathcal{R}$ writes($P_x[\pi]$).

Deriving $q^2 \longrightarrow^* q'$

It remains to construct μ, such that $t[\pi] \longrightarrow^* t[\mu]$ and $t[\sigma]$ $\mathcal{R}$ $t[\mu]$. If t is a variable y appearing on the right-hand side of a premise, then $t[\pi] = \pi(y)$ which is some $q_i^{2\prime}$ from above, for which we have

$q_i^{2\prime} \longrightarrow^* q_i'$. Then set $\mu(y) = q_i'$ and we have $t[\pi] \longrightarrow^* t[\mu]$ and so $p = p_i' = t[\sigma] \mathcal{R}\ t[\pi] = q_i' = q$. If t is a term $f'(t_1, \ldots, t_m)$, then each variable y appearing on the right-hand side of a premise is receiving in t. Then by Definition 23 there is a congruence rule for each corresponding argument in subterms of t. For each $\pi(y) \longrightarrow^* q_i'$ we set $\mu(y) = q_i'$ and for all other variables x we set $\mu(x) = \pi(x)$. Then we can repeatedly use the congruence rules to obtain $t[\pi] \longrightarrow^* t[\mu]$.

Finally, we have substitutions π and μ, such that $q = f(q_1, \ldots, q_n) \longrightarrow^* f(u_1[\pi], \ldots, u_n[\pi]) \xrightarrow{(P[\pi])} t[\pi] \longrightarrow^* t[\mu] = q'$. By $\mathcal{R}$ being a congruence we have $p = t[\sigma] \mathcal{R}\ t[\mu] = q$ as required.

Condition 2. Assume $p = v(p_1, \ldots, p_n)$ for $v \in VC_\Sigma$. Then $f = v$ and so $q = v(q_1, \ldots, q_n)$. By induction hypothesis $p_i \mathcal{R}\ q_i$ and so (by reflexivity of $\longrightarrow^*$) $q \longrightarrow^* v(q_1, \ldots, q_n)$ with $p_i \mathcal{R}\ q_i$ as required.

We have shown that $\mathcal{R}$ satisfies both conditions of Definition 20 and thus that it is a weak MSOS bisimulation. Hence we have shown that $\approx_{msos}$ is a congruence under an MSOS specification in the Simply WB Cool MSOS format. □