Compositionality of Hennessy-Milner Logic through Structural Operational Semantics*

Wan Fokkink[1,2], Rob van Glabbeek[1], and Paulien de Wind[2]

[1] CWI, Department of Software Engineering
PO Box 94079, 1090 GB Amsterdam, The Netherlands
[2] Vrije Universiteit Amsterdam, Department of Theoretical Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

wan@cwi.nl, rvg@cs.stanford.edu, pdwind@cs.vu.nl
http://www.cwi.nl/~wan/
http://theory.stanford.edu/~rvg/
http://www.cs.vu.nl/~pdwind/

Abstract. This paper presents a method for the decomposition of HML formulae. It can be used to decide whether a process algebra term satisfies a HML formula, by checking whether subterms satisfy certain formulae, obtained by decomposing the original formula. The method uses the structural operational semantics of the process algebra. The main contribution of this paper is that an earlier decomposition method from Larsen [14] for the De Simone format is extended to the more general ntyft/ntyxt format without lookahead.

1 Introduction

In the past two decades, compositional methods have been developed for checking the validity of assertions in modal logics, used to describe the behaviour of processes. This means that the truth of an assertion for a composition of processes can be deduced from the truth of certain assertions for the components of the composition. Most research papers in this area focus on a particular process algebra.

Barringer, Kuiper & Pnueli [3] present (a preliminary version of) a compositional proof system for concurrent programs, which is based on a rich temporal logic, including operators from process logic [10] and LTL [20]. For modelling concurrent programs they define a language including assignment, conditional and while statements. Interaction between parallel components is done via shared variables.

In Stirling [22] modal proof systems are developed for subsets of CCS [16] (with and without silent actions) including only sequential and alternative composition, to decide the validity of formulae from Hennessy-Milner Logic (HML) [11]. In Stirling [23, 24] the results from [22] are extended, creating proof sys-

* To appear in: Proc. 14th International Symposium on *Fundamentals of Computation Theory*, Malmö, Sweden, LNCS, Springer, August 2003.

tems for subsets of CCS and SCCS [18] including asynchronous and synchronous parallelism and infinite behaviour, using ideas from [3]. In STIRLING [25] the proposals in [23, 24] are generalised to be able to cope with the restriction operator.

In WINSKEL [26] a method is given to decompose formulae with respect to each operation in SCCS. The language of assertions is HML with infinite conjunction and disjunction. This decomposition provides the foundations of Winskel's proof system for SCCS with modal assertions. In [27], [2] and [1] processes are described by specification languages inspired by CCS and CSP [6]. The articles describe compositional methods for deciding whether processes satisfy assertions from a modal μ-calculus [13].

LARSEN [14] developed a more general compositional method for deciding whether a process satisfies a certain property. Unlike the aforementioned methods, this method is not oriented towards a particular process algebra, but it is based on structural operational semantics [19], which provides process algebras and specification languages with an interpretation. A transition system specification, consisting of an algebraic signature and a set of transition rules of the form $\frac{\text{premises}}{\text{conclusion}}$, generates a transition relation between the closed terms over the signature. An example of a transition rule, for alternative composition, is

$$\frac{x_1 \xrightarrow{a} y}{x_1 + x_2 \xrightarrow{a} y}$$

meaning for states t_1, t_2 and u that if state t_1 can evolve into state u by the execution of action a, then so can state $t_1 + t_2$. Larsen showed how to decompose HML formulae with respect to a transition system specification in the De Simone format [21]. This format was originally put forward to guarantee that the bisimulation equivalence associated with a transition system specification is a congruence, meaning that bisimulation equivalence is preserved by all functions in the signature. LARSEN AND XINXIN [15] extended this decomposition method to HML with recursion (which is equivalent to the modal μ-calculus).

Since modal proof systems for specific process algebras are tailor-made, they may be more concise than the ones generated by the general decomposition method of Larsen (e.g., [23–25]). However, in some cases the general decomposition method does produce modal proof systems that are similar in spirit to those in the literature (e.g., [22, 26]).

In BLOOM, FOKKINK & VAN GLABBEEK [4] a method is given for decomposing formulae from a fragment of HML with infinite conjunctions, with respect to terms from any process algebra that has a structural operational semantics in ntyft/ntyxt format [9] without lookahead. This format is a generalisation of the De Simone format, and still guarantees that bisimulation equivalence is a congruence. The decomposition method is not presented in its own right, but is used in the derivation of congruence formats for a range of behavioural equivalences from VAN GLABBEEK [8].

In this paper the decomposition method from [4] is extended to full HML with infinite conjunction, again with respect to terms from any process algebra that has a structural operational semantics in ntyft/ntyxt format without lookahead.

2 Preliminaries

In this section we give the basic notions of structural operational semantics and Hennessy-Milner Logic (HML) that are needed to define our decomposition method.

2.1 Structural Operational Semantics

Structural operational semantics [19] provides a framework to give an operational semantics to programming and specification languages. In particular, because of its intuitive appeal and flexibility, structural operational semantics has found considerable application in the study of the semantics of concurrent processes.

Let V be an infinite set of variables. A syntactic object is called *closed* if it does not contain any variables from V.

Definition 1 (signature). *A* signature *is a collection Σ of* function symbols *$f \notin V$, equipped with a function $ar : \Sigma \to \mathbb{N}$. 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 $t \in \mathbb{T}(\Sigma)$ with $var(t) = \emptyset$. A Σ-substitution *σ is a partial function from V to $\mathbb{T}(\Sigma)$. If σ is a Σ-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 $T(\Sigma)$.*

In the remainder, let Σ denote a signature and A a set of actions, satisfying $|\Sigma| \leq |V|$ and $|A| \leq |V|$.

Definition 2 (literal). *A* positive Σ-literal *is an expression $t \xrightarrow{a} t'$ and a* negative Σ-literal *an expression $t \xrightarrow{a}\!\!\!\!\!/\;\;$ with $t, t' \in \mathbb{T}(\Sigma)$ and $a \in A$. For $t, t' \in \mathbb{T}(\Sigma)$ and $a \in A$, the literals $t \xrightarrow{a} t'$ and $t \xrightarrow{a}\!\!\!\!\!/\;\;$ are said to* deny *each other.*

Definition 3 (transition rule). *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 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.*

Definition 4 (transition system specification). *A* transition system specification *(TSS) is a pair (Σ, R) with R a collection of transition rules over Σ.*

Definition 5 (proof). *Let $P = (\Sigma, R)$ be a TSS. A* proof *of a transition rule $\frac{H}{\alpha}$ from P is a well-founded, upwardly branching tree of which the nodes are labelled by Σ-literals, and some of the leaves are marked "hypothesis", such that:*

- *the root is labelled by* α,
- H *contains the labels of the hypotheses, and*
- *if* β *is the label of a node* q *which is not an hypothesis and* K *is the set of labels of the nodes directly above* q*, then* $\frac{K}{\beta}$ *is a substitution instance of a transition rule in* R.

If a proof of $\frac{K}{\alpha}$ *from* P *exists, then* $\frac{K}{\alpha}$ *is* provable *from* P*, notation* $P \vdash \frac{K}{\alpha}$.

Definition 6 (transition relation). *A* transition relation *over* Σ *is a relation* $\rightarrow \subseteq T(\Sigma) \times A \times T(\Sigma)$. *We write* $p \xrightarrow{a} q$ *for* $(p, a, q) \in \rightarrow$ *and* $p \stackrel{a}{\not\rightarrow}$ *for* $\neg\exists q \in T(\Sigma) : p \xrightarrow{a} q$.

Thus a transition relation over Σ can be regarded as a set of closed positive Σ-literals (*transitions*). A TSS with only positive premises specifies a transition relation in a straightforward way as the set of all provable transitions. But it is much less trivial to associate a transition relation to a TSS with negative premises. Several solutions are proposed in Groote [9], Bol & Groote [5] and van Glabbeek [7]. From the latter we adopt the notion of a well-supported proof and a complete TSS.

Definition 7 (well-supported proof). *Let* $P = (\Sigma, R)$ *be a TSS. A* well-supported proof *of a closed literal* α *from* P *is a well-founded, upwardly branching tree of which the nodes are labelled by closed* Σ*-literals, such that:*

- *the root is labelled by* α*, and*
- *if* β *is the label of a node* q *and* K *is the set of labels of the nodes directly above* q*, then*
  1. *either* $\frac{K}{\beta}$ *is a closed substitution instance of a transition rule in* R
  2. *or* β *is negative and for every set* N *of negative closed literals such that* $P \vdash \frac{N}{\gamma}$ *for* γ *a closed literal denying* β*, a literal in* K *denies one in* N.

We say α *is ws-*provable *from* P*, notation* $P \vdash_{ws} \alpha$*, if a well-supported proof of* α *from* P *exists.*

In [7] it was noted that $\vdash_{ws}$ is *consistent*, in the sense that no standard TSS admits well-supported proofs of two literals that deny each other.

Definition 8 (completeness). *A TSS* P *is* complete *if for any closed literal* $p \stackrel{a}{\not\rightarrow}$ *either* $P \vdash_{ws} p \xrightarrow{a} p'$ *for some closed term* p' *or* $P \vdash_{ws} p \stackrel{a}{\not\rightarrow}$.

Now a TSS specifies a transition relation if and only if it is complete. The specified transition relation is then the set of all *ws*-provable transitions.

2.2 Hennessy-Milner Logic

A variety of modal logics have been developed to express properties of transition relations. Modal logic aims to formulate properties of process terms, and to identify terms that satisfy the same properties. Hennessy & Milner [11] have

defined a modal language, often called Hennessy-Milner Logic (HML), which characterises the bisimulation equivalence relation on process terms, assuming that each term has only finitely many outgoing transitions. This assumption can be discarded if infinite conjunctions are allowed [17, 12].

Definition 9 (Hennessy-Milner Logic). *Assume an action set A. The set $\mathbb{O}$ of* potential observations *or* modal formulae *is recursively defined by*

$$\varphi ::= \bigwedge_{i\in I} \varphi_i \mid \langle a\rangle\varphi \mid \neg\varphi$$

with $a \in A$ and I some index set.

Definition 10 (satisfaction relation). *Let $P = (\Sigma, R)$ be a TSS. The* satisfaction relation $\models_P \subseteq T(\Sigma) \times \mathbb{O}$ *is defined as follows, with $p \in T(\Sigma)$:*

$$\begin{array}{ll} p \models_P \bigwedge_{i\in I} \varphi_i & \text{iff } p \models_P \varphi_i \text{ for all } i \in I \\ p \models_P \langle a\rangle\varphi & \text{iff there is a } q \in T(\Sigma) \text{ such that } P \vdash_{ws} p \xrightarrow{a} q \text{ and } q \models_P \varphi \\ p \models_P \neg\varphi & \text{iff } p \not\models_P \varphi \end{array}$$

We will use the binary conjunction $\varphi_1 \wedge \varphi_2$ as an abbreviation of $\bigwedge_{i\in\{1,2\}} \varphi_i$, whereas $\top$ is an abbreviation for the empty conjunction. We identify formulae that are logically equivalent using the laws $\top \wedge \varphi \cong \varphi$, $\bigwedge_{i\in I}(\bigwedge_{j\in J_i} \varphi_j) \cong \bigwedge_{i\in I,\, j\in J_i} \varphi_j$ and $\neg\neg\varphi \cong \varphi$. This is justified because $\varphi \cong \psi$ implies $p \models_P \varphi \Leftrightarrow p \models_P \psi$.

3 Decomposing HML Formulae

In this section we will see how one can decompose HML formulae with respect to process terms. The TSS defining the transition relation on these terms should be in *ready simulation format* [4], allowing only ntyft/ntyxt rules [9] without lookahead.

Definition 11 (ntyxt,ntyft,nxytt). *An* ntytt rule *is a transition rule in which the right-hand sides of positive premises are variables that are all distinct, and that do not occur in the source. An ntytt rule is an* ntyxt rule *if its source is a variable, and an* ntyft rule *if its source contains exactly one function symbol and no multiple occurrences of variables. An ntytt rule is an* nxytt rule *if the left-hand sides of its premises are variables.*

Definition 12 (lookahead). *A transition rule has* no lookahead *if the variables occurring in the right-hand sides of its positive premises do not occur in the left-hand sides of its premises.*

Definition 13 (ready simulation format). *A TSS is in* ready simulation format *if its transition rules are ntyft or ntyxt rules that have no lookahead.*

Definition 14 (free). *A variable occurring in a transition rule is* free *if it does not occur in the source nor in the right-hand sides of the positive premises of this rule.*

Definition 15 (decent). *A transition rule is* decent *if it has no lookahead and does not contain free variables.*

In Bloom, Fokkink & van Glabbeek [4] for any TSS P in ready simulation format the collection of *P-ruloids* is defined. These are decent nxytt rules for which the following holds:

Theorem 1. *[4] Let P be a TSS in ready simulation format. Then $P \vdash_{ws} \sigma(t) \xrightarrow{a} p$ for t a term, p a closed term and σ a closed substitution, iff there are a P-ruloid $\frac{H}{t \xrightarrow{a} u}$ and a closed substitution σ' with $P \vdash_{ws} \sigma'(\alpha)$ for $\alpha \in H$, $\sigma'(t) = \sigma(t)$ and $\sigma'(u) = p$.*

Given a TSS $P = (\Sigma, R)$ in ready simulation format, the following definition assigns to each term $t \in \mathbb{T}(\Sigma)$ and each observation $\varphi \in \mathbb{O}$ a collection $t_P^{-1}(\varphi)$ of *decomposition mappings* $\psi : V \to \mathbb{O}$. Each of these mappings $\psi \in t_P^{-1}(\varphi)$ guarantees, given a closed substitution σ, that $\sigma(t)$ satisfies φ if $\sigma(x)$ satisfies the formula $\psi(x)$ for all $x \in var(t)$. Moreover, whenever for some closed substitution σ the term $\sigma(t)$ satisfies φ, there must be a decomposition mapping $\psi \in t_P^{-1}(\varphi)$ with $\sigma(x)$ satisfying $\psi(x)$ for all $x \in var(t)$. This is formalised in Theorem 2 and proven thereafter.

Definition 16. *Let $P = (\Sigma, R)$ be a TSS in ready simulation format. Then $\cdot_P^{-1} : \mathbb{T}(\Sigma) \to (\mathbb{O} \to \mathcal{P}(V \to \mathbb{O}))$ is defined by:*

- *$\psi \in t_P^{-1}(\langle a \rangle \varphi)$ iff there is a P-ruloid $\frac{H}{t \xrightarrow{a} u}$ and a $\chi \in u_P^{-1}(\varphi)$ and $\psi : V \to \mathbb{O}$ is given by*

$$\psi(x) = \begin{cases} \chi(x) \wedge \bigwedge\limits_{(x \xrightarrow{b} y) \in H} \langle b \rangle \chi(y) \wedge \bigwedge\limits_{(x \xrightarrow{c} \not\to) \in H} \neg \langle c \rangle \top & \text{if } x \in var(t) \\ \top & \text{if } x \notin var(t) \end{cases}$$

- *$\psi \in t_P^{-1}(\bigwedge_{i \in I} \varphi_i)$ iff*

$$\psi(x) = \bigwedge_{i \in I} \psi_i(x)$$

  where $\psi_i \in t_P^{-1}(\varphi_i)$ for $i \in I$.
- *$\psi \in t_P^{-1}(\neg \varphi)$ iff there is a function $h : t_P^{-1}(\varphi) \to var(t)$ and $\psi : V \to \mathbb{O}$ is given by*

$$\psi(x) = \bigwedge_{\chi \in h^{-1}(x)} \neg \chi(x)$$

When clear from the context, the subscript P will be omitted.

It is not hard to see that if $\psi \in t_P^{-1}(\varphi)$ then $\psi(x) = \top$ for all $x \notin var(t)$.

Theorem 2. *Let $P = (\Sigma, R)$ be a complete TSS in ready simulation format. Let $\varphi \in \mathbb{O}$. For any term $t \in \mathbb{T}(\Sigma)$ and closed substitution $\sigma : V \to T(\Sigma)$ one has*

$$\sigma(t) \models \varphi \quad \Leftrightarrow \quad \exists \psi \in t^{-1}(\varphi) \forall x \in var(t) \big(\sigma(x) \models \psi(x)\big)$$

Proof. With induction on the structure of φ.

- $\varphi = \langle a \rangle \varphi'$
  $\boxed{\Rightarrow}$ Suppose $\sigma(t) \models \langle a \rangle \varphi'$. Then by Definition 10 there is a $p \in T(\Sigma)$ with $P \vdash_{ws} \sigma(t) \xrightarrow{a} p$ and $p \models \varphi'$. Thus, by Theorem 1 there must be a P-ruloid $\frac{H}{t \xrightarrow{a} u}$ and a closed substitution σ' with $P \vdash_{ws} \sigma'(\alpha)$ for $\alpha \in H$, $\sigma'(t) = \sigma(t)$, i.e. $\sigma'(x) = \sigma(x)$ for $x \in var(t)$, and $\sigma'(u) = p$. Since $\sigma'(u) \models \varphi'$, the induction hypothesis can be applied, and there must be a $\chi \in u^{-1}(\varphi')$ such that $\sigma'(z) \models \chi(z)$ for all $z \in var(u)$. Furthermore $\sigma'(z) \models \chi(z) = \top$ for all $z \notin var(u)$. Now define ψ as indicated in Definition 16. By definition, $\psi \in t^{-1}(\langle a \rangle \varphi')$. Let $x \in var(t)$. For $(x \xrightarrow{b} y) \in H$ one has $P \vdash_{ws} \sigma'(x) \xrightarrow{b} \sigma'(y)$ and $\sigma'(y) \models \chi(y)$, so $\sigma'(x) \models \langle b \rangle \chi(y)$. Moreover, for $(x \xrightarrow{c}\!\!\!\!\!\!/\;) \in H$ one has $P \vdash_{ws} \sigma'(x) \xrightarrow{c}\!\!\!\!\!\!/\;$, so the consistency of $\vdash_{ws}$ yields $P \nvdash_{ws} \sigma'(x) \xrightarrow{c} q$ for all $q \in T(\Sigma)$, and thus $\sigma'(x) \models \neg\langle c \rangle \top$. It follows that $\sigma(x) = \sigma'(x) \models \psi(x)$.
  $\boxed{\Leftarrow}$ Now suppose that there is a $\psi \in t^{-1}(\langle a \rangle \varphi')$ such that $\sigma(x) \models \psi(x)$ for all $x \in var(t)$. This means that there is a P-ruloid

$$\frac{\{x \xrightarrow{a_i} y_i \mid i \in I_x,\ x \in var(t)\} \cup \{x \xrightarrow{b_j}\!\!\!\!\!\!/\;\mid j \in J_x,\ x \in var(t)\}}{t \xrightarrow{a} u}$$

  and a decomposition mapping $\chi \in u^{-1}(\varphi')$ such that, for all $x \in var(t)$,

$$\sigma(x) \models \chi(x) \wedge \bigwedge_{i \in I_x} \langle a_i \rangle \chi(y_i) \wedge \bigwedge_{j \in J_x} \neg\langle b_j \rangle \top$$

  By Definition 10 it follows that, for $x \in var(t)$ and $i \in I_x$, $P \vdash_{ws} \sigma(x) \xrightarrow{a_i} p_i$ for some $p_i \in T(\Sigma)$ with $p_i \models \chi(y_i)$. Moreover, for $x \in var(t)$ and $j \in J_x$, $P \nvdash_{ws} \sigma(x) \xrightarrow{b_j} q$ for all $q \in T(\Sigma)$, so by the completeness of P, $P \vdash_{ws} \sigma(x) \xrightarrow{b_j}\!\!\!\!\!\!/\;$. Let σ' be a closed substitution with $\sigma'(x) = \sigma(x)$ for $x \in var(t)$ and $\sigma'(y_i) = p_i$ for $i \in I_x$ and $x \in var(t)$. Here we use that the variables x and y_i are all different. Now $\sigma'(z) \models \chi(z)$ for $z \in var(u)$, using that u contains only variables that occur in t or in the premises of the ruloid. Thus the induction hypothesis can be applied, and $\sigma'(u) \models \varphi'$. Moreover, $P \vdash_{ws} \sigma'(x) \xrightarrow{a_i} \sigma'(y_i)$ for $x \in var(t)$ and $i \in I_x$, and $P \vdash_{ws} \sigma'(x) \xrightarrow{b_j}\!\!\!\!\!\!/\;$ for $x \in var(t)$ and $j \in J_x$. So, by Theorem 1, $P \vdash_{ws} \sigma'(t) \xrightarrow{a} \sigma'(u)$, which implies $\sigma(t) = \sigma'(t) \models \langle a \rangle \varphi'$.

– $\varphi = \bigwedge_{i\in I} \varphi_i$
$\sigma(t) \models \bigwedge_{i\in I} \varphi_i \;\Leftrightarrow\; \forall i{\in}I : \sigma(t) \models \varphi_i$
$\Leftrightarrow\; \forall i{\in}I\; \exists \psi_i{\in}t^{-1}(\varphi_i)\; \forall x{\in}var(t) : \sigma(x) \models \psi_i(x)$
$\Leftrightarrow\; \exists \psi{\in}t^{-1}(\bigwedge_{i\in I} \varphi_i)\; \forall x{\in}var(t) : \sigma(x) \models \psi(x).$
– $\varphi = \neg\varphi'$
$\boxed{\Rightarrow}$ Suppose $\sigma(t) \models \neg\varphi'$. Then by Definition 10 we have $\sigma(t) \not\models \varphi'$. Using the induction hypothesis, there is no $\chi \in t^{-1}(\varphi')$ such that $\sigma(x) \models \chi(x)$ for all $x \in var(t)$. So for all $\chi \in t^{-1}(\varphi')$ there is an $x \in var(t)$ such that $\sigma(x) \models \neg\chi(x)$. Let us denote this x as $h(\chi)$, so that we obtain a function $h : t^{-1}(\varphi') \to var(t)$ such that $\sigma(h(\chi)) \models \neg\chi(h(\chi))$ for all $\chi \in t^{-1}(\varphi')$. Define $\psi \in t^{-1}(\neg\varphi')$ as indicated in Definition 16, using h. Let $x \in var(t)$. If $x = h(\chi)$ for some $\chi \in t^{-1}(\varphi')$ then $\sigma(x) \models \neg\chi(x)$. Hence, $\sigma(x) \models \bigwedge_{\chi\in h^{-1}(x)} \neg\chi(x) = \psi(x)$.
$\boxed{\Leftarrow}$ Suppose that there is a $\psi \in t^{-1}(\neg\varphi')$ such that $\sigma(x) \models \psi(x)$ for all $x \in var(t)$. By Definition 16 there is a function $h : t^{-1}(\varphi') \to var(t)$ such that $\psi(x) = \bigwedge_{\chi\in h^{-1}(x)} \neg\chi(x)$ for all $x \in var(t)$. So for all $x \in var(t)$ and for all $\chi \in h^{-1}(x)$ we have that $\sigma(x) \models \neg\chi(x)$. In other words, for all $\chi \in t^{-1}(\varphi')$,we have $\sigma(h(\chi)) \models \neg\chi(h(\chi))$. So $\neg\exists\chi \in t^{-1}(\varphi')\forall x \in var(t)\big(\sigma(x) \models \chi(x)\big)$. Then using the induction hypothesis, we have $\sigma(t) \not\models \varphi'$, so $\sigma(t) \models \neg\varphi'$.

We give a few examples of the application of Definition 16.

Example 1. Let $A = \{a, b\}$ and let $P = (\Sigma, R)$ with Σ consisting of the constant c and the binary function symbol f and R is:

$$c \xrightarrow{a} c \qquad \frac{x_1 \xrightarrow{a} y}{f(x_1, x_2) \xrightarrow{b} y} \qquad \frac{x_2 \xrightarrow{a} y \quad x_1 \not\xrightarrow{b}}{f(x_1, x_2) \xrightarrow{b} y}$$

This TSS is complete and in ready simulation format. We proceed to compute $f(x_1, x_2)^{-1}(\langle b\rangle\top)$. There are two P-ruloids with a conclusion of the form $f(x_1, x_2) \xrightarrow{b} \_$, namely $\frac{x_1 \xrightarrow{a} y}{f(x_1,x_2) \xrightarrow{b} y}$ and $\frac{x_2 \xrightarrow{a} y \quad x_1 \not\xrightarrow{b}}{f(x_1,x_2) \xrightarrow{b} y}$. According to Definition 16, we have $f(x_1, x_2)^{-1}(\langle b\rangle\top) = \{\psi_1, \psi_2\}$ with ψ_1 and ψ_2 as defined below, using $\chi \in y^{-1}(\top)$ (so $\chi(x) = \top$ for all variables $x \in V$):

$$\psi_1(x_1) = \chi(x_1) \wedge \langle a\rangle\chi(y) = \top \wedge \langle a\rangle\top = \langle a\rangle\top$$
$$\psi_1(x_2) = \chi(x_2) = \top$$
$$\psi_1(x) = \top \text{ for } x \notin var(f(x_1, x_2))$$

$$\psi_2(x_1) = \chi(x_1) \wedge \neg\langle b\rangle\top = \top \wedge \neg\langle b\rangle\top = \neg\langle b\rangle\top$$
$$\psi_2(x_2) = \chi(x_2) \wedge \langle a\rangle\chi(y) = \top \wedge \langle a\rangle\top = \langle a\rangle\top$$
$$\psi_2(x) = \top \text{ for } x \notin var(f(x_1, x_2))$$

By Theorem 2 a closed term $f(u_1, u_2)$ can execute a b if and only if the closed term u_1 can execute an a, or the closed term u_1 can not execute a b and the closed term u_2 can execute an a. Looking at the premises, this is what we would expect.

Example 2. Using the TSS and the mappings $\psi_1, \psi_2 \in f(x_1, x_2)^{-1}(\langle b\rangle\top)$ from Example 1, we can compute $f(x_1, x_2)^{-1}(\neg\langle b\rangle\top)$. There are four possible functions $h : f(x_1, x_2)^{-1}(\langle b\rangle\top) \to var(f(x_1, x_2))$, yielding four possible definitions of $\psi \in f(x_1, x_2)^{-1}(\neg\langle b\rangle\top)$.

1. If $h(\psi_1) = h(\psi_2) = x_1$ then
$$\psi(x_1) = \neg\psi_1(x_1) \wedge \neg\psi_2(x_1) = \neg\langle a\rangle\top \wedge \neg\neg\langle b\rangle\top = \neg\langle a\rangle\top \wedge \langle b\rangle\top$$
$$\psi(x_2) = \top$$
2. If $h(\psi_1) = h(\psi_2) = x_2$ then
$$\psi(x_1) = \top$$
$$\psi(x_2) = \neg\psi_1(x_2) \wedge \neg\psi_2(x_2) = \neg\top \wedge \neg\langle a\rangle\top$$
3. If $h(\psi_1) = x_1$ and $h(\psi_2) = x_2$ then
$$\psi(x_1) = \neg\psi_1(x_1) = \neg\langle a\rangle\top$$
$$\psi(x_2) = \neg\psi_2(x_2) = \neg\langle a\rangle\top$$
4. If $h(\psi_1) = x_2$ and $h(\psi_2) = x_1$ then
$$\psi(x_1) = \neg\psi_2(x_1) = \neg\neg\langle b\rangle\top = \langle b\rangle\top$$
$$\psi(x_2) = \neg\psi_1(x_2) = \neg\top$$

By Theorem 2 a closed term $f(u_1, u_2)$ can *not* execute a b if and only if (1) the closed term u_1 can execute a b but not an a, or (3) the closed term u_1 can not execute an a and the closed term u_2 can not execute an a. Looking at the premises, this is again what we would expect. The other two possibilities (2) and (4) do not qualify, since no term can ever satisfy $\neg\top$.

A little less obvious example is the following:

Example 3. Let $A = \{a, b\}$ and let $P = (\Sigma, R)$ with Σ consisting of the constant c and the unary function symbol f and R is:

$$c \xrightarrow{a} c \qquad \frac{x \xrightarrow{a} y}{f(x) \xrightarrow{b} y} \qquad \frac{x \xrightarrow{b} y}{f(x) \xrightarrow{a} f(y)}$$

This TSS is complete and in ready simulation format. We proceed to compute $f(f(x))^{-1}(\langle b\rangle\langle a\rangle\top)$. The only P-ruloid that has a conclusion $f(f(x)) \xrightarrow{b} \_$ is $\frac{x \xrightarrow{b} y}{f(f(x)) \xrightarrow{b} f(y)}$. So for each $\psi \in f(f(x))^{-1}(\langle b\rangle\langle a\rangle\top)$, $\psi(x) = \chi(x) \wedge \langle b\rangle\chi(y)$ with $\chi \in f(y)^{-1}(\langle a\rangle\top)$. The only P-ruloid that has a conclusion $f(y) \xrightarrow{a} \_$ is $\frac{y \xrightarrow{b} z}{f(y) \xrightarrow{a} f(z)}$. So $\chi(y) = \chi'(y) \wedge \langle b\rangle\chi'(z)$ with $\chi' \in f(z)^{-1}(\top)$. Since $\chi'(y) = \chi'(z) = \top$ we have $\chi(y) = \langle b\rangle\top$. Moreover $x \notin var(f(y))$ implies $\chi(x) = \top$. Hence $\psi(x) = \langle b\rangle\langle b\rangle\top$.

By Theorem 2 a closed term $f(f(u))$ can execute a b followed by an a if and only if the closed term u can execute two consecutive b's.

The following example shows that in Theorem 2 it is essential that the TSS is complete. That is, the theorem would fail if we would take the transition relation induced by a TSS to consist of those transitions for which a well-supported proof exists.

Example 4. Let $A = \{a, b\}$ and let $P = (\Sigma, R)$ with Σ consisting of the constant c and the unary function symbol f and R is:

$$\frac{x \overset{a}{\not\longrightarrow}}{f(x) \overset{b}{\longrightarrow} c} \qquad \frac{c \overset{a}{\not\longrightarrow}}{c \overset{a}{\longrightarrow} c}$$

This TSS, which is in ready simulation format, is incomplete. For example, neither $P \vdash_{ws} c \overset{a}{\longrightarrow} t$ for a closed term t nor $P \vdash_{ws} c \overset{a}{\not\longrightarrow}$.

Let us assume that the transition relation induced by this TSS consists of those transitions for which a well-supported proof exists. Then there is no a-transition for c and no b-transition for $f(c)$, so $c \not\models \langle a\rangle\top$ and $f(c) \not\models \langle b\rangle\top$.

The only P-ruloid is $\frac{x \overset{a}{\not\longrightarrow}}{f(x) \overset{b}{\longrightarrow} c}$. Hence Theorem 2 would yield $f(c) \models \langle b\rangle\top \Leftrightarrow c \models \neg\langle a\rangle\top \Leftrightarrow c \not\models \langle a\rangle\top$. Since this is false, Theorem 2 would fail with respect to P.

References

1. H. R. Andersen, C. Stirling & G. Winskel (1994): *A compositional proof system for the modal μ-calculus.* In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, Paris, France, pp. 144–153.
2. H. R. Andersen & G. Winskel (1992): *Compositional checking of satisfaction. Formal Methods in System Design* 1(4), pp. 323–354.
3. H. Barringer, R. Kuiper & A. Pnueli (1984): *Now you may compose temporal logic specifications.* In *ACM Symposium on Theory of Computing (STOC '84)*, ACM Press, Baltimore, USA, pp. 51–63.
4. B. Bloom, W. J. Fokkink & R. J. van Glabbeek (2003): *Precongruence formats for decorated trace semantics. ACM Transactions on Computational Logic.* To appear.
5. R. Bol & J. F. Groote (1996): *The meaning of negative premises in transition system specifications. Journal of the ACM* 43(5), pp. 863–914.
6. S. D. Brookes, C. A. R. Hoare & A. W. Roscoe (1984): *A theory of communicating sequential processes. Journal of the ACM* 31(3), pp. 560–599.
7. R. J. van Glabbeek (1996): *The meaning of negative premises in transition system specifications II.* In F. Meyer auf der Heide & B. Monien, editors: *Automata, Languages and Programming, 23rd Colloquium (ICALP '96)*, *Lecture Notes in Computer Science* 1099, Springer-Verlag, Paderborn, Germany, pp. 502–513.
8. R. J. van Glabbeek (2001): *The linear time – branching time spectrum I: The semantics of concrete, sequential processes.* In J. A. Bergstra, A. Ponse & S. A. Smolka, editors: *Handbook of Process Algebra*, chapter 1, Elsevier, pp. 3–99.
9. J. F. Groote (1993): *Transition system specifications with negative premises. Theoretical Computer Science* 118(2), pp. 263–299.

10. D. Harel, D. Kozen & R. Parikh (1982): *Process logic: Expressiveness, decidability, completeness. Journal of Computer and System Sciences* 25(2), pp. 144–170.
11. M. C. B. Hennessy & R. Milner (1985): *Algebraic laws for non-determinism and concurrency. Journal of the ACM* 32(1), pp. 137–161.
12. M. C. B. Hennessy & C. Stirling (1985): *The power of the future perfect in program logics. Information and Control* 67(1–3), pp. 23–52.
13. D. Kozen (1983): *Results on the propositional μ-calculus. Theoretical Computer Science* 27(3), pp. 333–354.
14. K. G. Larsen (1986): *Context-Dependent Bisimulation between Processes.* PhD thesis, University of Edinburgh, Edinburgh.
15. K. G. Larsen & L. Xinxin (1991): *Compositionality through an operational semantics of contexts. Journal of Logic and Computation* 1(6), pp. 761–795.
16. R. Milner (1980): *A Calculus of Communicating Systems.* Springer-Verlag. Volume 92 of *Lecture Notes in Computer Science.*
17. R. Milner (1981): *A modal characterization of observable machine-behaviour.* In E. Astesiano & C. Böhm, editors: *CAAP '81: Trees in Algebra and Programming, 6^{th} Colloquium, Lecture Notes in Computer Science* 112, Springer-Verlag, Genoa, pp. 25–34.
18. R. Milner (1983): *Calculi for synchrony and asynchrony. Theoretical Computer Science* 25(3), pp. 267–310.
19. G. D. Plotkin (1981): *A structural approach to operational semantics.* Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark.
20. A. Pnueli (1981): *The temporal logic of concurrent programs. Theoretical Computer Science* 13, pp. 45–60.
21. R. De Simone (1985): *Higher-level synchronising devices in* Meije–*SCCS. Theoretical Computer Science* 37(3), pp. 245–267.
22. C. Stirling (1985): *A proof-theoretic characterization of observational equivalence. Theoretical Computer Science* 39(1), pp. 27–45.
23. C. Stirling (1985): *A complete compositional modal proof system for a subset of CCS.* In W. Brauer, editor: *Automata, Languages and Programming, 12th Colloquium (ICALP '85), Lecture Notes in Computer Science* 194, Springer-Verlag, pp. 475–486.
24. C. Stirling (1985): *A complete modal proof system for a subset of SCCS.* In H. Ehrig, C. Floyd, M. Nivat & J. W. Thatcher, editors: *Mathematical Foundations of Software Development: Proceedings of the Joint Conference on Theory and Practice of Software Development (TAPSOFT), Volume 1: Colloquium on Trees in Algebra and Programming (CAAP '85), Lecture Notes in Computer Science* 185, Springer-Verlag, pp. 253–266.
25. C. Stirling (1987): *Modal logics for communicating systems. Theoretical Computer Science* 49(2-3), pp. 311–347.
26. G. Winskel (1986): *A complete proof system for SCCS with modal assertions. Fundamenta Informaticae* IX, pp. 401–420.
27. G. Winskel (1990): *On the compositional checking of validity (extended abstract).* In J. C. M. Baeten & J. W. Klop, editors: *CONCUR '90: Theories of Concurrency: Unification and Extension, Lecture Notes in Computer Science* 458, Springer-Verlag, Amsterdam, The Netherlands, pp. 481–501.