Context-Free Processes and Push-Down Processes

MSc Thesis *(Afstudeerscriptie)*

written by

Zeno de Hoop
(born August 17, 1991 in Goes, Netherlands)

under the supervision of **Prof. Dr. Jos Baeten**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
August 29, 2017	Prof. Dr. Benedikt Loewe
	Prof. Dr. Jos Baeten
	Prof. Dr. Wan Fokkink
	Prof. Dr. Yde Venema

Institute for Logic, Language and Computation

Contents

Abstract

The purpose of this thesis is to examine in which cases context-free processes and push-down processes are the same. In particular, we depart from the well-known case of language-equivalence and instead look at processes using process theory and more fine-grained equivalences, such as bisimulation and contrasimulation.

We identify two difficulties when looking at process specifications: head-recursion and transparency. Here, two new results are achieved: we prove that when excluding transparency, context-free processes and push-down processes are equivalent up to at least branching bisimilarity without explicit divergences. When including transparency, we prove that they are equivalent up to at least contrasimulation. Finally, we present a new result where, when one excludes head-recursion and adopts a modified definition of sequential composition, one can improve the equivalence to strong bisimulation. Some of the drawbacks of this modification are discussed as well.

Acknowledgements

The first person I would like to extend my gratitude to is my supervisor, Jos Baeten, without whom the creation of this thesis would not have been possible. My interest in the topic of this thesis was sparked by his course "Computability and Interaction", and it was his enthusiastic teaching that made this one of the most memorable courses in my Master's. He also encouraged us to try our hand at open problems in the field, which was the starting point of many of the ideas found in this thesis.

This brings me to two other people who deserve acknowledgement. The first is Sander in 't Veld, a fellow student in the aforementioned course. It was one of his ideas, unpolished at the time, that eventually grew into one of the results found in this thesis (in particular, the result regarding head-recursion). This unpolished idea was then taken up by Fei Yang, who developed this idea further, bringing it very close to the final state found in this thesis. Finding a proof for this idea was one of the major breakthroughs in the process of writing this thesis, so I thank them both for their insight. I would further like to thank Fei Yang for having taken the time to proofread my thesis.

Lastly, this thesis would not have been possible without the support of the people around me.

1 Introduction

The subject of this thesis is the relation between *Context-Free Processes* and *Push-Down Processes*. However, since that statement is, in itself, too general to be useful, some further clarifications are in order.

To begin with, it is a well-known fact that for each context-free grammar one can find a push-down automaton that has the same language, and vice versa. This result can be found in many text-books on process or automata theory, among which [12].

However, with language-equivalence, one sees the behaviour of a system as merely the sequence of observable actions. This notion can be widened. One could for instance look at behaviour of a system as not only the total of events or actions that it can perform and the order in which they can be executed (as is the case with language-equivalence), but one could refine behaviour to include when certain "decisions" are made. The latter is completely abstracted from with language-equivalence: a process which decides its final action as soon as it begins, and one that only decides its final action at the very end are, under language-equivalence, indistinguishable.

With this in mind, we can make the method and purpose of this thesis a bit more precise. A *process*, in the context of this thesis, will be a labelled transition system. This is a generalization of non-deterministic finite automata. This also means that we will only consider discrete processes.

The differences between processes, when given as labelled transition systems, can be captured by equivalence relations on these systems. Language-equivalence is the most well-known of these equivalences, but there are many more. Of particular interest to this thesis will be *Bisimulation* and *Contrasimulation*. For more information on equivalence relations on transition systems in general, and a description of the lattice that these relations form, we refer to [8] and [7].

The primary question of this thesis will therefore be as follows: when is a process from a context-free grammar the same as a process from a push-down automaton? In particular, what is the finest equivalence relation realising this?

The structure of the thesis will be as follows:

- In the first section we present a number of definitions and relevant previous results.
- In the second section we give a proof of a conjecture from [14]; we show that for any sequential specification with head-recursion (without transparent names) one can construct a push-down automaton so that the associated transition systems are equivalent up to branching bisimulation.
- In the third section we give a proof of another conjecture found in [14];

we show that for any sequential specification with transparency (without head-recursion) one can construct a push-down automaton so that the associated transition systems are equivalent up to contrasimulation.

- In the fourth section we combine the results from the previous two sections: given a sequential specification, without restrictions on head-recursion or transparency, one can construct a push-down automaton so that the associated transition systems are equivalent up to contrasimulation.

- Finally, in the fifth section, we show that, if one modifies one rule of the operational semantics, one can find the same result as detailed in the third section, but up to strong bisimilarity. We also briefly discuss the drawbacks of this rule change.

As mentioned, most of the original results of this thesis are proofs of conjectures found in [14]. We also refer to this work for a more detailed treatment of many of the topics found in the chapter dealing with the preliminaries.

2 Preliminaries

This section covers a large part of the definitions used throughout the paper. The first subject covered is push-down automata, followed by equivalences on push-down automata, and finally process theory.

2.1 Push-Down Automata

As mentioned in the introduction, a push-down automaton is a transition system that goes from state to state by performing actions, and which has at its disposal a stack that may contain data. We will formally define the notion of a Push-Down Automaton that will be used in this paper. The definitions used in this section can be found in, for instance, [12].

Before detailing the definition of a push-down automaton, we will first define transition systems, as these will be routinely used throughout this work.

Definition 2.1. Let $\mathcal{A}$ be a set of actions, and $\mathcal{A}_\tau = \mathcal{A} \cup \{\tau\}$, where τ is an unobservable action. A *labelled transition system* T is defined as a four-tuple $(\mathcal{S}, \rightarrow, \uparrow, \downarrow)$ where:

- $\mathcal{S}$ is a (possibly infinite) set of states.
- $\rightarrow \subseteq \mathcal{S} \times \mathcal{A}_\tau \times \mathcal{S}$ is a labelled transition. (s, a, t) is usually written as $s \xrightarrow{a} t$.
- $\uparrow \in \mathcal{S}$ is the initial state.
- $\downarrow \subseteq \mathcal{S}$ is the set of terminating states.

We also introduce the following two bits of notation: with $\twoheadrightarrow$ we mean the transitive-reflexive closure of $\xrightarrow{\tau}$, and $\rightarrow^+$ indicates the transitive closure of $\xrightarrow{\tau}$.

We will now define what it means to be a push-down automaton. Informally, a push-down automaton is a transition system which has access to a stack in which it can store data and later retrieve it.

Definition 2.2. A *Push-Down Automaton* (PDA) M consists of a six-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{D}, \rightarrow, \uparrow, \downarrow)$ where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{D}$ is a finite set of data.
- $\rightarrow \subseteq \mathcal{S} \times \mathcal{A}_\tau \times \mathcal{D}_\perp \times \mathcal{D}^* \times \mathcal{S}$ is a transition relation on $\mathcal{S}$, with every transition labelled by an element of $\mathcal{A}_\tau \times \mathcal{D}_\perp \times \mathcal{D}^*$.

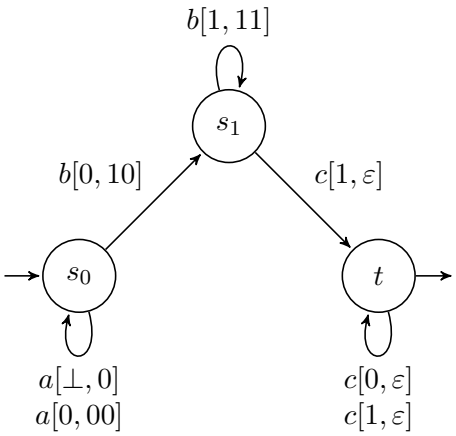

Figure 1: example PDA M

- $\uparrow \in \mathcal{S}$ is the initial state.
- $\downarrow \subseteq \mathcal{S}$ is the collection of terminating states.

This definition needs some further clarification. By $\mathcal{A}_\tau$ is meant the set $\mathcal{A} \cup \{\tau\}$, where τ is an unobservable action, not originally found in $\mathcal{A}$, also referred to as silent or internal action. By $\mathcal{D}_\bot$ is meant the set $\mathcal{D} \cup \{\bot\}$, where $\bot$ is a special symbol indicating that the stack is empty. Finally, by $\mathcal{D}^*$ is meant a sequence of data symbols. As in [14], we often let δ and ζ range over $\mathcal{D}^*$, and use ε for the empty string.

In terms of notation, if $(s, a, d, \delta, t) \in \rightarrow$, we will write $s \xrightarrow{a[d,\delta]} t$. The meaning of this is as follows: if the PDA is at state s, and the top element of the stack is d, then the PDA can perform action a to reach state t. The d on top of the stack is replaced by δ. Note that a need not be observable, and that δ can be empty, that is, ε. A special case of note is $s \xrightarrow{a[\bot,\delta]} t$. This is an action that can be performed if the stack is empty.

Example 2.1. In figure 1, one can see an example of a PDA. In its initial state, s_0, firstly, M can perform the action a an arbitrary number of times. Each time a is performed, the symbol 0 is added to the stack. After the action a has been performed at least one time, M can perform the action b to go to state s_1, which will add a 1 to the stack. At state s_1, M can perform the action b any number of times, and finally it can perform c to move to the state t. At t one can then perform c as many times as there are items in the stack.

We will call a string $\eta \in \mathcal{A}^*$ accepted if a PDA can be at a final state after performing all the actions in η. However, as the previous example demonstrates, this is not completely unambiguous: depending on whether

or not the stack starts empty, and whether or not one requires that the stack is empty before terminating, both influence the set of accepted strings.

Regarding the initial state of the stack, we will throughout this work assume that in a PDA's initial state the stack is empty.

When it comes to the latter issue, there are three possible interpretations:

- FS (Final State): a PDA can terminate when it is in a final state
- ES (Empty Stack): a PDA can terminate whenever its stack is empty
- FSES (Final State Empty Stack): a PDA can terminate when its stack is empty, and it is in a final state.

So, looking at the previous example, we firstly see that under the FSES interpretation, the set of accepted strings is $\{a^{1+n}b^{1+m}c^{2+n+m}|n,m \in \mathbb{N}\}$. This is almost the same as under the ES interpretation, except under the ES interpretation the empty string is also accepted. Finally, under the FS interpretation the accepted strings are $\{a^{1+n}b^{1+m}c^{p}|n,m \in \mathbb{N}, 0 \leq p \leq 2+n+m\}$.

In this work we will use the FSES interpretation (unless otherwise specified). For further information on this topic we refer to [14].

Definition 2.3. Under a given interpretation, and given a PDA M, we call the set $\mathcal{L}(M) \subseteq \mathcal{A}^*$ of all strings accepted by M the *language* of M.

Each PDA can also be transformed into a transition system.

Definition 2.4. Let $M = (\mathcal{S}, \mathcal{A}, \mathcal{D}, \rightarrow, \uparrow, \downarrow)$ be a PDA. We then define the associated transition system $\mathcal{T}(M) = (\mathcal{S}_{\mathcal{T}(M)}, \rightarrow_{\mathcal{T}(M)}, \uparrow_{\mathcal{T}(M)}, \downarrow_{\mathcal{T}(M)})$ as follows:

- $\mathcal{S}_{\mathcal{T}(M)} = \mathcal{S} \times \mathcal{D}^*$.
- The contents of the set $\rightarrow_{\mathcal{T}(M)}$ are:
  - $(s, d\zeta) \xrightarrow{a} (t, \delta\zeta)$ iff $s \xrightarrow{a[d,\delta]} t$ for all $s,t \in \mathcal{S}$, $a \in \mathcal{A}$, $d \in \mathcal{D}$, and $\delta, \zeta \in \mathcal{D}^*$.
  - $(s, \varepsilon) \xrightarrow{a} (t, \delta)$ iff $s \xrightarrow{a[\perp,\delta]} t$.
- The state $(\uparrow, \varepsilon)$ is the initial state of $\mathcal{T}(M)$.
- The final states of $\mathcal{T}(M)$ are $\downarrow_{\mathcal{T}(M)} = \{(s,\delta)|s \in \downarrow, \delta \in \mathcal{D}^*\}$.

The last thing we will define in this subsection is the concept of push- and pop-transitions. In the next section we will prove that under the relevant equivalence, all PDA's can be transformed into PDA's using only push- or pop-transitions.

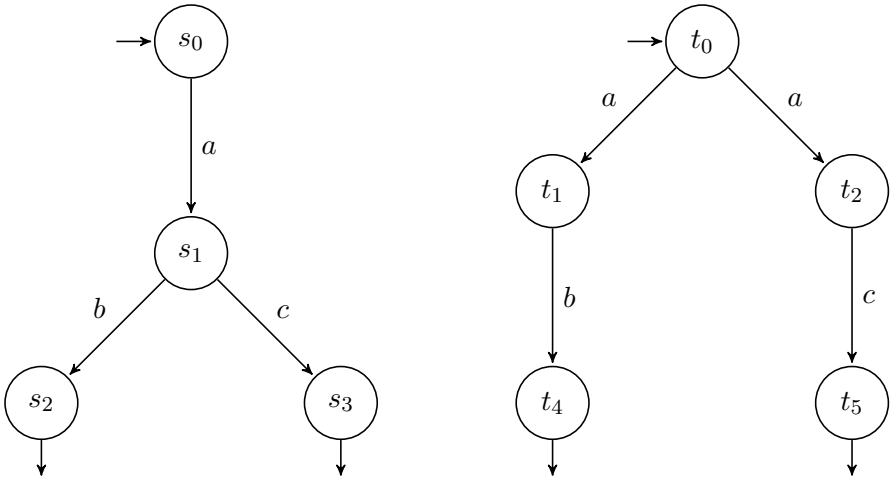

Figure 2: Two language-equivalent transition systems

Definition 2.5. Let M be a PDA which contains the states $s, t \in \mathcal{S}$, and let $a \in \mathcal{A}$, and $d, e \in \mathcal{D}$. We call a transition a *push-transition* if it of the form $s \xrightarrow{a[\bot,d]} t$ or $s \xrightarrow{a[d,ed]} t$. We call a transition a *pop-transition* if it is of the form $s \xrightarrow{a[d,\varepsilon]} t$.

2.2 Equivalences on PDA's

In this subsection we will define the behavioural equivalences used in this work. The definitions are taken from [14], unless otherwise specified. Additionally, for a more in-depth treatment of behavioural equivalences, we refer to [7].

Definition 2.6. Two PDA's M and N are *language equivalent* iff $\mathcal{L}(M) = \mathcal{L}(N)$. This is written as $M \approx N$.

Language equivalence is probably the most well-known, and most studied behavioural equivalence on automata. However, with language equivalence, a lot of information about the details of the process is lost. We will give an example of this in terms of transition systems.

Example 2.2. Consider the two transition systems given in figure 2. It should be clear that these two systems are language equivalent; they both accept the language $\{ab, ac\}$. However, their processes differ: in the left transition system one first makes an a-step, after which one chooses between b and c. In the system on the right one first chooses which a-step to take, after which one is forced to do a b or c step, depending on one's choice.

If one is interested in the actual details of the process, it is therefore a good idea to look at a different equivalence, preferably one that takes into

account the choice structure of the process. The behavioural equivalence that we will consider for most of the first part of this work is bisimulation.

Definition 2.7. Let $T_1 = (\mathcal{S}_1, \rightarrow_1, \uparrow_1, \downarrow_1)$ and $T_2 = (\mathcal{S}_2, \rightarrow_2, \uparrow_2, \downarrow_2)$ be transition systems. A *bisimulation* between T_1 and T_2 is an equivalence relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ such that for all a, $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$:

- $\uparrow_1 R \uparrow_2$.
- If $s_1 R s_2$ and $s_1 \xrightarrow{a} s_1'$ then there exists an s_2' such that $s_2 \xrightarrow{a} s_2'$.
- If $s_1 R s_2$ and $s_2 \xrightarrow{a} s_2'$then there exists an s_1' such that $s_1 \xrightarrow{a} s_1'$.
- If $s_1 \in \downarrow_1$ and $s_1 R s_2$ then $s_2 \in \downarrow_2$ and vice versa.

Transition systems T_1 and T_2 are called *bisimilar* if there exists a bisimulation between them. The notation for this is $T_1 \leftrightarroweq T_2$.

Revisiting the previous example, we can now see that while the two transition systems in figure 2 are language-equivalent, they are not bisimilar. This is becase s_0 would have to be related to t_0 by virtue of being an initial state, which would force one to relate s_1 to either t_1 or t_2. However, there is no c-transition from t_1, nor is there a b-transition from t_2, so no bisimulation can be found.

The definition of bisimulation is sometimes also called strong bisimulation. As shown via the example, for strong bisimulation each transition needs to have an equivalent transition in the other transition system, including τ transitions. This requirement is too strong in some situations. We will therefore introduce the notion of branching bisimilarity.

Definition 2.8. Let $T_1 = (\mathcal{S}_1, \rightarrow_1, \uparrow_1, \downarrow_1)$ and $T_2 = (\mathcal{S}_2, \rightarrow_2, \uparrow_2, \downarrow_2)$ be transition systems. A *branching bisimulation* between T_1 and T_2 is a relation $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ such that $\uparrow_1 R \uparrow_2$, and for all $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$, $s_1 R s_2$ implies:

- if $s_1 \xrightarrow{a} s_1'$ and $a \neq \tau$ then there exist $s_2', s_2'' \in \mathcal{S}_2$ such that $s_2 \twoheadrightarrow s_2'' \xrightarrow{a} s_2'$, and both $s_1 R s_2''$ and $s_1' R s_2'$.
- if $s_1 \xrightarrow{\tau} s_1'$ then there exist $s_2' \in \mathcal{S}_2$ such that $s_2 \twoheadrightarrow s_2'$, and $s_1' R s_2'$.
- if $s_2 \xrightarrow{a} s_2'$ and $a \neq \tau$ then there exist $s_1', s_1'' \in \mathcal{S}_1$ such that $s_1 \twoheadrightarrow s_1'' \xrightarrow{a} s_1'$, and both $s_1'' R s_2$ and $s_1' R s_2'$.
- if $s_2 \xrightarrow{\tau} s_2'$ then there exist $s_1' \in \mathcal{S}_1$ such that $s_1 \twoheadrightarrow s_1'$, and $s_1' R s_2'$.
- if $s_1 \in \downarrow_1$ then there exists a $s_2' \in \mathcal{S}_2$ such that $s_2 \twoheadrightarrow s_2'$ and both $s_1 R s_2'$ and $s_2' \in \downarrow_2$.

- if $s_2 \in\downarrow_2$ then there exists a $s_1' \in \mathcal{S}_1$ such that $s_1 \twoheadrightarrow s_1'$ and both $s_1' R s_2$ and $s_1' \in\downarrow_1$.

The transition systems T_1 and T_2 are *branching bisimilar* if there exists a branching bisimulation between them. The notation for this is $T_1 \leftrightarrows_b T_2$.

Finally, we might want our branching bisimulation to preserve divergences (that is, unbounded sequences of τ actions). This equivalence is called *branching bisimilarity with explicit divergence* in [7], but we will follow [14] in calling it divergence-preserving branching bisimilarity.

Definition 2.9. Let $T_1 = (\mathcal{S}_1, \rightarrow_1, \uparrow_1, \downarrow_1)$ and $T_2 = (\mathcal{S}_2, \rightarrow_2, \uparrow_2, \downarrow_2)$ be transition systems, and let $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ be a branching bisimulation. This relation R is a *divergence-preserving* branching bisimulation if for all $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$, $s_1 R s_2$ implies:

- if there exists an infinite sequence $(s_{1,i})_{i\in\mathbb{N}}$ such that $s_{1,0} = s_1$, $s_{1,j} \xrightarrow{\tau} s_{1,j+1}$, and $s_{1,i} R s_2$ for all $i \in \mathbb{N}$, then there exists a state s_2' such that $s_2 \rightarrow^+ s_2'$ and $s_{1,i} R s_2'$ for some $i > 0$.

- if there exists an infinite sequence $(s_{2,i})_{i\in\mathbb{N}}$ such that$s_{2,0} = s_2$, $s_{2,j} \xrightarrow{\tau} s_{2,j+2}$, and $s_1 R s_{2,i}$ for all $i \in \mathbb{N}$, then there exists a state s_1' such that $s_1 \rightarrow^+ s_1'$ and $s_1' R s_{2,i}$ for some $i > 0$.

The transition systems T_1 and T_2 are *divergence-preserving branching bisimilar* if there exists a divergence-preserving branching bisimulation between them. The notation for this is $T_1 \leftrightarrows_b^\Delta T_2$.

We note that branching bisimilarity and divergence-preserving branching bisimilarity have been proven to be equivalence relations on labelled transition systems. For the proofs of this we refer to [6] and [10], respectively.

As mentioned at the end of the previous section, every PDA is, under a "relevant equivalence", equivalent to a PDA with only push- and pop-transitions. The relevant equivalence is divergence-preserving branching bisimulation, and we will prove this statement here, as it will make later results simpler to prove. This theorem can also be found in [14].

Theorem 2.1. *For every PDA M there exists a PDA M' which uses only push- and pop-transitions, such that* $\mathcal{T}(M) \leftrightarrows_b^\Delta \mathcal{T}(M')$

Proof. We prove this by modifying any transitions in M that are not push- or pop-transitions.

- We construct M' by modifying any transition of the type $s \xrightarrow{a[\bot,\varepsilon]} t$ by adding an extra state s_0 such that $s \xrightarrow{a[\bot,d]} s_0 \xrightarrow{\tau[d,\varepsilon]} t$, where d is an arbitrary element from $\mathcal{D}$.

- We modify any transition of the type $s \xrightarrow{a[\bot,\delta]} t$ where $\delta = d_n...d_1$ and $n \geq 2$ by adding extra states $s_0, ..., s_{n-2}$ such that:

$$s \xrightarrow{a[\bot,d_1]} s_0 \xrightarrow{\tau[d_1,d_2d_1]} ... \xrightarrow{\tau[d_{n-2},d_{n-1}d_{n-2}]} s_{n-2} \xrightarrow{\tau[d_{n-1},d_nd_{n-1}]} t$$

- Any transition of the type $s \xrightarrow{a[d/\delta]} t$ where $\delta = d_0d_1..d_n \neq ed$ for some e can be modified to:

$$s \xrightarrow{a[d/\varepsilon]} s_0 \xrightarrow{\tau[\alpha/d_0]} s_1 \xrightarrow{\tau[d_0,d_1d_0]} ... \xrightarrow{\tau[d_{n-2},d_{n-1}d_{n-2}]} s_{n-2} \xrightarrow{\tau[d_{n-1},d_nd_{n-1}]} t$$

  Here α can be any single element from $\mathcal{D} \cup \{\bot\}$.

For every transition modified this way, it is clear that if we relate $s \in M$ to $s \in M'$, and $t \in M$ to $s_0, ..., t \in M'$, we will have a divergence-preserving branching bisimulation. □

2.3 Process Specifications

In the previous section we have specified processes via automata. However, it is also possible to specify a process via a process specification, which corresponds to a grammar in language theory. In this section we will define a syntax and a semantics for this purpose. This section is largely based on the works of in [14] and [2] (the former, in turn, basing his work on [3] and [4]).

Definition 2.10. Given an alphabet $\mathcal{A}$ and a finite set $\mathcal{N}$ of names, a *Recursive Specification* S is a set of equations that associates with every name in $\mathcal{N}$ a term, which is constructed as follows:

- 0 and 1 are terms.
- Any $N \in \mathcal{N}$ is a term.
- For all $a \in \mathcal{A}$ and a term t, $a.t$ is a term.
- For all terms x and y, the alternative composition $x + y$ is a term.

A recursive specification is interpreted as follows: to start with, 0 denotes a transition system with a single initial state, which is not final, and no transitions, and 1 a transition system with a single initial state, which is final, and no transitions.

Secondly, in the term $a.N$, the $a._$ part is called the *action prefix*. So, if you have $N = a.M$, this means that at N, one can perform an action a to reach M.

Finally, $x + y$ means that one can perform either x, or y. So, to put everything together, $N = a.M + b.1$ means that at N, one can either perform

a and reach M, or one can perform b and terminate. We will refer to parts of an alternative composition as *summands*. So, in this example, $a.M$ and $b.1$ are both summands of M.

We will now give the structural operating semantics for the previous definition. These semantics will allow us to reason what transitions will be possible in the labelled transition system that describes the context-free process of the recursive specification. These semantics were originally proposed by Plotkin in [13], and are also found in [2].

Definition 2.11. The *Structural Operating Semantics* (SOS) for recursive specifications is as follows. To begin with, $\downarrow$ denotes that a state is final, and $\xrightarrow{a}$ is an a-labelled transition. The following rules will define the precise semantics of these predicates.

For all terms P_1, P_1', P_2, P_2', we have the following:

$$\frac{}{1\downarrow}$$

In words: the term 1 always terminates.

$$\frac{}{a.P \xrightarrow{a} P}$$

This rule means that if a term has an action prefix, it can make a transition of that type to that term.

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 + P_2 \xrightarrow{a} P_1'}$$

$$\frac{P_2 \xrightarrow{a} P_2'}{P_1 + P_2 \xrightarrow{a} P_2'}$$

The previous two rules say that and alternative composition can make an a-transition if and only if one of the summands can make an a-transition.

$$\frac{P_1\downarrow}{(P_1 + P_2)\downarrow}$$

$$\frac{P_2\downarrow}{(P_1 + P_2)\downarrow}$$

These two rules state that an alternative composition of terms can terminate if and only if one of the summands can terminate.

Finally, we have some rules for recursion. Let N be a name in a specification, and $N = P_1$. We then have:

$$\frac{P_1 \downarrow}{N \downarrow}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{N \xrightarrow{a} P_1'}$$

These rules will be the basis for reasoning about transitions and steps of recursive specifications.

Definition 2.12. We call a recursive specification a *Linear Specification* if each term associated with a name is linear. A term is *linear* if it meets one of these two requirements:

- it is 0, 1, or of the form $a.N$, where $a \in \mathcal{A}$ and $N \in \mathcal{N}$
- it is an alternative composition of two linear terms.

Using SOS, a term of a linear specification can also be associated with a (finite) automaton.

Definition 2.13. Given a linear specification and $N \in \mathcal{N}$ a term, we will call $\mathcal{T}(N) = (\mathcal{S}, \mathcal{A}, \rightarrow, \uparrow, \downarrow)$ the finite automaton associated with this term, which we will be as follows:

- We let the set of states $\mathcal{S} \subseteq \mathcal{N}$ be those names reachable from N.
- $P \xrightarrow{a} Q$ iff $P = a.Q$ or P contains a summand $a.Q$.
- $N =\uparrow$
- $P \in\downarrow$ iff $P = 1$ or P contains a summand 1.

Note that here the choice of the N in $\mathcal{T}(N)$ determines what state will be the initial state, as linear specifications in themselves have no indication of a "starting point".

Example 2.3. Consider the following recursive specification S:

$$N = a.M + b.1M \qquad\qquad = a.M$$

Now say that we're interested in the associated labelled transition system $\mathcal{T}_S(N)$. One could, for this, easily use the construction given in the previous definition, but by way of illustration we will piece it together here using the operational semantics.

To begin with, we have:

$$b.1 \xrightarrow{b} 1$$

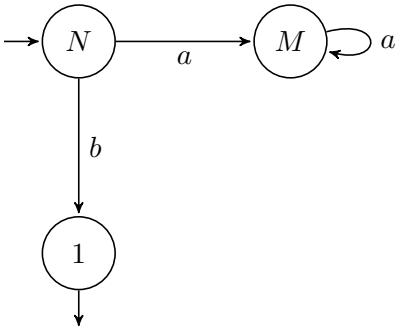

Figure 3: The labelled transition system $\mathcal{T}_S(N)$

So therefore:

$$a.M + b.1 \xrightarrow{b} 1$$

Which means that we have:

$$N \xrightarrow{b} 1$$

By the same reasoning, we therefore have:

$$N \xrightarrow{a} M$$

And finally, since $a.M \xrightarrow{a} M$, we also have:

$$M \xrightarrow{a} M$$

We finally note that 1 is the only terminating term present. So putting all this together, we get the labelled transition system seen in figure 3.

Every linear specification therefore can therefore be associated with a finite automaton. The converse of this statement, moreover, is also true: given a finite automaton, we can find a linear specification that corresponds with it up to divergence-preserving branching bisimulation. This theorem can be found in [2].

Theorem 2.2. *For every finite automaton $M = (\mathcal{S}, \mathcal{A}_\tau, \rightarrow, \uparrow, \downarrow)$ there exists a linear specification with an initial name I such that $\mathcal{T}(I) \leftrightarrows M$*

Proof. We begin by associating with each state $s \in \mathcal{S}$ a name N_s, where $N_\uparrow = I$. The terms of each name will depend on the outgoing arrows of the state. The basic principle is, that for each $s \xrightarrow{a} t$, the term for N_s will contain $a.N_t$ in its summation. Formally, we will define each name as follows:

$$N_s = \sum_{(s,a,t) \in \rightarrow} a.N_t[+1]_{s \in \downarrow}$$

The tail end of the term is a conditional addition, adding a 1 iff the state s terminates.

Using this construction, we can then easily verify that $\mathcal{T}(I)$ is stongly bisimilar to M.

To begin with, we will define the relation R as follows:

$$\forall s \in \mathcal{S}.sRN_s$$

We will now verify that R is indeed a strong bisimulation. We begin by noting that $\uparrow RN_{\uparrow}$, so the initial states are related.

Secondly, if $s \xrightarrow{a} t$ then we have $a.N_t$ as a summand of N_s, meaning that we have $N_s \xrightarrow{a} N_t$.

For the third requirement we simply note that if $N_s \xrightarrow{a} N_t$ then $a.N_t$ must be a summand of N_s, and so we must have that $s \xrightarrow{a} t$.

Lastly, if s terminates, then we have that N_s has as a summand 1, which means that N_s terminates. The same holds in the opposite direction.

We therefore conclude that R is a strong bisimulation, which concludes the proof.

□

We also note that every recursive specification

This shows that linear specifications are sufficient to describe finite automata. However, since we also wish to look at algebraic specifications of PDA's, we need to extend our algebra with another notion: sequential composition.

Definition 2.14. Given an alphabet $\mathcal{A}$ and a finite set $\mathcal{N}$ of names, a *Sequential Specification* S is a set of equations that associates with every name in $\mathcal{N}$ a term. These terms are recursively defined as follows:

- 0 and 1 are terms.
- Any $N \in \mathcal{N}$ is a term.
- For all $a \in \mathcal{A}$ and $N \in \mathcal{N}$, $a.N$ is a term.
- For all terms x and y, the alternative composition $x + y$ is a term.
- For all terms x and y, the sequential composition $x \cdot y$ is a term.

The sequential composition $x \cdot y$ means that one first performs x, and, when x successfully terminates, y is performed.

We will now extend the SOS to include sequential composition.

Definition 2.15. The *Structural Operating Semantics* for sequential specifications is the same as those for recursive specifications, with the following additions:

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \cdot P_2 \xrightarrow{a} P_1' \cdot P_2}$$

In words, if one can make a transition, this transition will still be possible if another term is sequentially composed.

$$\frac{P_1 \downarrow \qquad P_2 \xrightarrow{a} P_2'}{P_1 \cdot P_2 \xrightarrow{a} P_2'}$$

This rule states that, if, in a sequential composition, the first term can terminate, one can immediately start making transitions of the next term.

$$\frac{P_1 \downarrow \qquad P_2 \downarrow}{(P_1 \cdot P_2) \downarrow}$$

This rule states that if two terms terminate, then the sequential composition of these terms terminates as well.

We will now introduce the notion of sequential term, the primary function of which is that it makes sequential specifications easier to read.

Definition 2.16. A *Sequential Term* is a term that only contains 0, 1, action prefixes, names, and sequential composition (that is, anything but alternative composition). We say that a specification is in *Sequential Form* if each name is associated with an alternative composition of sequential terms.

Lemma 2.1. *Each sequential specification is strongly bisimilar to a sequential specification in sequential form.*

Proof. We will specify how to transform a specification that is not in sequential form. The only way that a specification could be not in sequential form is if there exists a name $n \in \mathcal{N}$ that contains a summand that is not a sequential term. For a term to be not sequential, it must contain alternative composition. This means that N must contain a summand that can be in one of the following three shapes: $\alpha \cdot (\beta + \gamma)$, $(\beta + \gamma) \cdot \delta$, or $\alpha \cdot (\beta + \gamma) \cdot \delta$, where $\alpha, \beta, \gamma, \delta$ are terms.

For the sake of brevity we will only consider the third case, as the procedure for the previous two can be easily deduced from it. The idea is that we simply introduce a new name N', such that $N' = \beta + \gamma$, and we rewrite the non-sequential term of N as $\alpha \cdot N' \cdot \delta$.

This can be done for every non-sequential term in every non-sequential name, and even if some of the new names may be non-sequential at first, since at every rewrite one reduces the depth of the terms by one, we can conclude that this is a finite process. □

It is additionally possible to rewrite any sequential specification in such a way that each summand contains at most two sequentially composed names.

Lemma 2.2. *Given a sequential specification S and a name X in S, it is possible to find a sequential specification S' such that all summands in all names in S' contain at most two sequentially composed names, and $\mathcal{T}_S(X) \leftrightarrows \mathcal{T}_{S'}(X)$.*

Proof. This is a relatively simple matter of rewriting. Let S consist of the names X_i, where $i = 0, 1, ..., n$. We then introduce new names Y_{ij}, where $Y_{ij} = X_i X_j$. One can then replace, in any sequential composition, starting from the left, a set of two names by their joint name. This process can then be repeated until all sequential compositions of names are of length 2 or less. □

Finally, we will define what we mean by a head-recursive term, as this will be relevant in later sections, and it is a central notion in one of the new results presented in this paper.

Definition 2.17. We call a term N *Head-Recursive* if its alternative composition contains a term of the form $N \cdot X$, where X is a non-empty sequential composition of names.

2.4 Greibach Normal Form for Sequential specifications

In this section we will detail the Greibach Normal Form (GNF) for sequential specifications, which was introduced in [11]. This specific form for sequential specifications will be used for many results, not least of which the language equivalence between PDA's and sequential specifications in GNF, to be detailed in the next section.

We begin by defining when a term is considered guarded as a basic notion that will be extended to pre-GNF, which will finally be developed into GNF proper.

Definition 2.18. A term in a sequential specification S is *guarded* if it is of the form $a.X$, where $a \in \mathcal{A}_\tau$, and where X is a sequential composition of one or more names, or 1.

Note that this means that, for instance, terms of the form $a.0$ are not considered guarded.

Definition 2.19. A sequential specification S is in *pre-Greibach Normal Form* (pre-GNF) if every term associated with a name N meets one of the following requirements:

- $N = 0$.

- It is an alternative composition of one or more terms. Each of these terms is of one of the following forms:
  - The term is guarded.
  - The term is 1.
  - The term is of the form $N \cdot X$, where N is head-recursive.

Theorem 2.3. *Given a sequential specification S, there exists a sequential specification S' such that S' is in pre-GNF, and $S \leftrightarrows S'$.*

Proof. We will detail how to obtain S' by rewriting S in bisimilar ways. To begin with, one rewrites S to be in sequential form (see previous section). Any single variables can then be replaced by the term associated with their names, or, if they are of the shape $N = N + x$, then the single variable N in N can be removed entirely (since $N \leftrightarrows x$).

If any alternative composition contains the term $N \cdot X$, where X is non-empty, then this can be rewritten by replacing N with its associated term in S. If one encounters a name N that has $N \cdot X$ as a summand, then this summand is head-recursive and can be left as is. □

Pre-GNF is designed to preserve bisimilarity. If one is merely concerned with language equivalence, then one can use GNF proper:

Definition 2.20. A sequential specification S is in *Greibach Normal Form* (GNF) if every term associated with a name meets one of the following requirements:

- The term is 0
- It is of the form $\sum a.X(+1)$, where $a \in \mathcal{A} \vee a = \tau$, X is a sequential composition, and $(+1)$ means that it has an optional 1-summand.

Theorem 2.4. *For every sequential specification S there exists a sequential specification S' such that S' is in GNF and $S \approx S'$*

Proof. Firstly, we rewrite S into S'', where S'' is in pre-GNF, and $S \leftrightarrows_b^\Delta S''$ (and therefore $S \approx S''$). We can then simply rewrite any head-recursive terms $N \cdot X$ into $\tau.N \cdot X$, which will give S' in such a way that $S'' \approx S'$, and so by transitivity of language equivalence, we get $S \approx S'$. □

Now, if we look at transition systems associated with sequential specifications in GNF, we see a desirable property emerge, namely, that no infinite or unbounded branching occurs. Additionally, it will contain only one final state.

Theorem 2.5. *If S is a sequential specification in GNF, and no names in S contain a 1-summand, then $\mathcal{T}_S(I)$, where I is a name occurring in S, will have finite, bounded branching, and at most one final state.*

Proof. Firstly, the states of the transition system will be labelled by the names in S, all sequential compositions of names reachable from S, and finally of a state labelled 1. The state labelled 1 will be the only terminating state, as no terms associated with names have immediate termination. Additionally, as may be clear, the state labelled I will be the initial state.

Now, for every state X, where X is a non-empty composition of names, all the outgoing transitions will be dependent on the first name in the sequential composition of X. For example, say that $X = N \cdot Y$ and N contains the summand $a.Z$, then in the transition system one will have $X \xrightarrow{a} Z \cdot Y$. This means that every state will have at most as many outgoing arrows as there are summands in the first name of the sequential composition. Since this is by definition finite, this means that all branching in $\mathcal{T}_S(I)$ will be finite, and therefore bounded. □

We will now also define what it means to be a sequential language, and what it means to be a sequential process.

Definition 2.21. Given a sequential specification S and an initial name I, we will denote the language associated with this specification as $\mathcal{L}_S(I)$.

Definition 2.22. Given a sequential specification S and an initial name I, the *Sequential Process* $\mathcal{P}_S(I)$ is the divergence-preserving branching bisimilarity class of transition systems that contains the transition system $\mathcal{T}_S(I)$.

Theorem 2.6. *The class of sequential processes is closed under $+$ and $\cdot$.*

Proof. Let $\mathcal{P}_S(I)$ and $\mathcal{P}_T(J)$ be sequential processes, and let $S \cap T = \emptyset$.

- $\mathcal{P}_S(I) + \mathcal{P}_T(J) = \mathcal{P}_{S \cup T}(I + J)$.
- $\mathcal{P}_S(I) \cdot \mathcal{P}_T(J) = \mathcal{P}_{S \cup T}(I \cdot J)$.

□

Theorem 2.7. *The class of sequential languages is closed under union and concatenation.*

Proof. Let $\mathcal{L}_S(I)$ and $\mathcal{L}_T(J)$ be sequential languages, and let $S \cap T = \emptyset$. The proof is then identical to the proof above, except regarding language rather than processes. □

Theorem 2.8. *Sequential languages (and therefore by extension sequential processes) are not closed under intersection and taking the complement.*

Proof. This is proven by counterexample. Consider the following two sequential specifications:

$$S = \{I = N \cdot M, N = 1 + a.N \cdot b.1, M = 1 + c.M\}$$

$$T = \{J = K \cdot L, K = 1 + a.K, L = 1 + b.L \cdot c.1\}$$

Given these specifications, we know that $\mathcal{L}_S(I) = \{a^n b^n c^m | n, m \geq 0\}$ and $\mathcal{L}_T(J) = \{a^n b^m c^m | n, m \geq 0\}$. However, this means that $\mathcal{L}_S(I) \cap \mathcal{L}_T(J) = \{a^n b^n c^n | n \geq 0\}$, which is well known to be non-sequential [1].

This result also immediately gives us the fact that sequential languages are not closed under taking complement. Consider the following set-theoretic fact:

$$(U^C \cup V^C)^C = U \cap V$$

Therefore, if sequential languages were closed under taking complement, they would be closed under intersection as well, which is a contradiction. □

2.5 PDA's & Sequential Algebras

This section will contain a number of results pertaining to the link between push-down automata and sequential specifications. The following proof is based mostly on [14], and has been slightly modified to suit the purpose of this work.

Theorem 2.9. *Given a sequential specification E in Greibach normal form, where E contains no transparent names, and an expression P_x from this specification, there exists a Push-Down Automaton (PDA) M (under the FSES interpretation) consisting of one state such that $\mathcal{T}_E(P_x) \leftrightarrows_b^\Delta \mathcal{T}(M)$, on the condition that M may start with a non-empty stack.*

Corollary 2.1. *Given a sequential specification E in Greibach normal form, and an name P_x from this specification, there exists a Push-Down Automaton (PDA) M (under the FSES interpretation) consisting of two states such that $\mathcal{T}_E(P_x) \leftrightarrows_b^\Delta \mathcal{T}(M)$.*

Proof. Let $E = \{P_i = \sum_{j \in J_i} a_{ij}.\xi_{ij} | i \in I\}$ be a sequential specification in Greibach normal form (GNF), and $x \in I$. Here ξ_j is a finite list of sequentially composed names from E.

The claim is now that this sequential specification can be simulated by a one-state PDA. This PDA is constructed as follows:

- One state, N, where $N \uparrow$ and $N \downarrow$.
- For all $i \in I$, let $N \xrightarrow{a_j[P_i/\xi_j]} N$ for all $j \in J$.
- Let the initial stack consist of the symbol P_x.

The claim is that $\mathcal{T}_E(P_x) \leftrightarrows_b^\Delta \mathcal{T}(M)$. Firstly, for $\mathcal{T}_E(P_x)$ the construction detailed previously is used. Note in particular that this means that for a state $P_i\chi_k$, (where χ_k is either empty or a list of names from E) it holds that $P_i\chi_k \xrightarrow{a_j} \xi_j\chi_k$ iff P_i contains $a_j\xi_j$ as part of its alternate composition.

Secondly, for $\mathcal{T}(M)$ the standard construction given earlier is used. This means that in the transition system, for any state $(N, s\zeta)$ (where $s\zeta$ is a stack with the element s on top) we have $(N, s\zeta) \xrightarrow{a_t} (N, \rho\zeta)$ iff in M we have $N \xrightarrow{a_t[s/\rho]} N$.

Now, to prove that $\mathcal{T}_E(P_x) \leftrightarrows_b^\Delta \mathcal{T}(M)$, let $R \subseteq S \times T$, where S is the set of states from $\mathcal{T}_E(P_x)$ and T is the set of states from $\mathcal{T}(M)$. We define R as follows: simply let $\alpha R(N, \alpha)$, where α is a list of names from E.

Now all that is left is to show that this is a branching bisimulation that preserves divergencies. First, we show that it is a bisimulation. For this, let $\alpha_i = P_y\beta_i$ and $\alpha_i R(N, \alpha_i)$.

Now, for the first condition of bisimulation, if $\alpha_i \xrightarrow{a_z} \alpha_j$, this means that $\alpha_j = \xi_z\beta_i$, and that, in E, the alternate composition of P_y contains $a_z.\xi_z$. This then implies that in the PDA M we have $N \xrightarrow{a_z[P_y/\xi_z]} N$, which implies that in $\mathcal{T}(M)$ we have $(N, P_y\beta_i) \xrightarrow{a_z} (N, \xi_z\beta_i)$, which proves the condition. Since the reverse of all these implications also holds, this also proves the second part of the definition. Furthermore, we note that this reasoning also holds for the terminating states, proving that R is in fact a bisimulation. Not only does it prove this, but it also shows that $\mathcal{T}_E(P_x)$ and $\mathcal{T}(M)$ are in fact isomorphic.

For the corollary one only needs to add one extra state Q to M, which will be the initial state instead of N, and the transition $Q \xrightarrow{\tau[\bot/P_x]} N$. It is then easy to show that the previous result still holds, although only up to divergence preserving branching bisimulation. □

Corollary 2.2. *Under the FS interpretation of PDAs, one additional state is needed.*

Proof. Under the FS interpretation one will need to add an additional state N', which will be the only terminating state (meaning that N will no longer terminate), and the transition $N \xrightarrow{\tau[\bot/\varepsilon]} N'$. It can easily be checked that this PDA is still branching bisimilar with divergencies to E. □

This result can also be used to get another classic result pretty much immediately:

Theorem 2.10. *Given a sequential specification S and an initial name I, there exists a PDA M such that $\mathcal{L}_S(I) \approx \mathcal{L}(M)$.*

Proof. Every sequential specification can be transformed to GNF under language equivalence, after which the only thing left to do is construct a fitting PDA as detailed above. □

One can also prove the converse (adapted from [12]).

Theorem 2.11. *Given a PDA M, there exists a sequential specification S with an initial state I such that S is in GNF, and $\mathcal{L}(M) \approx \mathcal{L}_S(I)$.*

Proof. Firstly, under language equivalence we can safely assume the following things about M:

- M has only one final state.
- M has only push/pop transitions.

The sequential specification S will now be constructed as follows: firstly, one creates names $V_{s\emptyset}$ and V_{sdt} for all $s, t \in \mathcal{S}$ and $d \in \mathcal{D}$. These names then get assigned summands as follows:

- We assign to $V_{s\emptyset}$ the summand $a.V_{tdu} \cdot V_{u\emptyset}$ for every $u \in \mathcal{S}$ and every step $s \xrightarrow{a[\emptyset/d]} t$.
- We assign to V_{sdt} the summand $a.1$ for every step $s \xrightarrow{a[d/\varepsilon]} t$.
- We assign to V_{sdt} the summand $a.V_{uev} \cdot V_{vdt}$ for every $t, v \in \mathcal{S}$ and every step $s \xrightarrow{a[d/ed]} u$.
- We assign to $V_{\downarrow\emptyset}$ the summand 1.

Any names without summands will be assigned the constant 0.

The basic idea of this construction is as follows: basically, the names will encode both the contents of the stack, and, so to say, a potential order in which the content of the stack will be "spent". So, say that we are in $V_{s\emptyset}$, and there is a step $s \xrightarrow{a[\emptyset/d]} t$. This means that the alternate composition of $V_{s\emptyset}$ will most likely contain many summands of the form $a.V_{tdu} \cdot V_{u\emptyset}$. Note that for each V_{tdu} there are two options: either the d in the stack is "spent" in the step to u, in which case it can terminate by choosing the summand $a1$, after which the process will be at the name $V_{u\emptyset}$, or another element is gained, in which case the list of sequentially composed names increases (and so do the potential options).

It is therefore clear that if the PDA M accepts a certain word, then the sequential specification S will too. However, one could still worry that, since so many potential (and potentially impossible) paths are created, perhaps S will accept more than M. However, it is not hard to see that this will never happen: say that one chooses a summand with an impossible path. Firstly, we can assume that the final name in the sequential composition is $V_{\downarrow\emptyset}$, as otherwise the state will not terminate. Secondly, we can assume that all names of the shape V_{sdt} will no longer choose any summands of the form $a.V_{uev} \cdot V_{vdt}$, as otherwise the path might be possible, rather than impossible.

Given these assumptions, it must then mean that there is a V_{sdt} such that no step of the form $s \xrightarrow{a[\emptyset/d]} t$ exists. However, this means that $V_s dt = 0$, and therefore causes a deadlock. $\square$

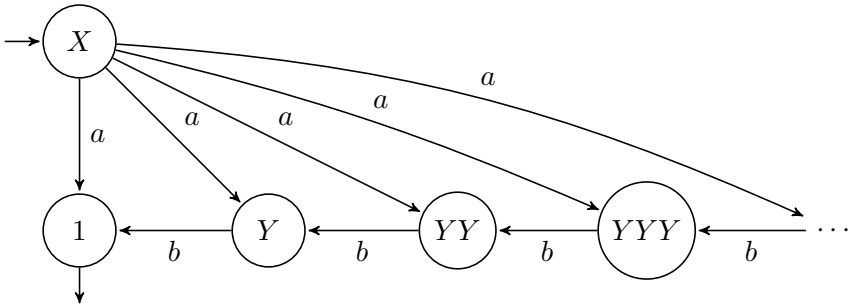

Figure 4: Transition system $\mathcal{T}_S(X)$

3 Head-Recursion in Process Algebras

We have seen that, given a sequential specification in GNF, one can find a PDA such that the respective transition systems are divergence-preserving branching bisimilar. In this section we will look at sequential specifications containing head-recursion.

Example 3.1. Consider the following sequential specification S:

$$\begin{aligned} X &= X \cdot Y + a.1 \\ Y &= b.1 \end{aligned}$$

If we expand the equation for X we see that the following transitions will be possible:

$$\begin{aligned} X &\xrightarrow{a} 1 \\ X \cdot Y &\xrightarrow{a} Y \\ X &\xrightarrow{a} Y \\ X \cdot Y &\xrightarrow{a} Y \cdot Y \\ X &\xrightarrow{a} Y \cdot Y \\ X \cdot Y &\xrightarrow{a} Y \cdot Y \cdot Y \\ X &\xrightarrow{a} Y \cdot Y \cdot Y \end{aligned}$$

And so on. It is clear that one can therefore, from X, make an a-step to an infinite number of states. The transition-system $\mathcal{T}_S(X)$ is therefore like in figure 4.

For this, under divergence-preserving branching bisimilarity, a matching PDA can not be found. However, if we drop the divergence-preserving aspect

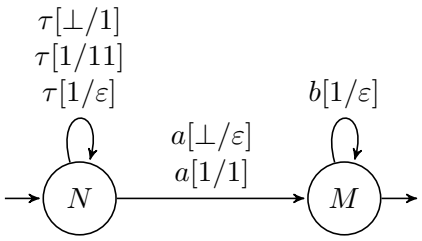

Figure 5: Push-down automaton M

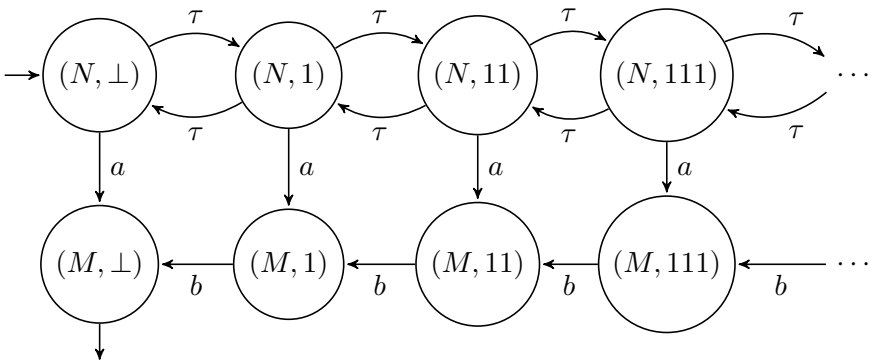

Figure 6: Transition system $\mathcal{T}(M)$

from the branching bisimilarity, it is possible. Consider the PDA shown in figure 5, and the associated transition system seen in figure 6.

We will now show that $\mathcal{T}_S(X)$ and $\mathcal{T}(M)$ are branching bisimilar. Consider the following relation R:

$$R = \{(X, (N, 1^n)) | n = 0, 1, ...\} \cup \{(Y^m, (M, 1^m)) | m = 0, 1, ...\}$$

One can easily verify that R is a branching bisimulation. The main point of the proof is that, since all $(N, 1^n)$ are related to X, the newly introduced τ-transitions simply correspond to standing still in $\mathcal{T}_S(X)$. Additionally, since one can move along the string of $(N, 1^n)$ states freely, any step from X to Y^m can be simulated by $(N, 1^n) \twoheadrightarrow (N, 1^m) \xrightarrow{a} (M, 1^m)$.

The previous example illustrates the basic idea of the more general case: to deal with head-recursion, one "unravels" the state in which the head-recursion occurs to form a string of states where one can get to any state using only τ-transitions (introducing divergencies where they previously didn't exist, meaning that it only works up to branching bisimilarity).

However, when one tries to generalize this idea, one runs into additional problems, illustrated by the next example:

Example 3.2. Consider the following sequential specification S:

$$X = X \cdot Y + a.1 + c.Z$$
$$Y = b.1$$
$$Z = d.X$$

When figuring out the transition system $\mathcal{T}_S(X)$, we begin by noting that the steps derived in example 3.1 are still possible. That is, there still exists the transition $X \xrightarrow{a} Y^n$ for any $n \in \mathbb{N}$. However, when expanding the equations, an additional pattern starts to emerge:

$$X \xrightarrow{c} Z$$
$$X \cdot Y \xrightarrow{c} Z \cdot Y$$
$$X \xrightarrow{c} Z \cdot Y$$
$$X \cdot Y \xrightarrow{c} Z \cdot Y \cdot Y$$
$$X \xrightarrow{c} Z \cdot Y \cdot Y$$
$$X \cdot Y \xrightarrow{c} Z \cdot Y \cdot Y \cdot Y$$

So the transition $X \xrightarrow{c} ZY$ is possible, from where one is forced into the transition $ZY \xrightarrow{d} XY$. So, contrary to Example 3.1, here XY (and XYY, and $XYYY$, etc.) are proper states. These states are clearly not identical to X, as one cannot transition from XYY straight to Y, for instance. The transition system $\mathcal{T}_S(X)$ can be seen in Figure 7 (for the sake of readability the transitions have been left unlabelled, but the action associated with the transition should in all cases be clear).

Keeping this difficulty in mind, we will now treat the more general case, where only the initial state is head-recursive.

Theorem 3.1. *Let S be a sequential specification defined by a set of variables $\{X, Y_i | i = 0, 1, ..., n\}$ as follows:*

$$X = X \cdot Y_0 + \sum_{k \in I_X} a_k.\xi_k$$
$$Y_i = \sum_{j \in I_i} a_{ij}.\xi_{ij}$$

*Here ξ is a sequential composition of any number of variables, where zero variables is understood to be equal to **1**.*

There exists a PDA P such that $\mathcal{T}_S(X) \leftrightarrows_\Delta \mathcal{T}(P)$.

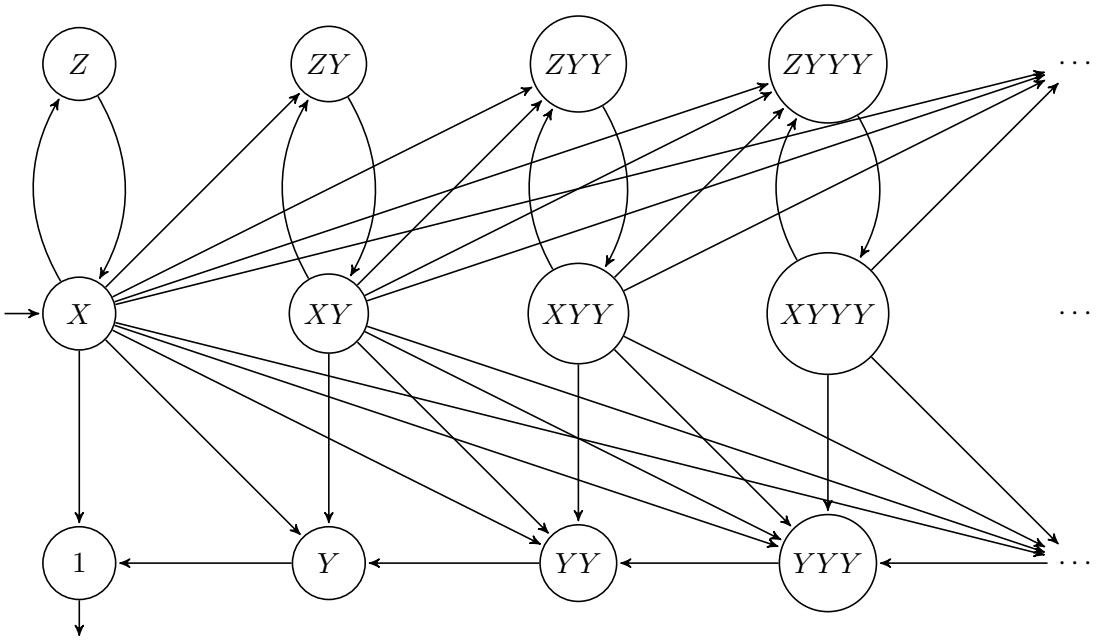

Figure 7: Transition system of a more complex head recursion

Proof. Keeping the difficulties from the previous example in mind, we introduce the segment symbol $\diamond$, with which we will mark any return to the head-recursive state X, so that we don't accidentally throw away any previously committed Y_0's from the stack. The set of data $\mathcal{D}$ used by this PDA will therefore be defined as $\{X, Y_i | i = 0, 1, ..., n\} \cup \{\diamond\}$. The PDA P will consist of three states, $\{1, N, M\}$. The state 1 will simply behave as the terminating state. The state N will take care of the head-recursion of X, and will therefore behave as follows:

$$N \xrightarrow{\tau[\bot/Y_0]} N$$
$$N \xrightarrow{\tau[Y_0/Y_0Y_0]} N$$
$$N \xrightarrow{\tau[Y_0/\varepsilon]} N$$
$$N \xrightarrow{\tau[\diamond/Y_0\diamond]} N$$
$$\forall k \in I_X, d \in \mathcal{D}. N \xrightarrow{a_k[d/\xi_k d]} M$$

The first four steps take care of the head-recursion: it allows the PDA to stack and remove as many Y_0's as is needed, though any Y_0's on the stack below the segment symbol $\diamond$ will not be removed. Additionally, one can

terminate at this state, and the final scheme for steps indicate that at any point N can choose any other summand.

The second state, M, will simulate the behaviour of all other variables:

$$\begin{aligned} \forall i \in \{0,1,...,n\}, j \in I_i . M &\xrightarrow{a_{ij}[Y_i/\xi_{ij}]} M \\ M &\xrightarrow{\tau[\bot/\varepsilon]} 1 \\ M &\xrightarrow{\tau[\diamond/\varepsilon]} M \\ M &\xrightarrow{\tau[X/\diamond]} N \end{aligned}$$

The first two lines specify the simulation of the Y_i variables, functioning in the same vein as before. The latter two lines specify the functioning of the segment symbol.

This concludes the specification of the PDA P. All that remains is to show that $\mathcal{T}_S(X) \leftrightarrows_\Delta \mathcal{T}(P)$. For that we will first introduce some new notation, to enhance readability.

Firstly, we will denote with ζ a string of variables from S, possibly also containing the segment symbol $\diamond$. To put it more formally, $\zeta \in (S \cup \{\diamond\})^*$ With ζ^- we will mean the string ζ, but with any segment symbols removed. With $\zeta^\diamond$ is meant any element from the set of strings that one can obtain by placing segment symbols anywhere in the string ζ. Finally, with the notation $[\diamond]\zeta$ we mean the string $\diamond\zeta$ if ζ is not empty, and the empty string otherwise.

We will now give the relation $R \subseteq \mathcal{T}_S(X) \times \mathcal{T}(P)$ that will be our branching bisimulation:

$$\begin{array}{ll} \forall \zeta \in S^*.\zeta & R(M, \zeta^\diamond) \\ \forall m \in \mathbb{N}.X & R(N, Y_0^m) \\ \forall \zeta \in S^*.X\zeta & R(N, Y_0^m \diamond \zeta^\diamond)) \\ 1 & R1 \end{array}$$

Where $m \in \mathbb{N}$.

To prove that R is a branching bisimulation, we first check that $\uparrow_1 R \uparrow_2$. Since $\uparrow_1 = X$, and $\uparrow_2 = (N, \bot)$, and $XR(N, \bot)$, this first condition is satisfied.

For the rest of the conditions, assume that $s_1 R s_2$.

- If $s_1 \xrightarrow{a} s_1'$ and $a \neq \tau$, we claim that there exist $s_2', s_2'' \in \mathcal{T}(P)$ such that $s_2 \twoheadrightarrow s_2'' \xrightarrow{a} s_2'$ and $s_1' R s_2'$.

  First, assume that $s_1 = X\zeta$, with ζ potentially empty. Then $s_2 = (N, Y^m[\diamond]\zeta)$, or $s_2 = (M, X\zeta^\diamond)$. However, the latter reduces to the former case, as $(M, X\zeta^\diamond) \xrightarrow{\tau} (N, \diamond\zeta^\diamond)$. There is then one option for

non-τ steps from s_1, namely that $s_1' = \xi_k Y_0^{m'}\zeta$, so the step taken is $X\zeta \xrightarrow{a_k} \xi_k Y_0^{m'}\zeta$, or, in words, one goes down the head-recursion m' times, and then chooses another, non-head-recursive summand from X. Then the desired states in $\mathcal{T}(P)$ are $s_2'' = (N, Y_0^{m'}[\diamond]\zeta)$ and $s_2' = (M, \xi_k Y_0^{m'}[\diamond]\zeta)$. This is because

$$(N, Y_0^{m}[\diamond]\zeta) \twoheadrightarrow (N, Y_0^{m'}[\diamond]\zeta)$$

since one can freely change the number of Y_0's on the stack in the state N, and since $\forall k \in I_X, d \in \mathcal{D}.N \xrightarrow{a_k[d/\xi_k d]} M$, we have:

$$(N, Y_0^{m'}[\diamond]\zeta) \xrightarrow{a_k} (M, \xi_k Y_0^{m'}[\diamond]\zeta)$$

Additionally, it is easy to see that these states also relate in the correct way:

$$X\zeta R (N, Y_0^{m'}[\diamond]\zeta)$$
$$\xi_k Y_0^{m'}\zeta R (M, \xi_k Y_0^{m'}[\diamond]\zeta)$$

meaning that the requirements for branching bisimilarity are, in this case, satisfied.

So if $s_1 = X\zeta$, this requirement for bisimilarity holds. We will now consider the simpler case where $s_1 = Y_i\zeta$. This means that $s_2 = (M, Y_i\zeta^{\diamond})$. The only possible transition in this scenario is a transition where $s_1' = \xi_{ij}\zeta$ and so the transition made is $Y_i\zeta \xrightarrow{a_{ij}} \xi_{ij}\zeta$. However, since this means that Y_i contains the summand $a_{ij}.\xi_{ij}$, we also have that $M \xrightarrow{a_{ij}[Y_i/\xi_{ij}]} M$, and so we have:

$$(M, Y_i\zeta^{\diamond}) \xrightarrow{a_{ij}} (M, \xi_{ij}\zeta^{\diamond})$$

And additionally we know that:

$$\xi_{ij}\zeta R (M, \xi_{ij}\zeta^{\diamond})$$

Therefore, for every non-τ transition from s_1 we can find the appropriate s_2'', s_2' to satisfy branching bisimilarity.

- If $s_1 \xrightarrow{\tau} s_1'$, then there exists a s_2' such that $s_2 \twoheadrightarrow s_2'$ and $s_1' R s_2'$. If there is a transition from s_1 of the shape $s_1 \xrightarrow{\tau} s_1'$, then this means that the first variable in the name of s_1, say Z_i, must have contained a summand of the shape $\tau.\xi_{ij}$, which means that the proof for the previous point applies here as well.

- If $s_2 \xrightarrow{a} s_2'$ and $a \neq \tau$, then there exist $s_1', s_1'' \in \mathcal{T}_S(X)$ such that $s_1 \twoheadrightarrow s_1'' \xrightarrow{a} s_1'$.

  First, we look at the case where $s_2 = (N, Y_0^m[\diamond]\zeta)$. This means that s_1 must be the state $X\zeta^-$. There is now only one non-τ transition that can be made from s_2, and that is the case in which $s_2' = (M, \xi_k Y^m[\diamond]\zeta)$, so the transition made is:

  $$(N, Y_0^m[\diamond]\zeta) \xrightarrow{a_k} (M, \xi_k Y^m[\diamond]\zeta)$$

  Here we have that $s_1'' = s_1$ and $s_1' = \xi_k Y^m \zeta^-$. Since the transition in $\mathcal{T}(M)$ is possible because X contains the summand $a_k.\xi_k$, we can similarly in $\mathcal{T}_S(X)$ make the transition:

  $$X\zeta^- \xrightarrow{a_k} \xi_k Y^m \zeta^-$$

  And we know that:

  $$\xi_k Y^m \zeta^- R (M, \xi_k Y^m[\diamond]\zeta)$$

  So if $s_2 = (N, Y_0^m[\diamond]\zeta)$, this requirement of bisimilarity holds.

  The only other option is that $s_2 = (M, Y_i\zeta)$. Note that we do not consider the state $(M, X\zeta)$ and $(M, \bot)$ here: this is because no non-τ steps can be taken from these states.

  If $s_2 = (M, Y_i\zeta)$, we know that $s_1 = Y_i\zeta^-$. All non-τ transitions that can be taken from s_2 are of the following shape:

  $$(M, Y_i\zeta) \xrightarrow{a_{ij}} (M, \xi_{ij}\zeta)$$

  Again, if this transition is possible, it must be that in M we have the transition $M \xrightarrow{a_{ij}[Y_i/\xi_{ij}]}$. If this step exists in M, then we know that Y_i must have $a_{ij}.\xi_{ij}$ as a summand. Which in turn means that in $\mathcal{T}_S(X)$ we can make the following transition:

  $$Y_i\zeta^- \xrightarrow{a_{ij}} \xi_{ij}\zeta^-$$

  Which is, of course, the desired transition, since:

  $$\xi_{ij}\zeta^- R (M, \xi_{ij}\zeta)$$

  Which means that, in every scenario, if $s_2 \xrightarrow{a} s_2'$, there exist appropriate s_1', s_1'' for bisimilarity to hold.

- If $s_2 \xrightarrow{\tau} s_2'$, then there exists a $s_1' \in \mathcal{T}_S(X)$ such that $s_1 \twoheadrightarrow s_1'$ and $s_1' R s_2'$.

  This direction of the requirement on τ steps requires a bit more work than its converse, since $\mathcal{T}(M)$ contains many more silent transitions than its process-algebraic counterpart.

  First, we consider the case in which $s_2 = (N, Y_0^m[\diamond]\zeta)$. This means that $s_1 = X\zeta^-$. In this case, only one type of silent transition can be made: M can either add, or remove, a Y_0 from the stack. So:

  $$(N, Y_0^m[\diamond]\zeta) \xrightarrow{\tau} (N, Y_0^{m\pm1}[\diamond]\zeta)$$

  In this case, one simply takes s_1' to be equal to s_1, since $s_1 \twoheadrightarrow s_1'$ is not only the transitive, but also reflexive closure of τ, and additionally we have

  $$X\zeta^- R (N, Y_0^{m\pm1}[\diamond]\zeta)$$

  Meaning that for this case, the requirements hold.

  The second scenario we will consider is $s_2 = (M, \perp)$. Here it must be the case that $s_1 = 1$. The only possible transition from s_2 is $(M, \perp) \xrightarrow{\tau} 1$, which poses no problems since $1R1$.

  The third scenario to consider is when $s_2 = (M, \diamond\zeta)$. Here we must have $s_1 = (\diamond\zeta)^- = \zeta^-$. The only possibility for s_2' here is $s_2' = (M, \zeta)$, as no other τ-transitions exist from this state but the following one:

  $$(M, \diamond\zeta) \xrightarrow{\tau} (M, \zeta)$$

  However, again, this is no problem, as we can simply choose $s_1' = s_1$ once more, as $\zeta^- R (M, \zeta)$.

  Finally, the fourth case to consider is when $s_2 = (M, X\zeta)$. Here we have that $s_1 = X\zeta^-$, and the only possible option for s_2' is $s_2' = (N, \diamond\zeta)$, so the transition is:

  $$(M, X\zeta) \xrightarrow{\tau} (N, \diamond\zeta)$$

  Which, like before, is no issue if one takes $s_1' = s_1$, as $X\zeta^- R (N, \diamond\zeta)$.

  As a final note, any τ-transitions not covered by these cases are transitions based on summands in S, which means that the proof in the previous bullet-point applies.

- If $s_1 \in\downarrow_1$, then $s_1 = 1$, and so it must be the case that $s_2 = 1$, and therefore $s_2 \in\downarrow_2$, as $1R1$. The converse also holds, as both transition systems have only one terminating state (namely, 1).

This concludes the proof of R being a branching bisimilarity.

□

With this, we can make a further generalization to cover all non-transparent, head-recursive sequential specifications.

Theorem 3.2. *Let S be a sequential specification defined by a set of variables $\{X_i, Y_{i'} | i = 0, 1, ...n \wedge i' = 0, 1, ..., n'\}$ as follows:*

$$X_i = \sum_{j \in I_i} X_i \cdot V_{ij} + \sum_{k \in J_i} a_{ik}.\xi_{ik}$$
$$Y_{i'} = \sum_{j' \in I_{i'}} a_{i'j'}.\xi_{i'j'}$$

Let Z and each V_{ij} be a name from S. There exists a PDA P such that $\mathcal{T}_S(Z) \leftrightarrows_\Delta \mathcal{T}(P)$.

Proof. The PDA P will follow the same principles as the PDA seen in the previous proof. The two main differences, in words, will be the following: to begin, each X_i, that is, each head-recursive name, will get its state, N_i, which can add or remove any V_{ij} at will. The second difference is that each X_i is now allowed to contain any finite number of head-recursive summands. This does not change anything fundamentally: it simply means that one can jump to any sequence of V_{ij}'s from X.

The basic part of the PDA will therefore look as follows:

$$\begin{aligned}
\forall i = 0, 1, ..., n; j \in I_i . N_i &\xrightarrow{\tau[\bot/V_{ij}]} N_i \\
N_i &\xrightarrow{\tau[V_{ij}/V_{ij}V_{ij}]} N_i \\
N_i &\xrightarrow{\tau[V_{ij}/\varepsilon]} N_i 0 \\
N_i &\xrightarrow{\tau[\diamond/V_{ij}\diamond]} N_i \\
\forall k \in J_i, d \in \mathcal{D} . N_i &\xrightarrow{a_{ik}[d/\xi_{ik}d]} M \\
\forall i' \in \{0, 1, ..., n'\}, j' \in I_{i'} . M &\xrightarrow{a_{i'j'}[Y_{i'}/\xi_{i'j'}]} M \\
M &\xrightarrow{\tau[\bot/\varepsilon]} 1 \\
M &\xrightarrow{\tau[\diamond/\varepsilon]} M \\
M &\xrightarrow{\tau[X_i/\diamond]} N_i
\end{aligned}$$

However, since we have also lifted the demand to have a specific name be the initial name, we will introduce one extra state, M':

$$M' \xrightarrow{\tau[\bot/Z]} M$$

The relation $R \subseteq \mathcal{T}_S(Z) \times \mathcal{T}(P)$ that will be our branching bisimulation will also not hold any particular surprises:

$$\begin{gathered} \zeta R(M, \zeta^\diamond) \\ Z R(M', \bot) \\ X_i \zeta_1 R(N_i, \zeta_2^\diamond \diamond \zeta_1^\diamond) \\ X_i \zeta_1 R(N_i', \zeta_2^\diamond \diamond \zeta_1^\diamond) \\ 1 R 1 \end{gathered}$$

As for the proof, all the work is essentially done in the preceding proof: for each moment of head-recursion in S, the state M sends one to the appropriate state N_i, where the desired stack can be freely generated, before being sent back to M to proceed. The only thing left to check is the transition from the newly made state M':

$$(M', \bot) \xrightarrow{\tau[\bot/Z]} (M, Z)$$

However, since both $Z R(M', \bot)$ and $Z R(M, Z)$, it will still hold that R is a branching bisimulation.

□

4 Transparency in Process Algebras

In this section we will look at sequential specifications that contain transparent names, and show that these specifications can be emulated by push-down automata up to contrasimulation.

We will begin by defining contrasimulation. Recall that with $\twoheadrightarrow$ we mean the transitive, reflexive closure of τ-transitions. We also introduce the following new notation: with $s \stackrel{a}{\twoheadrightarrow} t$ we mean $s \twoheadrightarrow u \stackrel{a}{\rightarrow} v \twoheadrightarrow t$.

Definition 4.1. Let $\mathcal{T}_1, \mathcal{T}_2$ be labelled transition systems. A pair of relations (R, Q), where $R \subseteq \mathcal{T}_1 \times \mathcal{T}_2$ and $Q \subseteq \mathcal{T}_2 \times \mathcal{T}_1$ is a *contrasimulation* if it meets the following requirements.

First, we have that $\uparrow_1 R \uparrow_2$ and $\uparrow_2 Q \uparrow_1$.

Let $s_1 R s_2$. Then:

- If $s_1 \stackrel{a}{\twoheadrightarrow} s_1'$ and $a \neq \tau$, there exist $s_2' \in \mathcal{T}_2$ such that $s_2 \stackrel{a}{\twoheadrightarrow} s_2'$, and $s_2' Q s_1'$.
- If $s_1 \stackrel{\tau}{\twoheadrightarrow} s_1'$, then there exists a $s_2' \in \mathcal{T}_2$ such that $s_2 \twoheadrightarrow s_2'$ and $s_2' Q s_1'$.
- If $s_1 \in\downarrow_1$, then there exists a $s_2' \in \mathcal{T}_2$ such that $s_2 \twoheadrightarrow s_2'$, $s_2' \in\downarrow_2$ and $s_2' Q s_1$.

Secondly, we require the converse of the above requirements to hold:

Let $s_2 Q s_1$. Then:

- If $s_2 \stackrel{a}{\twoheadrightarrow} s_2'$ and $a \neq \tau$, there exist $s_1' \in \mathcal{T}_1$ such that $s_1 \stackrel{a}{\twoheadrightarrow} s_1'$, and $s_1' R s_2'$.
- If $s_2 \stackrel{\tau}{\twoheadrightarrow} s_2'$, then there exists a $s_1' \in \mathcal{T}_1$ such that $s_1 \twoheadrightarrow s_1'$ and $s_1' R s_2'$.
- If $s_2 \in\downarrow_2$, then there exists a $s_1' \in \mathcal{T}_1$ such that $s_1 \twoheadrightarrow s_1'$, $s_1' \in\downarrow_1$ and $s_1' R s_2$.

We note that contrasimulation sits between $\leftrightarrows_b$ and $\approx$ in Glabbeek's lattice of equivalences. For more details we again refer to [7].

Before we present any general results, we will first consider a number of examples. The first example is the "canonical" example of why transparency in process algebras can be troublesome.

Example 4.1. Consider the following specification S:

$$X = a.X \cdot Y + b.1$$
$$Y = c.1 + 1$$

In figure 8 one can see the transition system $\mathcal{T}_S(X)$. It is clear that the main problem occurs on the right: due to the fact that the name Y

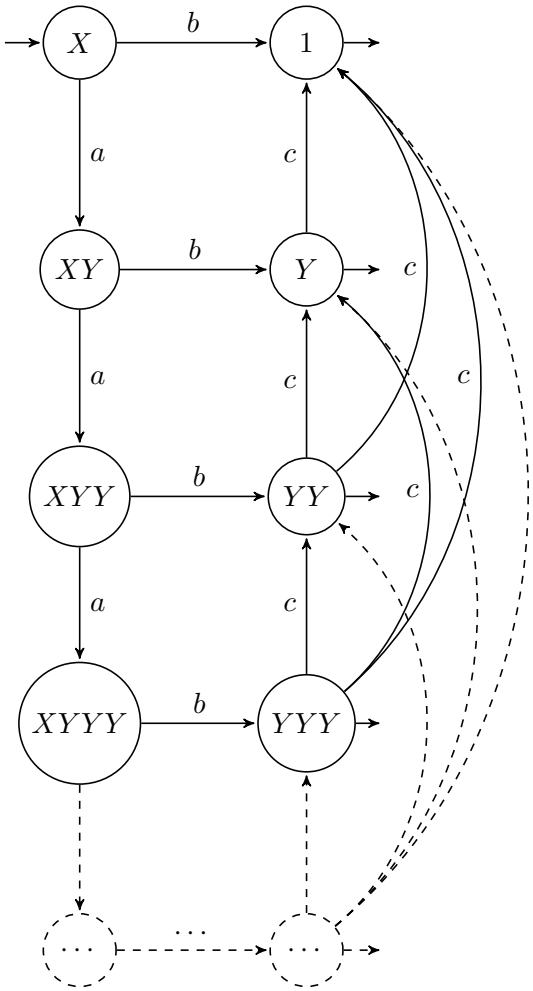

Figure 8: Transition system $\mathcal{T}_S(X)$

contains the summand 1, this means that it is possible to make the transition $Y^n \xrightarrow{c} Y^m$ for any $m < n$, and since one can also choose to pick the 1-summand in all Y's, we have that all Y^n states terminate.

Now, where the previous results worked for the FSES interpretation of PDA's, here we are forced to limit ourselves to FS. This is to be able to deal with the fact that arbitrarily large sequences of names can instantaneously terminate (provided that all names are transparent).

Now, our goal with this example is to find a PDA that will be equivalent to $\mathcal{T}_S(X)$ up to contrasimulation. We claim that the PDA P, shown in figure 9 will do exactly that. The associated transition system $\mathcal{T}(P)$ is shown in figure 10.

The transition system for P illustrates the basic principle for dealing with

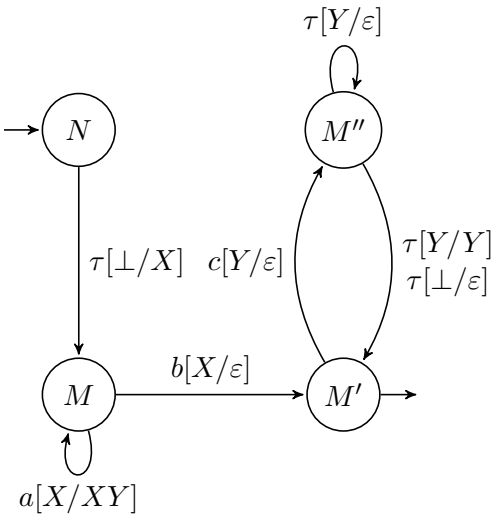

Figure 9: Push-down automaton P

transparency in the context of contrasimulation, so it will be informative to look at the contrasimulation (R, Q) in detail.

To begin with, we will define R as follows:

$$\begin{array}{ll} X & R(N, \bot) \\ \forall n \in \mathbb{N}. XY^n & R(M, XY^n) \\ \forall n \in \mathbb{N}. Y^n & R(M', Y^n) \\ \forall n \in \mathbb{N}. Y^n & R(M'', Y^n) \end{array}$$

And we will define the relation Q as follows:

$$\begin{array}{ll} (N, \bot) & QX \\ \forall n \in \mathbb{N}. (M, XY^n) & QXY^n \\ \forall n \in \mathbb{N}. (M', Y^n) & QY^n \end{array}$$

Note that none of the states originating from M'' have a relation in the Q-direction.

Now, we want to show that this is indeed a contrasimulation. We begin by noting that $XR(N, \bot)$ and $(N, \bot)QX$, meaning that we meet the requirement that the initial states should be related in both directions.

First, let $s_1 R s_2$. Then:

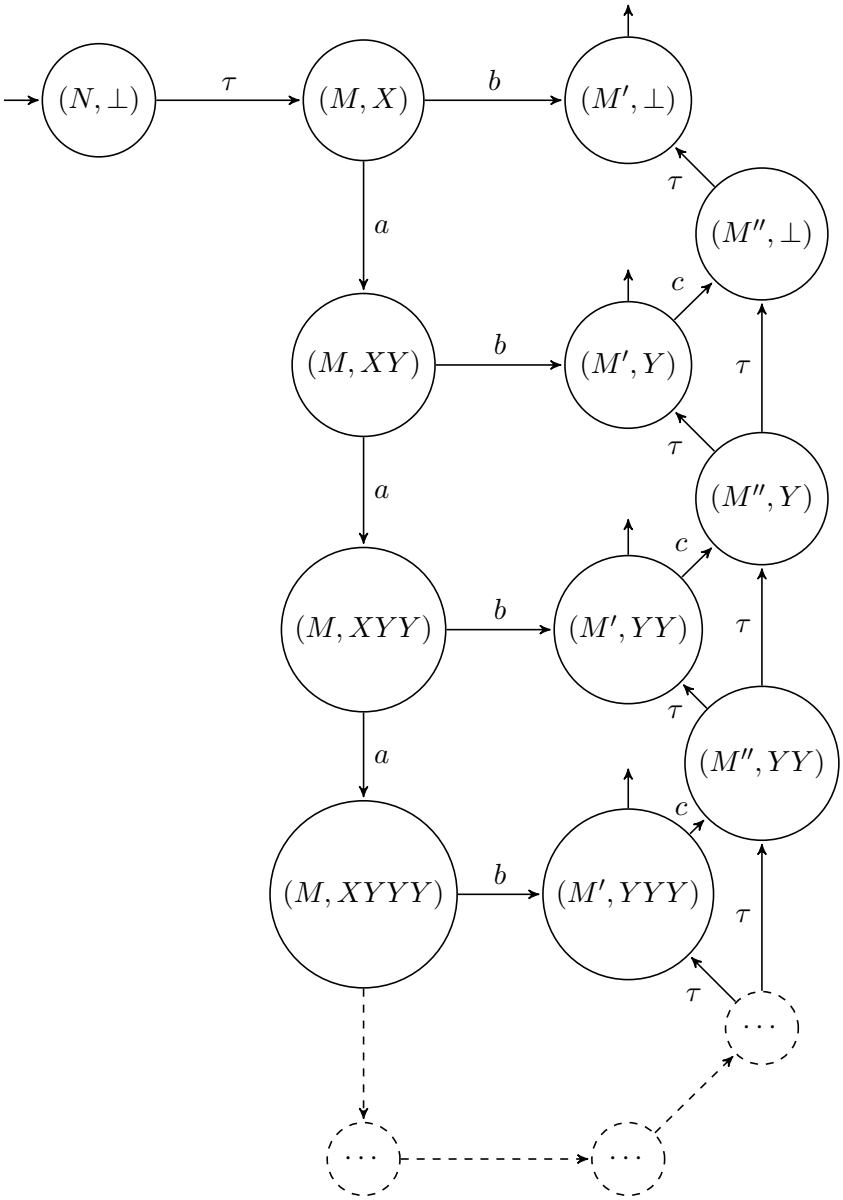

Figure 10: Transition system $\mathcal{T}(P)$

- If $s_1 \overset{a}{\twoheadrightarrow} s_1'$ and $a \neq \tau$, there exist $s_2' \in \mathcal{T}_2$ such that $s_2 \overset{a}{\twoheadrightarrow} s_2'$, and $s_2' Q s_1'$.

  If $s_1 = XY^n$ for any $n \in \mathbb{N}$, we must have $s_2 = (M, XY^n)$. Then the only options are to take an a or a b transition. The former transition will be:

  $$XY^n \overset{a}{\rightarrow} XY^{n+1}$$

  Which can be mirrored in $\mathcal{T}(P)$ by:

  $$(M, XY^n) \overset{a}{\rightarrow} (M, XY^{n+1})$$

  And we have that $(M, XY^{n+1}) Q XY^{n+1}$. Identical reasoning applies for the b-transition.

  The more interesting case is when $s_1 = Y^n$. We have two options here for s_2, but we begin by considering the case where $s_2 = (M'', Y^n)$ (the other option, where $s_2 = (M', Y^n)$, will also be covered by this case). We can then make the following transitions:

  $$\forall m < n. Y^n \overset{c}{\rightarrow} Y^m$$

  Now, we can mirror this as follows:

  $$(M'', Y^n) \overset{\tau}{\rightarrow} (M', Y^n) \overset{c}{\rightarrow} (M'', Y^{n-1}) \twoheadrightarrow (M', Y^m)$$

  And since $(M', Y^m) Q Y^m$, the requirement is met.

- The requirement about τ-transitions need not be considered, as $\mathcal{T}_S(X)$ contains none.

- If $s_1 \in \downarrow_1$, then there exists a $s_2' \in \mathcal{T}_2$ such that $s_2 \twoheadrightarrow s_2'$, $s_2' \in \downarrow_2$ and $s_2' Q s_1$.

  If s_1 terminates, then it must be a state Y^n for some $n \in \mathbb{N}$. To begin with, if we then have that $s_2 = (M', Y^n)$, then there is no problem, since $(M', Y^n) \in \downarrow_2$ and $(M', Y^n) Q Y^n$.

  If $s_2 = (M'', Y^n)$, then one can simply make the transition:

  $$(M'', Y^n) \overset{\tau}{\rightarrow} (M', Y^n)$$

  to terminate properly.

Now all that is left is to check the requirements in the other direction. Let $s_2 Q s_1$:

- If $s_2 \stackrel{a}{\twoheadrightarrow} s_2'$ and $a \neq \tau$, there exist $s_1' \in \mathcal{T}_1$ such that $s_1 \stackrel{a}{\twoheadrightarrow} s_1'$, and $s_1' R s_2'$.

  We will only consider the interesting case here, as all other cases are straightforward. We therefore let $s_2 = (M', Y^n)$ and $s_1 = Y^n$. The only transition we will consider is the following:

  $$(M', Y^n) \stackrel{c}{\twoheadrightarrow} (M'', Y^m)$$

  Where $m < n$. This will be mirrored in $\mathcal{T}_S(X)$ by the following:

  $$Y^n \stackrel{c}{\rightarrow} Y^m$$

  Which meets the requirement, since $Y^m R (M'', Y^m)$.

- Again, the case with τ-transitions need not be considered, since there is no s_2 that stands in relation Q to some S_2 that can make τ-transitions (without mixing in at least one action).

- Regarding terminating states the proof will work like in the R direction.

This therefore shows that (R, Q) is a contrasimulation.

The previous example illustrated the basic problem of transparency. However, it is not the only difficulty that transparancy brings. We will give a few more examples of problems that may arise when dealing with transparent names, and we'll sketch how these problems are solved in the general proof.

Example 4.2. Consider the following sequential specification S:

$$\begin{aligned} X_1 &= a.X_2 \cdot Y + b.1 \\ X_2 &= a.X_2 \cdot Z + b.1 \\ Y &= c.1 + 1 \\ Z &= d.1 + 1 \end{aligned}$$

The main difference here, compared to the previous example, is that one builds a stack of both Y's and Z's, both of which are transparent.

The associated transition system $\mathcal{T}_S(X_1)$ can be seen in figure 11. Now, the most important thing to note here is that most paths, after they have performed the b action, can both make a d-transition from any of the Z's present in the sequential composition, or a c action, from the single Y.

This illustrates a more general problem when trying to find a matching PDA: one needs to know exactly which transparent names are reachable from

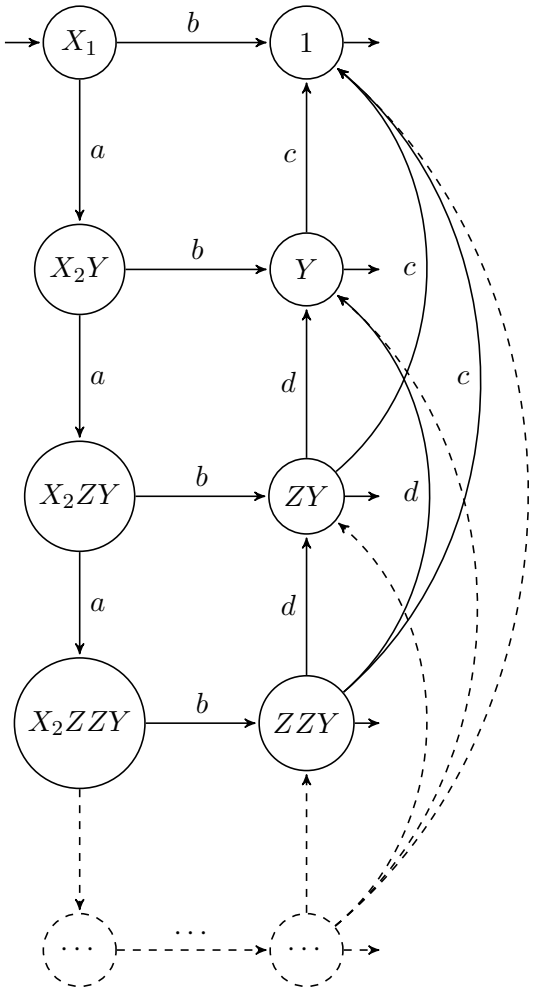

Figure 11: Transition system $\mathcal{T}_S(X)$

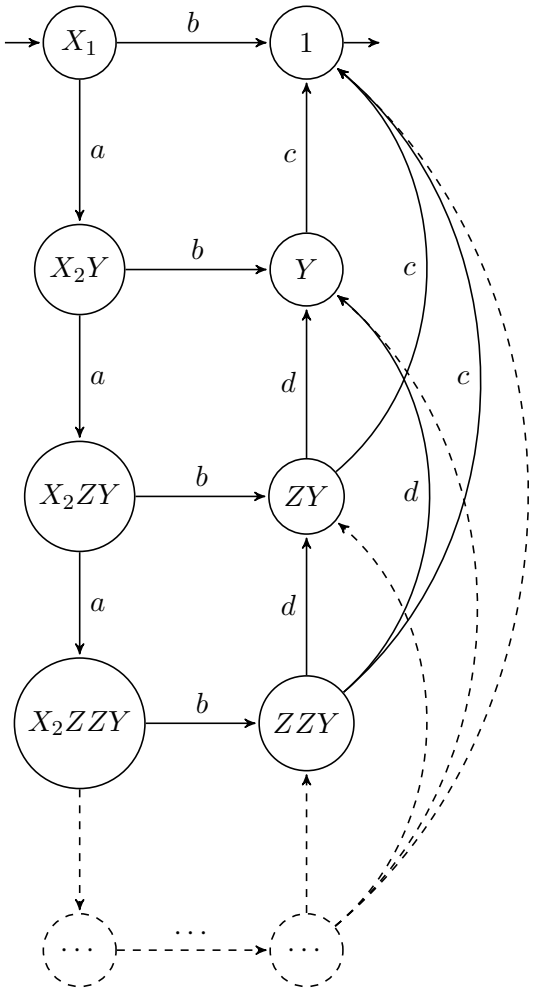

Figure 12: Transition system $\mathcal{T}_S(X)$ where $Y = c.1$

where one is. In the more general solution this will be solved as followed: we will "save" this information in the name of the state, by tacking an element from $\mathcal{P}(\mathcal{N}_{tr})$ (that is, the powerset of $\mathcal{N}_{tr}$, where $\mathcal{N}_{tr}$ is the set of transparent names) to the PDA state, so that at each moment it is clear which are reachable.

Furthermore we note that one needs to know one additional thing: say that in this example Y was not transparent, for instance, $Y = c.1$. In this case one would in fact be obliged to perform a c before one could terminate. The transition system in this case can be seen in figure 12. All it would change is that one cannot terminate at any state Z^nY anymore. We will therefore in the general solution also keep track of the first upcoming opaque name (if any).

We will now give the more general result regarding transparency in se-

quential specifications and contrasimulation.

Theorem 4.1. *Let S be a sequential specification defined by a set of variables $\{X_i, Y_j | i = 0, 1, ..., n \wedge j = 0, 1, ..., m\}$ as follows:*

$$X_i = \sum_{k \in I_i} a_{ik}.\xi_{ik} + 1$$
$$Y_i = \sum_{l \in I_j} a_{jl}.\xi_{jl}$$

Here ξ is a sequential composition of at most 2 variables, where zero variables is understood to be equal to $\mathbf{1}$. Let Z be a name in S.

There exists a PDA P under the FS-interpretation such that $\mathcal{T}_S(Z)$ is contrasimilar to $\mathcal{T}(P)$.

Proof. We will first outline in general terms the construction of the PDA P. To begin with, we will introduce two types of segment-symbols.

The first type is α^{χ, Y_j}. Here $\chi \subseteq \{X_i | i = 0, 1, ..., n\}$. This segment symbol will, in words, mean the following: there will follow a number of transparent names in the stack, each of which is in χ, and the first opaque name after this is Y_j.

The second segment symbol we'll introduce is ω^{X_i}. This symbol will signify that P has just passed the final X_i, either in the current transparent part of the stack, or in the entire stack (if the stack contains no more opaque names).

To give an example, consider the following sequence of names:

$$X_1X_2X_1Y_1X_1, X_3X_3Y_2Y_2X_1$$

The PDA P will then store this name as the following stack:

$$X_1X_2\omega^{X_2}X_1\omega^{X_1}Y_1\alpha^{\{X_1,X_3\},Y_2}X_1\omega^{X_1}X_3X_3\omega^{X_3}Y_2Y_2\alpha^{\{X_1\},\perp}X_1\omega^{X_1}$$

Now to give a more precise specification of P. We will create a number of states in P, since we intend to "save" up to three things in the name of the state.

- We will have a state $M^{\emptyset}$. Here we know that the top element of the stack is opaque.
- For all $\chi \subseteq \{X_i | i = 0, 1, ..., n\}$, that is, all subsets of transparent names, and all Y_j, there will be a state M^{χ, Y_j}. These states contain the information that there is a transparent part of the stack with the names χ, followed by the opaque name Y_j.

- We will have states $M^{\chi,\perp}$, which signify that the entire stack is transparent, and contains the names in χ.
- For every $X_i \in \chi$ and ξ such that $a.\xi$ is a summand of X_i, there will be a state $M^{\chi,Y_j}_{X_i,\xi}$ (and $M^{\chi,\perp}_{X_i,\xi}$). These states contain the information that ξ is yet to be added to the stack, and that this ξ comes from an X_i.
- For every Y_j and every summand $a_{jl}.\xi_{jl}$ there will be a state $M^{\emptyset}_{Y_j,\xi_{jl}}$. This state has as its purpose to remove any transparent names from the stack, and add ξ_{jl} once the Y_j on the stack is reached.
- For all ξ such that $a.\xi$ is a summand of some Y_j, there will be a state $M^{\emptyset,?}_{\xi}$. These are intermediary states to determine the next opaque name in the stack, so as to correctly keep track of this.
- Finally, there will be a starting state N, and an extra terminating state T.

To make the specification more readable, we will write as $\xi \in Z$ when we mean that there exists an a such that $a.\xi$ is a summand of Z. We also let $\mathcal{X}$ be the set of all transparent names, and $\mathcal{Y}$ the set of all opaque names.

We will now begin with the proper specification of P. To begin with, there are two options for the starting state N, depending on whether or not Z is opaque. If it is, we have the following transition:

$$N \xrightarrow{\tau[\perp/Z]} M^{\emptyset}$$

If it is transparent, we have the following transition:

$$N \xrightarrow{\tau[\perp/Z\omega^Z]} M^{\{Z\},\perp}$$

For $M^{\emptyset}$, the state dealing with opaque names, we will have the following transitions:

$$\forall Y_j \in \mathcal{Y}, l \in I_j, \xi_{jl} = 1.M^{\emptyset} \xrightarrow{a_{jl}[Y_j/\varepsilon]} M^{\emptyset}$$
$$\forall Y_j, Y \in \mathcal{Y}, l \in I_j, \xi_{jl} = Y.M^{\emptyset} \xrightarrow{a_{jl}[Y_j/Y]} M^{\emptyset}$$
$$\forall Y_j, Y, Y' \in \mathcal{Y}, l \in I_j, \xi_{jl} = YY'.M^{\emptyset} \xrightarrow{a_{jl}[Y_j/YY']} M^{\emptyset}$$
$$\forall Y_j, Y \in \mathcal{Y}, X \in \mathcal{X}, l \in I_j, \xi_{jl} = XY.M^{\emptyset} \xrightarrow{a_{jl}[Y_j/X\omega^X Y]} M^{\{X\},Y}$$
$$\forall Y_j \in \mathcal{Y}, X \in \mathcal{X}, l \in I_j, \xi_{jl} = X.M^{\emptyset} \xrightarrow{a_{jl}[Y_j/\varepsilon]} M_X^{\emptyset,?}$$
$$\forall Y_j, Y \in \mathcal{Y}, X \in \mathcal{X}, l \in I_j, \xi_{jl} = YX.M^{\emptyset} \xrightarrow{a_{jl}[Y_j/\varepsilon]} M_{YX}^{\emptyset,?}$$
$$\forall Y_j \in \mathcal{Y}, X, X' \in \mathcal{X}, l \in I_j, \xi_{jl} = XX'.M^{\emptyset} \xrightarrow{a_{jl}[Y_j/\varepsilon]} M_{XX'}^{\emptyset,?}$$
$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}.M^{\emptyset} \xrightarrow{\tau[\alpha^{\chi,y}/\varepsilon]} M^{\chi,y}$$
$$M^{\emptyset} \xrightarrow{\tau[\bot/\varepsilon]} T$$

We will also detail the functioning of the states of the type $M^{\emptyset,?}$.

$$\forall X \in \mathcal{X}, y \in \mathcal{Y} \cup \{\bot\}.M_X^{\emptyset,?} \xrightarrow{\tau[y/X\omega^X y]} M^{\{X\},y}$$
$$\forall X \in \mathcal{X}, \chi \in \mathcal{P}(\mathcal{X}), X \notin \chi, y \in \mathcal{Y} \cup \{\bot\}.M_X^{\emptyset,?} \xrightarrow{\tau[\alpha^{\chi,y}/X\omega^X]} M^{\chi\cup\{X\},y}$$
$$\forall X \in \mathcal{X}, \chi \in \mathcal{P}(\mathcal{X}), X \in \chi, y \in \mathcal{Y} \cup \{\bot\}.M_X^{\emptyset,?} \xrightarrow{\tau[\alpha^{\chi,y}/X]} M^{\chi,y}$$
$$\forall X \in \mathcal{X}.M_X^{\emptyset,?} \xrightarrow{\tau[\bot/X\omega^X]} M^{\{X\},\bot}$$
$$\forall X \in \mathcal{X}, Y \in \mathcal{Y}, y \in \mathcal{Y} \cup \{\bot\}.M_{YX}^{\emptyset,?} \xrightarrow{\tau[y/Y\alpha^{\{X\},y}X\omega^X y]} M^{\emptyset}$$
$$\forall X \in \mathcal{X}, \chi \in \mathcal{P}(\mathcal{X}), Y \in \mathcal{Y}, X \notin \chi, y \in \mathcal{Y} \cup \{\bot\}.M_{YX}^{\emptyset,?} \xrightarrow{\tau[\alpha^{\chi,y}/Y\alpha^{\chi\cup\{X\},y}X\omega^X]} M^{\emptyset}$$
$$\forall X \in \mathcal{X}, \chi \in \mathcal{P}(\mathcal{X}), Y \in \mathcal{Y}, X \in \chi, y \in \mathcal{Y} \cup \{\bot\}.M_{YX}^{\emptyset,?} \xrightarrow{\tau[\alpha^{\chi,y}/Y\alpha^{\chi,y}X]} M^{\emptyset}$$
$$\forall X \in \mathcal{X}, Y \in \mathcal{Y}.M_{YX}^{\emptyset,?} \xrightarrow{\tau[\bot/Y\alpha^{\chi,\bot}X\omega^X]} M^{\emptyset}$$

$$\forall X, X' \in \mathcal{X}, X \neq X', y \in \mathcal{Y} \cup \{\bot\}. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[y/X\omega^X X'\omega^{X'} y]} M^{\{X,X'\},y}$$

$$\forall X, X' \in \mathcal{X}, X \neq X', \chi \in \mathcal{P}(\mathcal{X}), X, X' \notin \chi, y \in \mathcal{Y} \cup \{\bot\}. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[\alpha^{\chi,y}/X\omega^X X'\omega^{X'}]} M^{\chi\cup\{X,X'\},y}$$

$$\forall X, X' \in \mathcal{X}, X \neq X', \chi \in \mathcal{P}(\mathcal{X}), X \notin \chi, X' \in \chi, y \in \mathcal{Y} \cup \{\bot\}. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[\alpha^{\chi,y}/X\omega^X X']} M^{\chi\cup\{X\},y}$$

$$\forall X, X' \in \mathcal{X}, X \neq X', \chi \in \mathcal{P}(\mathcal{X}), X \in \chi, X' \notin \chi, y \in \mathcal{Y} \cup \{\bot\}. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[\alpha^{\chi,y}/XX'\omega^{X'}]} M^{\chi\cup\{X'\},y}$$

$$\forall X, X' \in \mathcal{X}, X \neq X', \chi \in \mathcal{P}(\mathcal{X}), X, X' \in \chi, y \in \mathcal{Y} \cup \{\bot\}. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[\alpha^{\chi,y}/XX']} M^{\chi,y}$$

$$\forall X, X' \in \mathcal{X}, X = X', y \in \mathcal{Y} \cup \{\bot\}. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[y/XX\omega^X y]} M^{\{X\},y}$$

$$\forall X, X' \in \mathcal{X}, X = X', \chi \in \mathcal{P}(\mathcal{X}), X, X' \notin \chi, y \in \mathcal{Y} \cup \{\bot\}. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[\alpha^{\chi,y}/XX\omega^X]} M^{\chi\cup\{X\},y}$$

$$\forall X, X' \in \mathcal{X}, X = X', \chi \in \mathcal{P}(\mathcal{X}), X, X' \in \chi, y \in \mathcal{Y} \cup \{\bot\}. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[\alpha^{\chi,y}/XX]} M^{\chi,y}$$

$$\forall X, X' \in \mathcal{X}, X \neq X'. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[\bot/X\omega^X X'\omega^{X'}]} M^{\{X,X'\},\bot}$$

$$\forall X, X' \in \mathcal{X}, X = X'. M^{\emptyset,?}_{XX'} \xrightarrow{\tau[\bot/XX\omega^X]} M^{\{X\},\bot}$$

We will now detail the specification of states of the type $M^{\chi,y}$, that is, states that deal with transparent parts of the stack.

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X_i, X \in \chi, k \in I_k. M^{\chi,y} \xrightarrow{a_{ik}[X/X]} M^{\chi,y}_{X_i,\xi_{ik}}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), X \in \chi \cup \{\omega^{X'} | X' \in \chi\}, Y_j \in \mathcal{Y}, l \in I_j. M^{\chi,Y_j} \xrightarrow{a_{jl}[X/\varepsilon]} M^{\emptyset}_{Y_j,\xi_{jl}}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X_i \in \chi. M^{\chi,y} \xrightarrow{\tau[\omega^{X_i}/\varepsilon]} M^{\chi-\{X_i\},y}$$

For the last transition we note that $M^{\emptyset,y} = M^{\emptyset}$.

We will now detail the transitions of states of the type $M^{\chi,y}_{X_i,\xi_{ik}}$. In these states one can throw away parts of the transparent stack until one reaches the X_i one wishes to "execute".

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X_i, X \in \chi, k \in I_k . M^{\chi,y}_{X_i,\xi_{ik}} \xrightarrow{\tau[X/\varepsilon]} M^{\chi,y}_{X_i,\xi_{ik}}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X_i, X \in \chi, X_i \neq Xk \in I_k . M^{\chi,y}_{X_i,\xi_{ik}} \xrightarrow{\tau[\omega^X/\varepsilon]} M^{\chi-\{X\},y}_{X_i,\xi_{ik}}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X_i \in \chi . M^{\chi,y}_{X_i,1} \xrightarrow{\tau[X_i/\varepsilon]} M^{\chi,y}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X \in \mathcal{X}, X_i, X \in \chi . M^{\chi,y}_{X_i,X} \xrightarrow{\tau[X_i/X]} M^{\chi,y}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X \in \mathcal{X}, X_i \in \chi, X \notin \chi . M^{\chi,y}_{X_i,X} \xrightarrow{\tau[X_i/X\omega^X]} M^{\chi\cup\{X\},y}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, Y \in \mathcal{Y}, X_i \in \chi . M^{\chi,y}_{X_i,Y} \xrightarrow{\tau[X_i/Y\alpha^{\chi,y}]} M^{\emptyset}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X, X' \in \mathcal{X}, X_i, X, X' \in \chi . M^{\chi,y}_{X_i,XX'} \xrightarrow{\tau[X_i/XX']} M^{\chi,y}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X, X' \in \mathcal{X}, X_i, X \in \chi, X' \notin \chi . M^{\chi,y}_{X_i,XX'} \xrightarrow{\tau[X_i/XX'\omega^{X'}]} M^{\chi\cup\{X'\},y}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X, X' \in \mathcal{X}, X_i, X' \in \chi, X \notin \chi . M^{\chi,y}_{X_i,XX'} \xrightarrow{\tau[X_i/X\omega^X X']} M^{\chi\cup\{X\},y}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X, X' \in \mathcal{X}, X_i \in \chi, X, X' \notin \chi . M^{\chi,y}_{X_i,XX'} \xrightarrow{\tau[X_i/X\omega^X X'\omega^{X'}]} M^{\chi\cup\{X,X'\},y}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X \in \mathcal{X}, Y \in \mathcal{Y}, X_i, X \in \chi . M^{\chi,y}_{X_i,YX} \xrightarrow{\tau[X_i/Y\alpha^{\chi,y}X]} M^{\emptyset}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X \in \mathcal{X}, Y \in \mathcal{Y}, X_i \in \chi, X \notin \chi . M^{\chi,y}_{X_i,YX} \xrightarrow{\tau[X_i/Y\alpha^{\chi\cup\{X\},y}X\omega^X]} M^{\emptyset}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, X \in \mathcal{X}, Y \in \mathcal{Y}, X_i \in \chi . M^{\chi,y}_{X_i,XY} \xrightarrow{\tau[X_i/X\omega^X Y\alpha^{\chi,y}]} M^{\{X\},Y}$$

$$\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, Y, Y' \in \mathcal{Y}, X_i \in \chi . M^{\chi,y}_{X_i,YY'} \xrightarrow{\tau[X_i/YY'\alpha^{\chi,y}]} M^{\emptyset}$$

The last states to be specified will be states of the type $M^{\emptyset}_{Y_j,\xi_{jl}}$. The purpose of these states is to clean out the entire transparent part of the stack.

$$\forall Y_j \in \mathcal{Y}, l \in I_j, X \in \mathcal{X} \cup \{\omega^{X'} | X' \in \mathcal{X}\}. M^{\emptyset}_{Y_j,\xi_{jl}} \xrightarrow{\tau[X/\varepsilon]} M^{\emptyset}_{Y_j,\xi_{jl}}$$

$$\forall Y_j \in \mathcal{Y}. M^{\emptyset}_{Y_j,1} \xrightarrow{\tau[Y_j/\varepsilon]} M^{\emptyset}$$

$$\forall Y_j \in \mathcal{Y}, X \in \mathcal{X}. M^{\emptyset}_{Y_j,X} \xrightarrow{\tau[Y_j/\varepsilon]} M^{\emptyset,?}_{X}$$

$$\forall Y_j, Y \in \mathcal{Y}. M^{\emptyset}_{Y_j,Y} \xrightarrow{\tau[Y_j/Y]} M^{\emptyset}$$

$$\forall Y_j \in \mathcal{Y}, X, X' \in \mathcal{X}. M^{\emptyset}_{Y_j,XX'} \xrightarrow{\tau[Y_j/\varepsilon]} M^{\emptyset,?}_{XX'}$$

$$\forall Y_j, Y \in \mathcal{Y}, X \in \mathcal{X}. M^{\emptyset}_{Y_j,XY} \xrightarrow{\tau[Y_j/X\omega^X Y]} M^{\{X\},Y}$$

$$\forall Y_j, Y \in \mathcal{Y}, X \in \mathcal{X}. M^{\emptyset}_{Y_j,YX} \xrightarrow{\tau[Y_j/\varepsilon]} M^{\emptyset,?}_{YX}$$

$$\forall Y_j, Y, Y' \in \mathcal{Y}. M^{\emptyset}_{Y_j,YY'} \xrightarrow{\tau[Y_j/YY']} M^{\emptyset}$$

This covers all the states and transitions of P. As mentioned before, we have N as the initial state. Other than that, any $M^{\chi,\perp}$ can terminate (as the entire stack is transparent), and T can terminate.

Now that we have fully specified P, all that is left to give a R and Q that satisfy the requirements for contrasimulation.

We will first introduce some extra notation. Firstly, we let ζ again be any sequence of names. We let ζ^+ be the sequence ζ, with the correct segment symbols added in the appropriate places (that is, $\alpha^{\chi,y}$'s at the start of transparent segments, with correct labels, and ω^X's after each last X of its type in a transparent segment.). If ζ contains any segment symbols, we will understand ζ^- to mean the same sequence, but without the segment symbols. Additionally, we will understand $\zeta^=$ to be the set of transparent names found in the first segment of transparent names (so its $\emptyset$ if the first element of ζ is opaque), and $\zeta^\Rightarrow$ to be the first opaque name found in ζ, after a transparent segment. We remind the reader that we interpret $M^{\emptyset,Y}$ as $M^{\emptyset}$. Lastly we let W be the set of all segment symbols

We will define $R \subseteq \mathcal{T}_S(Z) \times \mathcal{T}(P)$ as follows:

$$
\begin{array}{rl}
Z & R(N, \bot) \\
1 & R(T, \bot) \\
\forall \zeta \in S^*.\zeta & R(M^{\zeta^=, \zeta^\Rightarrow}, \zeta^+) \\
\forall \zeta \in S^*, x \in \mathcal{X}.X\zeta & R(M_X^{\emptyset,?}, (X\zeta)^+) \\
\forall \zeta \in S^*, V, V' \in S.VV'\zeta & R(M_{VV'}^{\emptyset,?}, (VV'\zeta)^+) \\
\forall \zeta \in S^*, \zeta' \in (\mathcal{X} - \{X_i\})^*, X_i \in \mathcal{X}, k \in I_i.\xi_{ik}\zeta & R(M_{X_i,\xi_{ik}}^{\chi,y}, (\zeta' X_i \zeta)^+) \\
\forall \zeta \in S^*, \zeta' \in \mathcal{X}^*, Y_j \in \mathcal{Y}, L \in I_j.\xi_{jl}\zeta & R(M_{Y_j,\xi_{jl}}^{\emptyset}, (\zeta' Y_j \zeta)^+)
\end{array}
$$

And we define $Q \subseteq \mathcal{T}(P) \times \mathcal{T}_S(Z)$ as follows:

$$
\begin{array}{rl}
(N, \bot) & QZ \\
(T, \bot) & Q1 \\
\forall \chi \in \mathcal{P}(\mathcal{X}), y \in \mathcal{Y} \cup \{\bot\}, \zeta \in (S \cup W)^*.(M^{\chi,y}, \zeta) & Q\zeta^- \\
\forall \zeta \in (S \cup W)^*.(M^{\emptyset}, \zeta) & Q\zeta^- \\
\forall \zeta \in (S \cup W)^*, X \in \mathcal{X}.(M_X^{\emptyset,?}, \zeta) & QX\zeta^- \\
\forall \zeta \in (S \cup W)^*, V, V' \in S.(M_{VV'}^{\emptyset,?}, \zeta) & QVV'\zeta^- \\
\forall \zeta \in (S \cup W)^*, \zeta' \in (\mathcal{X} \cup W)^*, Y_j \in \mathcal{Y}, l \in I_j.(M_{Y_j,\xi_{jl}}^{\emptyset}, \zeta' Y_j \zeta) & Q\xi_{jl}\zeta^-
\end{array}
$$

Now all that remains is to show that (R, Q) is a contrasimulation. We begin by noting that $ZR(N, \bot)$ and $(N, \bot)QZ$, so the initial states relate to eachother.

We will begin by examining if contrasimulation holds from R to Q. Let $s_1 R s_2$.

- If $s_1 \xrightarrow{a}\!\!\!\!\rightarrow s_1'$ and $a \neq \tau$, there exist $s_2' \in \mathcal{T}_2$ such that $s_2 \xrightarrow{a}\!\!\!\!\rightarrow s_2'$, and $s_2' Q s_1'$.

  First consider the case in which $s_1 = Y_j\zeta$ for some $Y_j \in \mathcal{Y}$. The only non-τ transition one could make from there is $Y_j\zeta \xrightarrow{a_{jl}} \xi_{jl}\zeta$ for some summand $a_{jl}.\xi_{jl}$ of Y_j. There are now several option for what state s_2 could be:

  - $s_2 = (M^{\emptyset}, (Y_j\zeta)^+)$. In this case, the exact transitions will depend on ξ_{jl} (in particular, it will determine if one has to detour through $M_{\xi_{jl}}^{\emptyset,?}$). However, in all cases one has:

    $$(M^{\emptyset}, (Y_j\zeta)^+) \xrightarrow{a_{jl}}\!\!\!\!\rightarrow (M^{(\xi_{jl}\zeta)^=, (\xi_{jl}\zeta)^\Rightarrow}, (\xi_{jl}\zeta)^+)$$

    And we have that $(M^{(\xi_{jl}\zeta)^=, (\xi_{jl}\zeta)^\Rightarrow}, (\xi_{jl}\zeta)^+) Q \xi_{jl}\zeta$.

– If $\zeta = X_i\zeta'$, we could have $s_2 = (M^{\emptyset,?}_{Y_jX_i}, \zeta'^+)$. However, we have:

$$(M^{\emptyset,?}_{Y_jX_i}, \zeta'^+) \xrightarrow{\tau} (M^{\emptyset}, (Y_jX_i\zeta')^+) = (M^{\emptyset}, (Y_j\zeta)^+)$$

So this case reduces to the previous one.

– If there exists an X_i that contains a summand ξ_{ik} such that $\xi_{ik} = Y_j\xi'_{ik}$ and $\zeta = \xi'_{ik}\zeta'$, we could have $s_2 = (M^{\chi,y}_{X_i,Y_j\xi'_{ik}}, (\zeta'' X_i\zeta'))$, where $\zeta'' \in \{X|X \in \chi\}^*$. However, here we have:

$$(M^{\chi,y}_{X_i,Y_j\xi'_{ik}}, (\zeta' X_i\zeta)) \twoheadrightarrow (M^{\emptyset}, (Y_j\xi'_{ik}\zeta')^+) = (M^{\emptyset}, (Y_j\zeta)^+)$$

Which means that this case again reduces to the first.

– If there exists an Y'_j that contains a summand ξ'_{jl} such that $\xi'_{jl} = Y_j\xi''_{jl}$ and $\zeta = \xi''_{jl}\zeta'$, we could have $s_2 = (M^{\emptyset}_{Y'_j,Y_j\xi''_{jl}}, (\zeta'' Y'_j\zeta')^+)$, where $\zeta'' \in \mathcal{X}^*$. However, again, from here we have:

$$(M^{\emptyset}_{Y'_j,Y_j\xi''_{jl}}, (\zeta'' Y'_j\zeta')^+) \twoheadrightarrow (M^{\emptyset}, (Y_j\xi''_{jl}\zeta')^+) = (M^{\emptyset}, (Y_j\zeta)^+)$$

In the case that $s_1 = X_i\zeta$, the only possible transition from there is of the type $X_i\zeta \xrightarrow{a_{ik}} \xi_{ik}\zeta$. We have as the most important possibility for s_2 that $s_2 = (M^{(X_i\zeta)^=,(X_i\zeta)^\Rightarrow}, (X_i\zeta)^+)$. For this state we have that:

$$(M^{(X_i\zeta)^=,(X_i\zeta)^\Rightarrow}, (X_i\zeta)^+) \xrightarrow{a_{ik}} (M^{(X_i\zeta)^=,(X_i\zeta)^\Rightarrow}_{X_i,\xi_{ik}}, (X_i\zeta)^+)$$

From which we have the option for the following transition(s):

$$(M^{(X_i\zeta)^=,(X_i\zeta)^\Rightarrow}_{X_i,\xi_{ik}}, (X_i\zeta)^+) \twoheadrightarrow (M^{(\xi_{ik}\zeta)^=,(\xi_{ik}\zeta)^\Rightarrow}, (\xi_{ik}\zeta)^+)$$

Which is what we needed, since $(M^{(\xi_{ik}\zeta)^=,(\xi_{ik}\zeta)^\Rightarrow}, (\xi_{ik}\zeta)^+)Q\xi_{ik}\zeta$.

All other options for s_2 reduce to this one in the same way as they did for $s_1 = Y_j\zeta$.

- If $s_1 \xrightarrow{\tau} s'_1$, then there exists a $s'_2 \in \mathcal{T}_2$ such that $s_2 \twoheadrightarrow s'_2$ and $s'_2Qs'_1$.

  For this point it suffices to note that if $\mathcal{T}_S(Z)$ contains a τ-transition, this will be because it is specified in the sequential specification. The proof will therefore be identical to that of the previous point.

- If $s_1 \in\downarrow_1$, then there exists a $s'_2 \in \mathcal{T}_2$ such that $s_2 \twoheadrightarrow s'_2$, $s'_2 \in\downarrow_2$ and s'_2Qs_1.

  If s_1 terminates, that means that there are two options. The first option is that $s_1 = 1$. In this case the only option is that $s_2 = (T, \perp)$. Since $(T, \perp) \in\downarrow$ and $(T, \perp)Q1$, the requirement is met.

The other option is that $s_1 = \zeta$, where $\zeta \in \mathcal{X}^*$, which is to say, ζ is fully transparent and can therefore terminate on the spot. We then have that $s_2 = (M^{\zeta^=,\perp}, \zeta^+)$ (or s_2 is a state that can reach $(M^{\zeta^=,\perp}, \zeta^+)$ in one or more τ-transitions). However, since we are in the FS interpretation, and $M^{\zeta^=,\perp}$ is a final state in P, we have that $(M^{\zeta^=,\perp}, \zeta^+) \in\downarrow$. Additionally, $(M^{\zeta^=,\perp}, \zeta^+)Q\zeta$, so this requirement is met in full.

The requirements for contrasimulation therefore hold if one goes from R to Q. The only remaining thing is to check if the inverse requirements hold, going from Q to R.

Let $s_2 Q s_1$:

- If $s_2 \xrightarrow{a} s_2'$ and $a \neq \tau$, there exist $s_1' \in \mathcal{T}_1$ such that $s_1 \xrightarrow{a} s_1'$, and $s_1' R s_2'$.

  Let $s_2 = (M^{\emptyset}, Y_j\zeta)$. Then all possible transitions of the type $\xrightarrow{a}$ will be $\xrightarrow{a_{jl}}$ for some summand a_{jl}, ξ_{jl} in Y_j. This then also means that we have a s_2'' such that:

  $$(M^{\emptyset}, Y_j\zeta) \xrightarrow{a_{jl}} s_2'' \twoheadrightarrow s_2'$$

  Now, we know that the only option for s_1 is that $s_1 = Y_j\zeta^-$. We also know that $Y_j\zeta^- \xrightarrow{a_{jl}} \xi_{jl}\zeta^-$. It can also easily be verified that for any such transition, we have that $\xi_{jl}\zeta^- R s_2''$. Additionally, any state s_2' reachable by τ-transitions from s_2'' will also be related as $\xi_{jl}\zeta^- R s_2'$, unless one makes a τ transition that is inherited from S (but in that case one can make that same τ-transition in S).

  We now consider the case where $s_2 = (M^{\chi,y}, X\zeta)$. Then all possible transitions of the type $\xrightarrow{a}$ will be $\xrightarrow{a_{ik}}$ for some summand a_{ik}, ξ_{ik} of some $X_i \in \chi$, or, if $y = Y_j \neq \perp$, then we can have $\xrightarrow{a_{ik}}$ for some summand a_{jl}, ξ_{jl} in Y_j.

  Given this s_2, our only option for s_1 is $s_1 = X\zeta^-$.

  We first consider the following transition:

  $$(M^{\chi,y}, X\zeta) \xrightarrow{a_{ik}} (M^{\chi,y}_{X_i,\xi_{ik}}, X\zeta)$$

  This one can be matched in $\mathcal{T}_S(Z)$ by the transition:

  $$X\zeta^- \xrightarrow{a_{ik}} \xi_{ik}\zeta'^-$$

  Where $\zeta = \zeta'' X_i \zeta'$, and $\zeta'' \in \chi - \{X_i\}$. Since $\xi_{ik}\zeta'^- R (M^{\chi,y}_{X_i,\xi_{ik}}, X\zeta)$, we have what we need. Note that if one takes any further τ-transitions

down from $(M^{\chi,y}_{X_i,\xi_{ik}}, X\zeta)$, these can be similarly matched from $X\zeta^-$ by jumping to the leftmost X_i still present in the stack.

We will now consider the case where $s_1 = X\zeta^-$ and we make the following transition:

$$(M^{\chi,y}, X\zeta) \xrightarrow{a_{jl}} (M^{\emptyset}_{Y_j,\xi_{jl}}, \zeta)$$

We again know that $s_1 = X\zeta^-$. Now let $\zeta = \zeta'' Y_j \zeta'$, where $\zeta'' \in \chi^*$. Then we have that:

$$X\zeta^- \xrightarrow{a_{jl}} \xi_{jl}\zeta'^-$$

And we know that $\xi_{jl}\zeta'^- R (M^{\emptyset}_{Y_j,\xi_{jl}}, \zeta)$. Further τ-transitions from $M^{\emptyset}_{Y_j,\xi_{jl}}, \zeta)$ (up to $(M^{\emptyset}, \xi_{jl}\zeta')$, where another non-$\tau$-transition (or a τ-transition prescribed by S) will have to be taken) will lead to states that all relate via R to $\xi_{jl}\zeta'^-$, meaning that the requirement is met.

We note that any other transition of the type $\xrightarrow{a}\!\!\!\!\!\rightarrow$ will be from states s_2 such that $s_2 \twoheadrightarrow s_2''$, where s_2'' is one of the aforementioned states, and $s_2'' Q s_1$, so these cases reduce the cases above.

- If $s_2 \xrightarrow{\tau}\!\!\!\!\!\rightarrow s_2'$, then there exists a $s_1' \in \mathcal{T}_1$ such that $s_1 \twoheadrightarrow s_1'$ and $s_1' R s_2'$.

  First, let $s_2 = (N, \bot)$. Then we have that $s_1 = Z$. The only transition possible from here is:

  $$(N, \bot) \xrightarrow{\tau[\bot/Z]} (M^{Z^=,\bot}, Z^+)$$

  From here only τ-transitions prescribed by S could occur, which can be treated as non-τ-transitions without loss of generality. However, we also have that $ZR(M^{Z^=,\bot}, Z^+)$, so the requirement is met in this case.

  Now let $s_2 = (M^{\emptyset}, \bot)$. We then have that $s_1 = 1$. From here only one τ-transition is possible, namely:

  $$(M^{\emptyset}, \bot) \xrightarrow{\tau} (T, \bot)$$

  However, we have that $1R(T, \bot)$, so again no problems occur.

  Now, let $s_2 = (M^{\emptyset}, \alpha^{\chi,y}\zeta)$. We then have that $s_1 = \zeta^-$. From here the only possible transition is:

  $$(M^{\emptyset}, \alpha^{\chi,y}\zeta) \xrightarrow{\tau} (M^{\chi,y}, \zeta)$$

Like before, from here only τ-transitions prescribed by S could occur, and again we have that $\zeta^- R (M^{\chi,y}, \zeta)$.

Now, let $s_2 = (M_\xi^{\emptyset,?}, \zeta)$. We then have that $s_1 = \xi\zeta^-$. The only possible transition from here is:

$$(M_\xi^{\emptyset,?}, \zeta) \xrightarrow{\tau} (M^{(\xi\zeta)^=,(\xi\zeta)^\Rightarrow}, (\xi\zeta)^+)$$

Again, from here only τ-transitions prescribed by S could occur. We also have that $\xi\zeta^- R (M^{(\xi\zeta)^=,(\xi\zeta)^\Rightarrow}, (\xi\zeta)^+)$.

Next, let $s_2 = (M^{\chi,y}, \omega^X \zeta)$. Here we have that $s_1 = (\omega^X \zeta)^- = \zeta^-$. From here our only option is the following transition:

$$(M^{\chi,y}, \omega^X \zeta) \xrightarrow{\tau} (M^{\chi - \{X\},y}, \zeta)$$

From where, once more, only τ-transitions prescribed by S could occur. However, like before, $\zeta^- R M^{\chi-\{X\},y}, \zeta)$, so the requirement is met in this case as well.

Finally, consider the case where $s_2 = (M_{Y,\xi}^{\emptyset}, \zeta' Y \zeta)$. We then can only have $s_1 = \xi\zeta^-$. From this state a set of transitions are possible. The entirety of ones options (again, barring τ-transitions found in S) is as follows:

$$(M_{Y,\xi}^{\emptyset}, \zeta' Y \zeta) \twoheadrightarrow (M^{(\xi\zeta)^=,(\xi\zeta)^\Rightarrow}, \xi\zeta)$$

It suffices here to note that for all states s_2'' passed in the above we have $\xi\zeta^- R s_2''$, meaning that the requirement is once again met.

This exhaust all the cases, meaning that the requirement is met.

- If $s_2 \in\downarrow_2$, then there exists a $s_1' \in \mathcal{T}_1$ such that $s_1 \twoheadrightarrow s_1'$, $s_1' \in\downarrow_1$ and $s_1' R s_2$.

  The reasoning here is pretty much identical to the one found going from R to Q: since we are in the FS interpretation of PDA's, we have that for any stack, we have that $(M^{\chi,\perp}, \zeta) \in\downarrow$, which corresponds to ζ^- being fully transparent. Since all terminating states are related to their corresponding state in both the Q and R direction, we conclude that this requirement is also met.

We have therefore shown that (R, Q) defines a contrasimulation, which concludes the proof. □

5 Combining Transparency and Head-Recursion

In this section we will examine the combination of the previous two results. We'll do this by extending the previous result to include head-recursive names as well as transparent ones.

Theorem 5.1. *Let S be a sequential specification defined by a set of variables $\{W_f, X_i, Y_j | f = 0, 1, ..., n_W \wedge i = 0, 1, ..., n_X \wedge j = 0, 1, ..., n_Y\}$ as follows:*

$$W_f = \sum_{g \in I_f} W_f \cdot V_{fg} + \sum_{h \in I'_f} a_{fh}.\xi_{fh}(+1)$$
$$X_i = \sum_{k \in I_i} a_{ik}.\xi_{ik} + 1$$
$$Y_j = \sum_{l \in I_j} a_{jl}.\xi_{jl}$$

Here V is a name from S, and ξ is a sequential composition of at most 2 variables, where zero variables is understood to be equal to $\mathbf{1}$. Let Z be a name in S.

There exists a PDA P under the FS-interpretation such that $\mathcal{T}_S(Z)$ is contrasimilar to $\mathcal{T}(P)$.

Proof. We let $\mathcal{W}$ be the set of W_f. Furthermore, we let $\mathcal{W}_{op}$ be the set of opaque head-recursive names (that is, without +1), and $\mathcal{W}_{tr}$ be the set of transparent head-recursive names. Furthermore, we let $\mathcal{N}_{tr} = \mathcal{X} \cup \mathcal{W}_{tr}$ (that is, all transparent names), and $\mathcal{N}_{op} = \mathcal{Y} \cup \mathcal{W}_{op}$ (which is the set of all opaque names). We also let $\mathcal{D}$ be the set of all possible data that P can have on its stack.

Combining transparent names and head-recursive names does not really give rise to any new problems. One can simply extend the PDA given in the previous section with the "machinery" for head-recursive names. To begin with, qua bookkeeping we will treat $\mathcal{W}_{op}$ the same as any opaque name, and $\mathcal{W}_{tr}$ will be treated as transparent names. The main difficulty encountered is that the "machinery" creating the headrecursive stack will have to be modified to properly do all the bookkeeping required (which is to say, add the right marker symbols, and keep the right information in the state-name).

For this we will add a set of states to P that will be labelled L. We will also reintroduce the segment symbol $\diamond$. We will now also label the $\diamond$ segment symbol with extra information, to keep track of what is "behind" it, since we must now also keep track of that.

Now, the specification of P is, for the most part, the same as in the previous result. We will therefore only give the specification for the new parts of P, that is, the parts dealing with head-recursion.

These specifications don't introduce any particular new ideas. However, to do all the bookkeeping correctly, it is quite lengthy.

We begin with the specification for opaque head-recursive names. The first thing that occurs is that we go to a state $L^{?}_{W_f}$, the purpose of which is to label the segment symbol $\diamond$ correctly:

$$\begin{aligned}
\forall W_f \in \mathcal{W}_{op}. M^{\perp} &\xrightarrow{\tau[W_f/\varepsilon]} L^{?}_{W_f} \\
\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{N}^{*}_{tr}, y \in \mathcal{N}_{op} \cup \{\perp\}. L^{?}_{W_f} &\xrightarrow{\tau[\alpha^{\chi,y}/\diamond^{\chi,y}]} L^{\chi,y}_{W_f} \\
\forall W_f \in \mathcal{W}_{op}, Y \in \mathcal{N}_{op}. L^{?}_{W_f} &\xrightarrow{\tau[Y/\diamond^{\emptyset,Y}Y]} L^{\emptyset,Y}_{W_f} \\
\forall W_f \in \mathcal{W}_{op}. L^{?}_{W_f} &\xrightarrow{\tau[\perp/\diamond^{\emptyset,\perp}]} L^{\emptyset,\perp}_{W_f}
\end{aligned}$$

Now, the following steps specify one can add and remove opaque names that are part of the head-recursion

$$\begin{aligned}
\forall W_f \in \mathcal{W}_{op}, d \in \mathcal{D}, g \in I_f, V_{fg} \in \mathcal{N}_{op}. L^{\emptyset,y}_{W_f} &\xrightarrow{\tau[d/V_{fg}d]} L^{\emptyset,V_{fg}}_{W_f} \\
\forall W_f \in \mathcal{W}_{op}, d \in \mathcal{D}, g \in I_f, V_{fg} \in \mathcal{N}_{op}. L^{\chi,y}_{W_f} &\xrightarrow{\tau[d/V_{fg}\alpha^{\chi,y}d]} L^{\emptyset,V_{fg}}_{W_f} \\
\forall W_f \in \mathcal{W}_{op}, g \in I_f, V_{fg} \in \mathcal{N}_{op}. L^{\emptyset,V_{fg}}_{W_f} &\xrightarrow{\tau[V_{fg}/\varepsilon]} L^{\emptyset,?}_{W_f}
\end{aligned}$$

The latter of these brings us to $L^{\emptyset,?}_{W_f}$, which will find out what to replace the questionmark with:

$$\begin{aligned}
\forall W_f \in \mathcal{W}_{op}, Y \in \mathcal{N}_{op}. L^{\emptyset,?}_{W_f} &\xrightarrow{\tau[Y/Y]} L^{\emptyset,Y}_{W_f} \\
\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{P}(\mathcal{N}_{tr}), y \in \mathcal{N}_{op} \cup \{\perp\}. L^{\emptyset,?}_{W_f} &\xrightarrow{\tau[\alpha^{\chi,y}/\varepsilon]} L^{\chi,y}_{W_f} \\
\forall W_f \in \mathcal{W}_{op}. L^{\emptyset,?}_{W_f} &\xrightarrow{\tau[\perp/\varepsilon]} L^{\emptyset,\perp}_{W_f} \\
\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{P}(\mathcal{N}_{tr}), y \in \mathcal{N}_{op} \cup \{\perp\}. L^{\emptyset,?}_{W_f} &\xrightarrow{\tau[\diamond^{\chi,y}/\diamond^{\chi,y}]} L^{\chi,y}_{W_f}
\end{aligned}$$

The following specifications detail how to add and remove transparent names from the stack:

$$\forall W_f \in \mathcal{W}_{op}, y \in \mathcal{N}_{op} \cup \{\bot\}, d \in \mathcal{D}, g \in I_f, V_{fg} \in \mathcal{N}_{tr}. L_{W_f}^{\emptyset,y} \xrightarrow{\tau[d/V_{fg}\omega^{V_{fg}}d]} L_{W_f}^{\{V_{fg}\},y}$$

$$\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{P}(\mathcal{N}_{tr}), y \in \mathcal{N}_{op} \cup \{\bot\},$$
$$d \in \mathcal{D}, g \in I_f, V_{fg} \in \mathcal{N}_{tr}, V_{fg} \notin \chi. L_{W_f}^{\chi,y} \xrightarrow{\tau[d/V_{fg}\omega^{V_{fg}}d]} L_{W_f}^{\chi\cup\{V_{fg}\},y}$$

$$\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{P}(\mathcal{N}_{tr}), y \in \mathcal{N}_{op} \cup \{\bot\},$$
$$d \in \mathcal{D}, g \in I_f, V_{fg} \in \mathcal{N}_{tr}, V_{fg} \in \chi. L_{W_f}^{\chi,y} \xrightarrow{\tau[d/V_{fg}d]} L_{W_f}^{\chi,y}$$

$$\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{P}(\mathcal{N}_{tr}), y \in \mathcal{N}_{op} \cup \{\bot\}, g \in I_f, V_{fg} \in \chi. L_{W_f}^{\chi,y} \xrightarrow{\tau[V_{fg}/\varepsilon]} L_{W_f}^{\chi,y}$$

$$\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{P}(\mathcal{N}_{tr}), y \in \mathcal{N}_{op} \cup \{\bot\}, g \in I_f, V_{fg} \in \chi. L_{W_f}^{\chi,y} \xrightarrow{\tau[\omega^{V_{fg}}/\varepsilon]} L_{W_f}^{\chi-\{V_{fg}\},y}$$

Finally, the following set of sequential specifications detail how to move back after building the desired stack in the head-recursive part. This can only be done by executing one of the steps of W_j (as it is not transparent). For the sake of brevity, the following set of specifications is to be read as prefixed by "$\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{P}(\mathcal{N}_{tr}), y \in \mathcal{N}_{op} \cup \{\bot\}, d \in \mathcal{D}, h \in I'_f$".

$$\xi_{fh} = 1.L_{W_f}^{\emptyset,y} \xrightarrow{a_{fh}[d/d]} M^{\emptyset}$$

$$\xi_{fh} = X.L_{W_f}^{\emptyset,y} \xrightarrow{a_{fh}[d/X\omega^X d]} M^{\{X\},y}$$

$$\xi_{fh} = Y.L_{W_f}^{\emptyset,y} \xrightarrow{a_{fh}[d/Yd]} M^{\emptyset}$$

$$\xi_{fh} = XX', X \neq X'.L_{W_f}^{\emptyset,y} \xrightarrow{a_{fh}[d/X\omega^X X'\omega^{X'} d]} M^{\{X,X'\},y}$$

$$\xi_{fh} = XX', X = X'.L_{W_f}^{\emptyset,y} \xrightarrow{a_{fh}[d/XX\omega^X d]} M^{\{X\},y}$$

$$\xi_{fh} = XY.L_{W_f}^{\emptyset,y} \xrightarrow{a_{fh}[d/X\omega^X Y d]} M^{\{X\},Y}$$

$$\xi_{fh} = YX.L_{W_f}^{\emptyset,y} \xrightarrow{a_{fh}[d/Y\alpha^{\{X\},y} X\omega^X d]} M^{\emptyset}$$

$$\xi_{fh} = YY'.L_{W_f}^{\emptyset,y} \xrightarrow{a_{fh}[d/YY'd]} M^{\emptyset}$$

$$\xi_{fh} = 1.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/d]} M^{\chi,y}$$

$$\xi_{fh} = X, X \in \chi.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/Xd]} M^{\chi,y}$$

$$\xi_{fh} = X, X \notin \chi.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/X\omega^X d]} M^{\chi\cup\{X\},y}$$

$$\xi_{fh} = Y.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/Y\alpha^{\chi,y} d]} M^{\emptyset}$$

$$\xi_{fh} = XX', X, X' \in \chi.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/XX'd]} M^{\chi,y}$$

$$\xi_{fh} = XX', X \neq X', X \notin \chi, X' \in \chi.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/X\omega^X X'd]} M^{\chi\cup\{X\},y}$$

$$\xi_{fh} = XX', X \neq X', X \in \chi, X' \notin \chi.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/XX'\omega^{X'} d]} M^{\chi\cup\{X'\},y}$$

$$\xi_{fh} = XX', X \neq X', X, X' \notin \chi.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/X\omega^X X'\omega^{X'} d]} M^{\chi\cup\{X,X'\},y}$$

$$\xi_{fh} = XX', X = X', X, X' \notin \chi.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/XX\omega^X d]} M^{\chi\cup\{X\},y}$$

$$\xi_{fh} = XY.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/X\omega^X Y d]} M^{\{X\},Y}$$

$$\xi_{fh} = YX, X \in \chi.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/Y\alpha^{\chi,y} Xd]} M^{\emptyset}$$

$$\xi_{fh} = YX, X \notin \chi.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/Y\alpha^{\chi\cup\{X\},y} X\omega^X d]} M^{\emptyset}$$

$$\xi_{fh} = YY'.L_{W_f}^{\chi,y} \xrightarrow{a_{fh}[d/YY'\alpha^{\chi,y} d]} M^{\emptyset}$$

Now, the treatment for transparent head-recursive states is mostly the

same, except for the fact that here one commits to a certain action in advance (and, by extension, one commits to ending the stack in certain way, as well). Aside from this, however, it does not significantly differ from what came before. What follows is to be read as prefixed by: "$\forall W_f \in \mathcal{W}_{tr}, \chi \in \mathcal{P}(\mathcal{N}_{tr}), y \in \mathcal{N}_{op} \cup \{\bot\}, d \in \mathcal{D}$"

$$
\begin{aligned}
g \in I_f, a.\xi \in V_{fg}.M^{\chi,y} &\xrightarrow{a[X/X]} M^{\chi,y}_{W_f,\xi} \\
h \in I'_f.M^{\chi,y} &\xrightarrow{a_{fh}[X/X]} M^{\chi,y}_{W_f,\xi_{fh}} \\
M^{\chi,y}_{W_f,\xi} &\xrightarrow{\tau[W_f/\diamond^{\chi,y}]} L^{\chi,y}_{W_f,\xi} \\
g \in I_f, V_{fg} \in \mathcal{N}_{op}.L^{\emptyset,y}_{W_f,\xi} &\xrightarrow{\tau[d/V_{fg}d]} L^{\emptyset,V_{fg}}_{W_f,\xi} \\
g \in I_f, V_{fg} \in \mathcal{N}_{op}.L^{\emptyset,V_{fg}}_{W_f,\xi} &\xrightarrow{\tau[V_{fg}/\varepsilon]} L^{\emptyset,?}_{W_f,\xi} \\
g \in I_f, V_{fg} \in \mathcal{N}_{op}.L^{\chi,y}_{W_f,\xi} &\xrightarrow{\tau[d/V_{fg}\alpha^{\chi,y}d]} L^{\emptyset,V_{fg}}_{W_f,\xi} \\
Y \in \mathcal{N}_{op}.L^{\emptyset,?}_{W_f,\xi} &\xrightarrow{\tau[Y/Y]} L^{\emptyset,Y}_{W_f,\xi} \\
L^{\emptyset,?}_{W_f,\xi} &\xrightarrow{\tau[\alpha^{\chi,y}/\varepsilon]} L^{\chi,y}_{W_f,\xi} \\
L^{\emptyset,?}_{W_f,\xi} &\xrightarrow{\tau[\bot/\varepsilon]} L^{\emptyset,\bot}_{W_f,\xi} \\
L^{\emptyset,?}_{W_f,\xi} &\xrightarrow{\tau[\diamond^{\chi,y}/\diamond^{\chi,y}]} L^{\chi,y}_{W_f,\xi} \\
g \in I_f, V_{fg} \in \mathcal{N}_{tr}.L^{\emptyset,y}_{W_f,\xi} &\xrightarrow{\tau[d/V_{fg}d]} L^{\{V_{fg}\},y}_{W_f,\xi} \\
g \in I_f, V_{fg} \in \mathcal{N}_{tr}, V_{fg} \notin \chi.L^{\chi,y}_{W_f,\xi} &\xrightarrow{\tau[d/V_{fg}\omega^{V_{fg}}]} L^{\chi\cup\{V_{fg}\},y}_{W_f,\xi} \\
g \in I_f, V_{fg} \in \mathcal{N}_{tr}, V_{fg} \in \chi.L^{\chi,y}_{W_f,\xi} &\xrightarrow{\tau[d/V_{fg}d]} L^{\chi,Y}_{W_f,\xi} \\
g \in I_f, V_{fg} \in \mathcal{N}_{tr}, V_{fg} \notin \chi.L^{\chi,y}_{W_f,\xi} &\xrightarrow{\tau[V_{fg}/\varepsilon]} L^{\chi,Y}_{W_f,\xi} \\
g \in I_f, V_{fg} \in \mathcal{N}_{tr}, V_{fg} \notin \chi.L^{\chi,y}_{W_f,\xi} &\xrightarrow{\tau[\omega^{V_{fg}}/\varepsilon]} L^{\chi-\{V_{fg}\},Y}_{W_f,\xi}
\end{aligned}
$$

After this, what follows is how to step out of the head-recursive stack-creation. This is again pretty much the same as before, but with one important difference: the action linked to this moment has already be executed, and so all one has to do is correctly add the ξ that this action would normally add to the stack. Again, for the sake of brevity, the following set of specifications is to be read as prefixed by "$\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{P}(\mathcal{N}_{tr}), y \in \mathcal{N}_{op} \cup \{\bot\}, d \in \mathcal{D}, X, X' \in \mathcal{N}_{tr}, Y, Y' \in \mathcal{N}_{op}$". First we have the case in which one is at a state of the type $L^{\emptyset,y}$

$$L^{\emptyset,y}_{W_f,1} \xrightarrow{\tau[d/d]} M^{\emptyset}$$

$$L^{\emptyset,y}_{W_f,X} \xrightarrow{\tau[d/X\omega^X d]} M^{\{X\},y}$$

$$L^{\emptyset,y}_{W_f,Y} \xrightarrow{a[d/Yd]} M^{\emptyset}$$

$$X \neq X'.L^{\emptyset,y}_{W_f,XX'} \xrightarrow{\tau[d/X\omega^X X'\omega^{X'} d]} M^{\{X,X'\},y}$$

$$X = X'.L^{\emptyset,y}_{W_f,XX'} \xrightarrow{\tau[d/XX\omega^X d]} M^{\{X\},y}$$

$$L^{\emptyset,y}_{W_f,XY} \xrightarrow{\tau[d/X\omega^X Yd]} M^{\{X\},Y}$$

$$L^{\emptyset,y}_{W_f,YX} \xrightarrow{\tau[d/Y\alpha^{\{X\},y}X\omega^X d]} M^{\emptyset}$$

$$L^{\emptyset,y}_{W_f,YY'} \xrightarrow{a_{fh}[d/YY'd]} M^{\emptyset}$$

What follows is for when is in a state of type $L^{\chi,y}$ (the same prefix as before applies).

$$L^{\chi,y}_{W_f,1} \xrightarrow{\tau[d/d]} M^{\chi,y}$$

$$X \in \chi . L^{\chi,y}_{W_f,X} \xrightarrow{\tau[d/Xd]} M^{\chi,y}$$

$$X \notin \chi . L^{\chi,y}_{W_f,X} \xrightarrow{\tau[d/X\omega^X d]} M^{\chi\cup\{X\},y}$$

$$L^{\chi,y}_{W_f,Y} \xrightarrow{\tau[d/Y\alpha^{\chi,y} d]} M^{\emptyset}$$

$$X, X' \in \chi . L^{\chi,y}_{W_f,XX'} \xrightarrow{\tau[d/XX'd]} M^{\chi,y}$$

$$X \neq X', X \notin \chi, X' \in \chi . L^{\chi,y}_{W_f,XX'} \xrightarrow{\tau[d/X\omega^X X'd]} M^{\chi\cup\{X\},y}$$

$$X \neq X', X \in \chi, X' \notin \chi . L^{\chi,y}_{W_f,XX'} \xrightarrow{\tau[d/XX'\omega^{X'} d]} M^{\chi\cup\{X'\},y}$$

$$X \neq X', X, X' \notin \chi . L^{\chi,y}_{W_f,XX'} \xrightarrow{\tau[d/X\omega^X X'\omega^{X'} d]} M^{\chi\cup\{X,X'\},y}$$

$$X = X', X, X' \notin \chi . L^{\chi,y}_{W_f,XX'} \xrightarrow{\tau[d/XX\omega^X d]} M^{\chi\cup\{X\},y}$$

$$L^{\chi,y}_{W_f,XY} \xrightarrow{\tau[d/X\omega^X Y d]} M^{\{X\},Y}$$

$$X \in \chi . L^{\chi,y}_{W_f,YX} \xrightarrow{\tau[d/Y\alpha^{\chi,y} X d]} M^{\emptyset}$$

$$X \notin \chi . L^{\chi,y}_{W_f,YX} \xrightarrow{\tau[d/Y\alpha^{\chi\cup\{X\},y} X\omega^X d]} M^{\emptyset}$$

$$L^{\chi,y}_{W_f,YY'} \xrightarrow{\tau[d/YY'\alpha^{\chi,y} d]} M^{\emptyset}$$

Finally there is the case where one is in a state of the type M^{χ,W_j} where $W_j \in \mathcal{W}_{op}$. This case functions as somewhat of a hybrid between the treatment of the opaque and the transparent head-recursive names: one does immediately commit to a certain action, but this action can only be an a_{fh} from W_j itself, and not an action from any of the names W_j can freely add to the stack, since it is not transparent.

The following specifications detail how to enter the L-states in this scenario

$$\forall W_f \in \mathcal{W}_{op}, h \in I'_f . M^{\chi, W_j} \xrightarrow{a_{fh}[W_f/\varepsilon]} L^{?}_{W_f, \xi_{fh}}$$

$$\forall W_f \in \mathcal{W}_{op}, \chi \in \mathcal{N}^*_{tr}, y \in \mathcal{N}_{op} \cup \{\bot\} . L^{?}_{W_f,\xi} \xrightarrow{\tau[\alpha^{\chi,y}/\diamond^{\chi,y}]} L^{\chi,y}_{W_f,\xi}$$

$$\forall W_f \in \mathcal{W}_{op}, Y \in \mathcal{N}_{op} . L^{?}_{W_f,\xi} \xrightarrow{\tau[Y/\diamond^{\emptyset,Y}Y]} L^{\emptyset,Y}_{W_f,\xi}$$

$$\forall W_f \in \mathcal{W}_{op} . L^{?}_{W_f,\xi} \xrightarrow{\tau[\bot/\diamond^{\emptyset,\bot}]} L^{\emptyset,\bot}_{W_f,\xi}$$

Other than this slight variation this case will behave the same as the transparent head-recursive names.

As the very last step of the specification we need to tell the rest of the PDA how to deal with the new segment symbol $\diamond$. This is, like in the simple head-recursive case, quite simple: they get thrown away. So for all non-L states M in P, and for all variants of $\diamond$ we have:

$$M \xrightarrow{\tau[\diamond/\varepsilon]} M$$

Now, we also need to extend the contrasimulation (R, Q) to deal with all these new states. First we show how R is extended. We let $\zeta^{W_f} \in \{V_{fg} | g \in I_f\}^*$. Furthermore, let the following hold for all $\chi \in \mathcal{P}(\mathcal{N}_{tr}) \cup \{?\}, y \in \mathcal{N}_{tr} \cup \{\bot\} \cup \{?\}, \zeta, \zeta' \in (\mathcal{N}_{tr} \cup \mathcal{N}_{op})^*$. We also note that we will understand $L^{?,y}$ as $L^{?}$.

$$\forall W_f \in \mathcal{W}_{op} . W_f \zeta \qquad R(L^{\chi,y}_{W_f}, (\zeta^{W_f} \diamond^{\zeta^{=},\zeta^{\Rightarrow}} \zeta)^+)$$

$$\forall W_f \in \mathcal{W}_{tr}, g \in I_f, h \in I'_f, \xi \in V_{fg} \vee \xi = \xi_{fh} . \xi \zeta^{W_f} \zeta \quad R(M^{\chi,y}_{W_f,\xi}, (\zeta' W_j \zeta)^+)$$

$$\forall W_f \in \mathcal{W}_{tr}, g \in I_f, h \in I'_f, \xi \in V_{fg} \vee \xi = \xi_{fh} . \xi \zeta^{W_f} \zeta \quad R(L^{\chi,y}_{W_f,\xi}, (\zeta^{W_j} \diamond^{\zeta^{=},\zeta^{\Rightarrow}} \zeta)^+)$$

This looks slightly more complicated than it is. In the case of opaque style head-recursion it is the same as before: since the action is performed at the very end of building the desired stack, all interim stacks simply relate to the original head-recursive name (for more detailed proof of this we refer to the section on head-recursion).

With head-recursion in a transparent part of the stack the basic idea is the following: every state ζ in $\mathcal{T}_S(Z)$ relates via R to any state of the type $M^{\chi,y}_{W_f,\xi}$ and $L^{\chi,y}_{W_f,\xi}$ that can reach it's "canonical" state, that is $(M^{\zeta^{=},\zeta^{\Rightarrow}}, \zeta^+)$, using only τ-transitions. This will allow these states to mirror any step from ζ by simply doing these τ-steps, and then following the step that ζ performs as detailed in the previous proof.

Finally, the Q relation is extended as follows:

$$\forall W_f \in \mathcal{W}_{op}.(L^{\chi,y}_{W_f}, (\zeta^{W_f} \diamond^{\zeta^{=},\zeta^{\Rightarrow}} \zeta)^{+}) \qquad\qquad RW_f\zeta$$

This simply mirrors the relation for the opaque-style head-recursion, since this, as mentioned before, has not changed from the regular case, which works up to branching bisimilarity.

In terms of proving that (R, Q) is a contrasimilarity, we will refer to the proof in the previous section: head-recursion does not create any kind of interference with transparent names, and so one can still mirror steps in both directions in the same way as detailed before.

We have therefore shown that for any specification one can find a PDA such that their associated transition systems are the same up to contrasimilarity. This concludes the proof.

□

6 Transparency with Modified Process Algebras

So we saw that we can deal with both transparency and head-recursion using contrasimulation. We conjecture that it can be done for no finer equivalence in Van Glabbeek's lattice, so in particular not for branching bisimulation. We can improve this result to strong bisimulation, however, if we change the semantics of sequential composition, as indicated next.

The modification made is that we replace $\cdot$ with $\bullet$. We note that [5], where this modification is proposed, uses the symbol ; instead. Recall that for $\cdot$ the following operational rule holds:

$$\frac{P_1 \downarrow \qquad P_2 \xrightarrow{a} P_2'}{P_1 \cdot P_2 \xrightarrow{a} P_2'}$$

However, for $\bullet$ we have all rules be the same as for $\cdot$, except the above one, which becomes the following:

$$\frac{P_1 \downarrow \qquad P_2 \xrightarrow{a} P_2' \qquad P_1 \not\to}{P_1 \bullet P_2 \xrightarrow{a} P_2'}$$

Or, put into words, one can only terminate P_1 instantly and perform a step from P_2 if there are no other transitions possible from P_1.

This change has a number of advantages, as can be seen in [5]. It additionally has the disadvantage of forcing us to use a negative premise in the rule. This makes for supported models, where not every transition is derivable for the rules. For more information on the theory of negative premisses and supported models, we refer to [9].

This negative premise, in particular, causes problems when looking at head-recursion. We will first give a small example of this, to illustrate why we exclude head-recursion from the general case.

Example 6.1. Consider the following sequential specification S:

$$X = X \bullet Y + 1$$
$$Y = a.1$$

Here one seemingly runs into a problem: if one assumes that X can make no transitions, then the X in $X \bullet Y$ can terminate by choosing 1, and one could make an a-transition. However, this would contradict the assumption that X can make no transitions.

So since assuming that $X \not\to$ leads to $X \xrightarrow{a}$ (and hence to contradiction), we must have that X can execute some action, for instance $X \xrightarrow{b} 0$. This

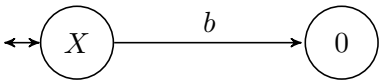

Figure 13: A possible labelled transition system $\mathcal{T}_S(X)$

means that the labelled transition system found in figure 13 satisfies the specification.

We therefore have that with head-recursion, a name can take any action, or even set of actions, as long as it reaches deadlock eventually.

However, all these transitions are not derivable from the rules. We will therefore merely look at transparent names, and leave head-recursion out.

We note that the following result has been achieved simultaneously to, and independently of [5].

Theorem 6.1. *Let S be a sequential specification defined by a set of variables $\{X_i, Y_j | i = 0, 1, ..., n \wedge j = 0, 1, ..., m\}$ as follows:*

$$X_i = \sum_{k \in I_i} a_{ik}.\xi_{ik} + 1$$
$$Y_i = \sum_{l \in I_j} a_{jl}.\xi_{jl}$$

Here ξ is a sequential composition (using $\bullet$) of any number of variables, where zero variables is understood to be equal to $\mathbf{1}$. Let Z be a name in S.

There exists a PDA P under the FS-interpretation such that $\mathcal{T}_S(Z) \leftrightarrows \mathcal{T}(P)$.

Proof. With the introduction of the modified rule, a transparent state can only terminate if all states after it are transparent as well. When constructing P it is therefore important to keep track of whether or not the stack one has is fully transparent or not. We will therefore introduce another marker symbol, $\Box$.

This marker symbol will be used to mark the last non-transparent name in a string of names. We will also introduce a piece of new notation: with $\xi^{(\Box)}$ we will mean the string ξ, but with the symbol $\Box$ added after the last non-transparent name in the string. So, for example, if $\xi = X_iY_jX_iY_j$, then we will have that $\xi^{(\Box)} = X_iY_jX_i\Box Y_j$.

Furthermore we will use the symbol ζ for any string of names, and ζ^- for the string ζ with its marker symbols removed.

To begin with, we will reorganize the specifications of the X_i's in S to be able to distinguish between ξ that contain a non-transparent name, and ones that don't.

$$X_i = \sum_{f \in I_{i(tr)}} a_{if}\beta_{if} + \sum_{h \in I_{i(op)}} a_{ih}\gamma_{ih}$$

Here β_{if} contain only transparent names, while γ_{ih} contain at least one opaque name.

The PDA P will then consist of three states. The first, N, is the initial state. This state will simulate the initial name, $Z = \sum_{f \in I_{j(tr)}} a_{jf}\beta_{jf} + \sum_{h \in I_{j(op)}} a_{jh}\gamma_{jh}$:

$$\forall f \in I_{j(tr)}.N \xrightarrow{a_{jf}[\bot/\beta_{jf}]} M$$
$$\forall h \in I_{i(op)}.N \xrightarrow{a_{jh}[\bot/\gamma_{jh}^{(\Box)}]} M'$$

Furthermore, if Z itself is transparent, we have $N \in\downarrow$.

The rest of the simulation of S is then done in the states M and M'.

As for the specification of M, this will be the state one is in as long as the entire stack is transparent. We therefore have $M \in\downarrow$, and the following specification:

$$\forall 0 \leq i \leq n, f \in I_{i(tr)}.M \xrightarrow{a_{if}[X_i/\beta_{if}]} M$$
$$\forall 0 \leq i \leq n, h \in I_{i(op)}.M \xrightarrow{a_{ih}[X_i/\gamma_{ih}^{(\Box)}]} M'$$

Note that there are no transitions for finding a Y_j on the stack. This is because this will never occur: whenever an opaque name Y_j is added to the stack, the PDA will transition to M', and it will not move back to M before all opaque names have been removed from the stack.

For M' we then finally have the following specifications:

$$\forall 0 \leq i \leq n, f \in I_{i(tr)}.M' \xrightarrow{a_{if}[X_i/\beta_{if}]} M'$$
$$\forall 0 \leq i \leq n, h \in I_{i(op)}.M' \xrightarrow{a_{ih}[X_i/\gamma_{ih}]} M'$$
$$\forall 0 \leq j \leq m, l \in I_j.M' \xrightarrow{a_{jl}[Y_j/\xi_{jl}]} M'$$
$$M' \xrightarrow{\tau[\Box/\varepsilon]} M$$

We would also like to stress the fact that $M' \notin\downarrow$.

Now, the claim is that the transition system associated with this PDA is strongly bisimilar to $\mathcal{T}_S(Z)$. We will therefore define a relation $R \subseteq \mathcal{T}_S(Z) \times \mathcal{T}(P)$, and show that it is a strong bisimulation. We define R as follows:

$$\zeta_{tr} R (M, \zeta_{tr})$$
$$\zeta_{tr} R (M', \Box\zeta_{tr})$$
$$\zeta_{op} R (M', \zeta_{op}^{(\Box)})$$
$$Z R (N, \bot)$$

Here ζ_{tr} is a string of transparent names, and ζ_{op} is a string that contains at least one opaque name.

We will now show that R is a strong bisimulation. We begin by noting that $ZR(N, \bot)$, so the initial states are related to one another. Now, let $s_1 R s_2$:

- If $s_1 \xrightarrow{a} s_1'$ then there need to exist $s_2' \in \mathcal{T}(P)$ such that $s_2 \xrightarrow{a} s_2'$, and both $s_1 R s_2''$ and $s_1' R s_2'$.

  Let $s_1 = V \cdot \zeta$. Here V is any name from S, and ζ is a string of zero or more sequentially composed names.

  Now, we first assume that $V\zeta$ contains only transparent names. We then have $V\zeta R (M, V\zeta)$. Since $V\zeta$ makes an a-transition, it must be because V contains a summand of the form $a.\zeta'$ (it cannot be a later name in $V\zeta$ due to the new rule here employed). We therefore have $V\zeta \xrightarrow{a} \zeta'\zeta$. Now, first assume that ζ' contains only transparent names. Then there exists the following transition in P:

  $$M \xrightarrow{a[V/\zeta']} M$$

  Which means that in $\mathcal{T}(P)$ we have:

  $$(M, V\zeta) \xrightarrow{a} (M, \zeta'\zeta)$$

  And $\zeta'\zeta R (M, \zeta'\zeta)$.

  Secondly, assume that ζ' contains at least one opaque name. This means that in P there exists the following transition:

  $$M \xrightarrow{a[V/\zeta'^{(\Box)}]} M'$$

  Which means that in $\mathcal{T}(P)$ we have:

  $$(M, V\zeta) \xrightarrow{a} (M', \zeta'^{(\Box)}\zeta)$$

  And $\zeta'\zeta R (M, \zeta'^{(\Box)}\zeta)$, since $\zeta'^{(\Box)}\zeta = (\zeta'\zeta)^{(\Box)}$, as ζ is fully transparent.

Lastly, consider the case in which $V\zeta$ contains at least one opaque name. This means that $V\zeta R(M', (V\zeta)^{(\Box)})$. We therefore have $V\zeta \xrightarrow{a} \zeta'\zeta$, which means that in P we must have:

$$M' \xrightarrow{a[V/\zeta']} M'$$

Which means that in $\mathcal{T}(P)$ we have:

$$(M', V\zeta) \xrightarrow{a} (M', \zeta'\zeta)$$

And $\zeta'\zeta R(M', \zeta'\zeta)$.

- If $s_2 \xrightarrow{a} s_2'$ then there need to exist $s_1' \in \mathcal{T}(P)$ such that $s_1 \xrightarrow{a} s_1'$, and both $s_1'' R s_2$ and $s_1' R s_2'$.

  Here we note once more that all transitions in the PDA directly mirror actions from S. This means that the direction of the implications of the first requirement can be reversed freely, meaning that this requirement is also met.

- If $s_1 \in\downarrow$ then there exists a s_2' such that $s_2' \in\downarrow$ and $s_1 R s_2'$.

  There are two cases in which a state in $\mathcal{T}_S(Z)$ can terminate. The first one is if $s_1 = 1$. This state (since it contains no opaque names) is related to $s_2 = (M, \bot)$, and since $M \in\downarrow$ we can therefore terminate without even needing to make τ-transitions.

  The second case is if $s_1 = \zeta$, where ζ consists only of transparent names. This means that $s_2 = (M, \zeta)$, and again, since M terminates, the requirement is instantly met.

- If $s_2 \in\downarrow$ then there exists a s_1' such that $s_1' \in\downarrow$ and $s_1 R s_2'$.

  The reverse implications of the previous point all hold, so this requirement is also met.

We can therefore conclude that R is a strong bisimulation, which concludes the proof.

□

7 Conclusion

In this section we first summarize the main results accomplished in this thesis, discuss some of their limitations, and mention some ideas for future research.

There are three primary accomplishments of this thesis. The first one is a proof of Paul van Tilburg's conjecture in [14] regarding head-recursion. We showed that given a sequential specification with head-recursion (and without transparency), one can construct a push-down automaton such that the associated transition systems are equivalent up to non-divergence-preserving branching bisimulation. This was achieved by adding a separate state in the PDA for each head-recursive name in the specification. This new PDA state can then freely add and remove elements via τ-transitions that the head-recursive name would be able to add instantaneously.

The second result is a proof for another conjecture from [14]; we proved that given a sequential specification with transparency, one can construct a PDA such that the associated transition systems are equivalent up to contrasimulation. The basic idea of this proof was to simulate "forgetful" transitions in the sequential specification (that is, steps where one or several names are "forgotten", due to them terminating instantaneously) by having non-deterministic τ-transitions.

As an extension to the previous two results, we also showed that they could be combined without any additional difficulty. That is, given any sequential specification, one can find a PDA such that the associated transition systems are equivalent up to contrasimulation.

Lastly, the third result showed that, if one modifies the operational semantics of sequential composition, one can improve the equivalence, in the case of transparency, up to strong bisimulation.

Now, with regard to the first two results, and their combination, the first relevant thing to note is that the proofs are merely constructive: we have shown that it is possible up to the specific equivalence, but we have not shown that it is impossible to do better. We do conjecture that, in the case of head-recursion, one cannot do better than non-divergence-preserving branching bisimulation, and in the case of transparency (and the general, combined case) one cannot do better than contrasimulation, but as we note that negative results of this type are usually much harder to achieve, this would be a good topic for further research.

Secondly, the third result can clearly not be improved upon, as strong bisimulation is the finest equivalence in van Glabbeek's lattice in [7]. However, it has as a downside that it introduces a negative premisse, which has as a result that in the case of head-recursion, not all transitions can be derived from the premisses.

In conclusion, we have shown in this thesis that context-free processes and push-down processes are equivalent at least up to contrasimulation.

References

[1] J.C.M. Baeten. Models of computation: Automata, formal languages, and communicating processes. Lecture Notes, February 2013.

[2] J.C.M. Baeten, T. Basten, and M.A. Reniers. *Process Algebra (Equational Theories of Communicating Processes)*. Number 50 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2009.

[3] J.C.M. Baeten, P.J.L. Cuijpers, B. Luttik, and P.J.A. van Tilburg. A process-theoretic look at automata. In F. Arbab and M. Sirjani, editors, *Proceedings FSEN'09*, number 5961 in Lecture Notes in Computer Science, pages 1–33, 2010.

[4] J.C.M. Baeten, B. Luttik, and P. van Tilburg. Computations and interaction. In R. Natarajan and A. Ojo, editors, *Proceedings ICDCIT 2011*, number 6536 in Lecture Notes in Computer Science, pages 35–54, 2011.

[5] Jos C. M. Baeten, Bas Luttik, and Fei Yang. Sequential composition in the presence of intermediate termination. *CoRR*, abs/1706.08401, 2017.

[6] T. Basten. Branching bisimilarity is an equivalence indeed! *Information Processing Letters*, 58(3):141–147, 1996.

[7] R.J. van Glabbeek. The linear time - branching time spectrum II. In *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, pages 66–81, 1993.

[8] R.J. van Glabbeek. The Linear Time – Branching Time Spectrum I. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.

[9] R.J. van Glabbeek. The meaning of negative premises in transition system specifications ii. *The Journal of Logic and Algebraic Programming*, 60-61:229–258, 2004.

[10] R.J. van Glabbeek, B. Luttik, and N. Trčka. Branching bisimilarity with explicit divergence. *Fundamenta Informaticae*, 93(4):371–392, 2009.

[11] S. A. Greibach. A new normal form theorem for context-free phrase structure grammars. *Journal of the ACM*, 12(1):42–54, 1965.

[12] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 2006.

[13] Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

[14] Paul J.A. van Tilburg. *From Computability to Executability (A Process-Theoretic View on Automata Theory)*. PhD thesis, Eindhoven University of Technology, 2011.