Safety Verification of Asynchronous Pushdown Systems with Shaped Stacks

Jonathan Kochems   C.-H. Luke Ong

University of Oxford

Abstract. In this paper, we study the program-point reachability problem of concurrent pushdown systems that communicate asynchronously via unbounded and unordered message buffers. Our goal is to relax the common restriction that messages can only be retrieved by a pushdown process when its stack is empty. We introduce a new class of asynchronously communicating pushdown systems with a mild shape constraint on the stacks, and show that the program-point coverability problem remains decidable: stacks that fit the shape may reach arbitrary heights; further a process may execute any communication action (be it process creation, message send or retrieval) whether or not its stack is empty. This class extends previous computational models studied in the context of asynchronous programs, and enables the safety verification of a large class of message passing programs.

Keywords: Concurrent pushdown systems, asynchronous message passing, verification

1 Introduction

The safety verification of concurrent and distributed systems, such as client-server environments, peer-to-peer networks and the myriad web-based applications, is an important topic of research. We consider *asynchronously communicating pushdown systems* (ACPS), a model of computation for such systems suitable for the algorithmic analysis of the reachability problem. Each process of the model is a pushdown system; processes may be spawned dynamically and they communicate asynchronously via a number of unbounded message buffers which may be ordered or unordered. In order to obtain a decision procedure for reachability, some models restrict the retrieval (or, dually, the sending) of messages or the scheduling of tasks, allowing it to take place only when the call stack is empty.

Can these restrictions on call stacks be relaxed? Unfortunately[1] some form of constraint on the call stacks in relation to the communication actions is unavoidable. Inspired by the work on asynchronous procedure calls [29, 22, 17], we consider processes that communicate asynchronously via a fixed number of unbounded and unordered message buffers which we call channels. Because channels are unordered, processes cannot observe the precise sequencing of such concurrency actions as message send and process creation; however, the sequencing of other actions, notably blocking actions such

---

[1] Any analysis that is both context-sensitive and synchronisation-sensitive is undecidable [28].

as message retrieval which requires synchronisation, is observable. If the behaviour of a process is given by its action sequences, then we may postulate that certain actions *commute* with each other (over sequential composition) while others do not. To formalise these assumptions, we make use of *partially commutative context-free grammars* (PC-CFG) [7], introduced recently by Czerwinski et al. as a study in process algebra. A PCCFG is just a context-free grammar equipped with an irreflexive symmetric relation, called *independence*, over an alphabet Σ of terminal symbols, which precisely captures the symbols that *commute* with each other. In our model, a process is described by a PCCFG that generates the set of its action sequences; terminal symbols represent concurrency and communication actions, while the non-terminal symbols represent procedure calls; and there is an induced notion of commutative procedure calls. With a view to deciding reachability, a key innovation of our work is to summarise the effects of the commutative procedure calls on the call stack. Rather than keeping track of the contents of the stack, we precompute the actions of those procedure calls that produce only commutative side-effects, and store them in caches on the call stack. The non-commutative procedure calls, which are left on the stack *in situ*, act as separators for the caches of commutative actions. As soon as the top non-commutative non-terminal on the stack is popped, which may be triggered by a concurrency action, the cache just below it is unlocked, and all the cached concurrency actions are then despatched at once.

In order to obtain a decision procedure for (a form of reachability called) *coverability*, we place a natural constraint on the shape of call stacks: at all times, no more than an *a priori* fixed number of *non-commutative non-terminals* may reside in the stack. Note that because the constraint does not apply to commutative non-terminals, call stacks can grow to arbitrary heights. Thanks to the shape constraint, we can prove that the coverability problem is decidable by an encoding into well-structured transition systems. To our knowledge, this class extends previous computational models studied in the context of asynchronous programs. Though our shape constraint is semantic, we give a simple sufficient condition which is expressed syntactically, thus enabling the safety verification of a large class of message-passing programs.

Example 1. In Figures 1 and 2 we give an example program written in a version of Erlang that employs channels (as opposed to Actor-style mailboxes), implementing a simple replicated workers pattern. It consists of a distributor process that initially spawns a number of workers, sets up a single shared resource, and distributes one task per worker over a one-to-many channel. Each worker runs a task-processing loop. Upon reception of a task, the worker recursively decomposes it, which involves communicating with the shared resource at each step. Note that the communication of each worker with the resource is protected by a lock. For the worker, the decomposition has two possible outcomes: (i) the task is partially solved, generating one subtask and an intermediate result or (ii) the task is broken down into one subtask and one new distributable task. In case (i) the worker recursively solves the subtask and combines the result with the intermediate result. In case (ii) the worker recursively solves the subtask and subsequently dispatches the newly generated distributable task before returning. When a worker has finished processing a task, it relays the result to the server and awaits a new task to

```
 1  main() → setup_network(),                21  % Resource
 2           redistribute () .               22  res_start (S) =
 3                                            23    fun() → toDistributor ! ready,
 4  setup_network() →                         24          resource(S)
 5    spawn(worker),                          25    end.
 6    case (∗) of                             26  resource(S) →
 7      true  → setup_network();              27    receive toResource:
 8      false →                               28      lock_req →
 9        spawn(res_start(init )),            29        toWorkers ! locked,
10        toResource ! isReady,              30        resource_locked(S)
11        receive toDistributor :           31    end.
12          ready → ()                        32
13        end;                                33  resource_locked(S) →
14    end, toWorkers ! task.                  34    receive toResource:
15                                            35      unlock_req → resource(S);
16  redistribute () →                         36      getState  →
17    receive toDistributor :                37        toWorkers ! state(S),
18      redist (Task) → toWorkers ! Task;    38        resource_locked(S);
19      result (Result) → print (Result);    39      update(X) → resource_locked(X)
20    end, redistribute () .                  40    end.
```

Fig. 1. A resource and a task distributor.

process. We have left the implemention of the functions decompose_task and combine open; for the purpose of this example we only assume that they do not perform any concurrency actions, but they may be recursive functions.

Note that the call stacks of both the distributor and the workers may reach arbitrary heights, and communication actions may be performed by a process at any stage of the computation, regardless of stack height. For example the worker sends and receives messages at every decomposition, and each recursive call increases the height of the call stack.

An interesting verification question for this example program is whether the locking mechanism for the shared resource guarantees exclusive access to the shared resource for each worker process in its critical section.

A Computational Model. To verify programs such as the above we need a computational model that allows us to model recursive procedure calls, message passing concurrency actions and process creation. Once the obvious abstractions are applied to make the data and message space finite, we arrive at a network of pushdown systems (equivalently context-free grammars) which can communicate asynchronously over a finite number of channels with unbounded capacity. Since we are interested in a class of such systems with decidable verification problems we assume that channels are unordered (FIFO queues with finite control are already Turing powerful [5]).

Outline. The rest of the paper is organised as follows. In Section 2 we present our model of asynchronous partially commutative pushdown systems (APCPS), its (standard) semantics and a verification problem. In Section 3 we investigate an alternative semantics for APCPS, a corresponding verification problem, and relate it to the verification problem of Section 2. In Section 4 we introduce the class of APCPS with shaped stacks and show that the verification problems are decidable for this class. In Section 5 we discuss

```
 1  worker() →                           22  decompose(Task) →
 2    receive  toWorkers:                23    lock(toResource),
 3      Task →                           24    toResource ! getState,
 4        result  = do_task(Task),       25    ?label("critical  "),
 5        toDistributor  ! result ;      26    receive  toWorkers:
 6    end, worker().                     27      state (State) →
 7                                        28        (Result,Update) =
 8  do_task(Task) →                      29          decompose_task(Task, State)
 9    case  decompose(Task) of           30    end,
10      local (Task', Int_result ) →     31    toResource ! update(Update),
11        Result = do_task(Task'),       32    unlock(toResource),
12        Result' =                      33    return  Result.
13          combine(Result,Int_result)  34
14        return  Result';               35  lock(C) →
15      redist (Task', Task'') →         36    C ! lock_req,
16        Result = do_task(Task'),       37    receive  toWorkers:
17        toDistributor  ! Redist(Task''), 38     locked → ()
18        return  Result;                39    end.
19    end.                               40  unlock(C) → C ! unlock_req.
20                                        41
21  combine(res,res')    → . . .         42  decompose_task(Task,State) → . . .
```

Fig. 2. A worker that recursively solves tasks and shares its workload.

related work and then conclude. Owing to space constraints we have relegated proofs to the appendix.

Notation. We write $\mathbb{M}[U]$ for the set of multisets over the set U, and we use $[\cdot]$ to denote multisets explicitly e.g. we write $[u, u, v, v]$ to mean the multiset containing two occurrences each of u and v. Given multisets M_1 and M_2, we write $M_1 \oplus M_2$ for the multiset union of M_1 and M_2. We write U^* for the set of finite sequences over U, and let $\alpha, \beta, \gamma, \mu, \nu, \ldots$ range over U^*. We define the *Parikh image* of $\alpha \in U^*$ to be the multiset over U, $\mathbb{M}_U(\alpha) : u \mapsto |\{i \mid \alpha(i) = u\}|$; we drop the subscript and write $\mathbb{M}(\alpha)$ whenever it is clear from the context. We order multisets in the usual way: $M_1 \leq_{\mathbb{M}} M_2$ just if for all u, $M_1(u) \leq M_2(u)$. Let $M \in \mathbb{M}[U]$ and $U_0 \subseteq U$. We define $M \restriction U_0$ to be the multiset M restricted to U_0 i.e. $(M \restriction U_0) : u \mapsto M(u)$ if $u \in U_0$, and 0 otherwise. We write $U \uplus V$ for the disjoint union of sets U and V.

2   Asynchronous Communicating Pushdown Systems

In this section we introduce our model of concurrency, *asynchronous partially commutative pushdown systems*. Processes are modelled by a variant of context-free grammars, which distinguish commutative and non-commutative concurrency actions. Communication between processes is asynchronous, via a fixed number of unbounded and unordered message buffers, which we call *channels*.

Preliminaries. An *independence relation* I over a set U is a symmetric irreflexive relation over U. It induces a congruence relation $\simeq_I$ on U^* defined as the least equivalence relation R containing I and satisfying: $(\mu, \mu') \in R \Rightarrow \forall \nu_0, \nu_1 \in U^*$: $(\nu_0 \, \mu \, \nu_1, \nu_0 \, \mu' \, \nu_1) \in R$.

Let I be an independence relation over U. An element $a \in U$ is *non-commutative* (with respect to I) just if $\forall b \in U : (a, b) \notin I$ i.e. a does not commute with any other element. An element b is *commutative* (with respect to I) just if for each $c \in U$, if c is not non-commutative then $(c, b) \in I$; intuitively it means that b commutes with all elements of U except those that are non-commutative. We call an independence relation I *unambiguous* if just every element of U is either commutative or non-commutative.

Definition 1. Let Σ be an alphabet of terminal symbols and $I \subseteq \Sigma \times \Sigma$ an independence relation over Σ. A *partially commutative context-free grammar* (PCCFG) is a quintuple $\mathcal{G} = (\Sigma, I, \mathcal{N}, \mathcal{R}, S)$ where $S \in \mathcal{N}$ is a distinguished start symbol, and $\mathcal{R}$ is a set of rewrite rules of the following types:[2] let $A \in \mathcal{N}$ (i) $A \to a$ where $a \in \Sigma \cup \{\epsilon\}$, (ii) $A \to a\,B$ where $a \in \Sigma$, $B \in \mathcal{N}$, (iii) $A \to B\,C$ where $B, C \in \mathcal{N}$. We refer to each $\rho \in \mathcal{R}$ as a $\mathcal{G}$-*rule*.

The (leftmost) derivation relation $\to_{\text{seq}}$ is a binary relation over $(\Sigma \cup \mathcal{N})^* /_{\simeq_I}$ defined as $X\,\alpha \to_{\text{seq}} \beta\,\alpha$ if $X \to \beta$ is a $\mathcal{G}$-rule. Note the derivation relation is defined over the quotient by $\simeq_I$, so the words generated are congruence classes induced by $\simeq_I$. As usual we denote the n-step relation as $\to_{\text{seq}}^n$ and reflexive, transitive closure as $\to_{\text{seq}}^*$.

We further define a k-index derivation to be a derivation in which every term contains at most k occurrences of non-terminals. Recent work [14, 12] has shown that for every commutative context-free grammar $\mathcal{G}$ there exists $k \geq 1$ such that the entire language of $\mathcal{G}$ can be generated by derivations of index k.

PCCFG was introduced by Czerwinski et al. as a study in process algebra. They investigated [7] the decidability of bisimulation for a class of processes described by PCCFG where the commutativity of the sequential composition is constrained by an independence relation on non-terminals. We propose to use words generated by PCCFGs to represent the sequence of concurrency actions of processes.

2.1 Asynchronous Partially Commutative Pushdown Systems

Our model of computation, asynchronous partially commutative pushdown systems, are in essence PCCFGs equipped with an independence relation over an alphabet Σ of terminal symbols, which represent the concurrency actions and program point labels. First some notation. Let $Chan$ be a finite set of *channel names* ranged over by c, Msg be a finite *message alphabet* ranged over by m, and $\mathcal{L}$ be a finite set of *program point labels* ranged over by l, l', l_1, etc. Further let $\mathcal{N}$ be a finite set of non-terminal symbols. We derive an alphabet Σ of terminal symbols

$$\Sigma := \mathcal{L} \cup \{c\,!\,m, c\,?\,m \mid c \in Chan, m \in Msg\} \cup \{\nu X \mid X \in \mathcal{N}\}. \qquad (1)$$

An action of the form $c\,!\,m$ denotes the sending of the message m to channel c, $c\,?\,m$ denotes the retrieval of message m from channel c, and νX denotes the spawning of a new process that begins execution from X. We will use a, a', b, etc. to range over Σ.

---

[2] Identifying rules of type (ii), which is a special case of type (iii), allows us to distinguish tail-recursive and non-tail recursive calls, which will be handled differently in the sequel, beginning with Definition **??**

Our computational model will emit program point labels in its computation, allowing us to pose questions of reachability. We will now define the computational power of our processes in terms of PCCFGs.

The words that are generated by a process *qua* PCCFG represent its action sequences. Because channels are unordered, processes will not be able to observe the precise sequencing of concurrency actions such as message send and process creation; however the sequencing of other actions such as message retrieval is observable. Using the language of partially commutative context-free grammar, we can make this sensitivity to sequencing precise by an independence relation on actions.

An Independence Relation for the Concurrency Actions. Let $\Xi \subseteq \Sigma$, we define the independence relation over Σ generated by Ξ as

$$IndRel_\Sigma(\Xi) := \{(a, a'), (a', a) \mid a, a' \in \Xi, a \neq a'\}$$

Now let $\Sigma^\flat := \mathcal{L} \cup \{c \mathbin{!} m \mid c \in Chan, m \in Msg\} \cup \{\nu X \mid X \in \mathcal{N}\}$ be the subset of Σ consisting of the program point labels and the send and spawn actions. It is straightforward to see that $IndRel_\Sigma(\Sigma^\flat)$ is, by construction, an unambiguous independence relation over Σ. Thus $IndRel_\Sigma(\Sigma^\flat)$ allows us to commute all concurrency actions *except* receive. Further we allow program point labels to commute. This is harmless, since our goal is to analyse (a form of) control-state reachability, i.e. the question whether a particular label can be reached, as opposed to questions that require sequential reasoning such as whether label l_1 will be reached before l_2 is reached.

We can now lift the independence relation to the non-terminals of a PCCFG $\mathcal{G}$. Let I be the least subset of $(\mathcal{N} \cup \Sigma)^2$ such that (i) $IndRel_\Sigma(\Sigma^\flat) \subseteq I$, and (ii) for all $b \in \Sigma \cup \mathcal{N}$ and $A \in \mathcal{N}$, if $\forall a \in \mathrm{RHS}(A) : (a, b) \in I$ then $\{(A, b), (b, A)\} \subseteq I$, where $\mathrm{RHS}(A) := \{a \in \mathcal{N} \cup \Sigma \mid A \to \alpha \in \mathcal{G}, a \text{ occurs in } \alpha\}$. We note that I, which is well-defined, is an unambiguous independence relation over $\mathcal{N} \cup \Sigma$. Thus we can partition both Σ and $\mathcal{N}$ into Σ^{com} and $\mathcal{N}^{\mathrm{com}}$, the *commutative* actions and non-terminals respectively, and $\Sigma^{\neg\mathrm{com}}$ and $\mathcal{N}^{\neg\mathrm{com}}$ their *non-commutative* counterparts respectively.

We can now define our model of computation.

Definition 2. Assume $\mathcal{L}$, $Chan$, Msg and $\mathcal{N}$ as introduced earlier, and the derived alphabet Σ of terminals as defined in (1). An *asynchronous partially commutative pushdown system* (APCPS) is just a PCCFG $\mathcal{G} = (\Sigma, I, \mathcal{N}, \mathcal{R}, S)$.

Henceforth we fix $\mathcal{L}$, $Chan$, Msg and $\mathcal{N}$, and the derived (1) alphabet Σ of terminals.

2.2   Standard Semantics

The operational semantics is given as a transition system. A configuration of the system is a pair, consisting of a parallel composition of processes and a set of channels. We represent the state of a single process as an element of $Control := (\Sigma \cup \mathcal{N})^* / {\simeq_I}$. The derivation relation of PCCFGs, $\to_{\mathrm{seq}}$, defines how processes make *sequential* transitions.

Next we define how processes interact concurrently by message passing via a fixed set of unbounded and unordered channels.

Definition 3 (Standard Concurrent Semantics). The *configurations* are elements of $\mathbb{M}[Control] \times (Chan \rightarrow \mathbb{M}[Msg])$. For simplicity, we write a configuration (say) $([\alpha, \beta], \{c_1 \mapsto [m_a, m_b, m_b], c_2 \mapsto []\})$ as $\alpha \parallel \beta \lhd [m_a, m_b, m_b]^{c_1}, []^{c_2}$. We abbreviate a set of processes running in parallel as Π and a set of channels by Γ with names in $Chan$. The operational semantics for APCPS, a binary relation $\rightarrow_{\text{con}}$ over configurations, is then defined by induction over the rule:

$$\frac{\alpha \rightarrow_{\text{seq}} \alpha'}{\alpha \parallel \Pi \lhd \Gamma \rightarrow_{\text{con}} \alpha' \parallel \Pi \lhd \Gamma} \tag{2}$$

and the following axioms: let $m \in Msg, c \in Chan, l \in \mathcal{L}$ and $X \in \mathcal{N}$

$$(c\,?\,m)\,\alpha \parallel \Pi \lhd ([m] \oplus q)^c, \Gamma \rightarrow_{\text{con}} \alpha \parallel \Pi \lhd q^c, \Gamma \tag{3}$$

$$(c\,!\,m)\,\alpha \parallel \Pi \lhd q^c, \Gamma \rightarrow_{\text{con}} \alpha \parallel \Pi \lhd ([m] \oplus q)^c, \Gamma \tag{4}$$

$$l\,\alpha \parallel \Pi \lhd \Gamma \rightarrow_{\text{con}} \alpha \parallel \Pi \lhd \Gamma \tag{5}$$

$$(\nu X)\,\alpha \parallel \Pi \lhd \Gamma \rightarrow_{\text{con}} \alpha \parallel X \parallel \Pi \lhd \Gamma. \tag{6}$$

The *start configuration* is $S \lhd \emptyset$. We define a partial order on configurations as follows: $\Pi \lhd \Gamma \leq \Pi' \lhd \Gamma'$ just if $\Pi \leq_{\mathbb{M}} \Pi'$ and for every $c \in Chan$, $\Gamma(c) \leq_{\mathbb{M}} \Gamma'(c)$.

2.3 Program-Point Coverability

In the sequential setting of (ordinary) pushdown systems, the control-state reachability problem is of central interest. In our notation, it asks, given a control-state A, if it is possible to reach a process-configuration $A\,\alpha$ where A is the control-state and α is some call stack. It should be clear that an equivalent problem is to ask whether $l\,\alpha$ is reachable, where l is a program-point label. We prefer a formulation that uses program-point labels because it simplifies our argument (and is equi-expressive).

In the concurrent setting, we wish to know whether, given a APCPS and program-point labels $l_1, \ldots, l_n$, there exist call stacks $\alpha_1, \ldots, \alpha_n$ and channel contents Γ such that the configuration $l_1\,\alpha_1 \parallel \cdots \parallel l_n\,\alpha_n \lhd \Gamma$ is $\rightarrow_{\text{con}}$-reachable, possibly in parallel with some other processes. Note that this question allows us to express not just control-state reachability queries but also mutual exclusion properties. We state the problem of program-point coverability more formally as follows.

Verification Problem 1 (Program-Point Coverability)**.** Given an APCPS $\mathcal{G}$ and a set of program point labels $l_1, \ldots, l_n$, a tuple $(\mathcal{G}; l_1, \ldots, l_n)$ is a yes-instance of the *program-point coverability* problem just if there exist a configuration $\Pi \lhd \Gamma$ and $\alpha_1, \ldots, \alpha_n \in (\Sigma \cup \mathcal{N})^*/_{\simeq_I}$ such that $\Pi \lhd \Gamma$ is $\rightarrow_{\text{con}}$-reachable and $l_1\,\alpha_1 \parallel \cdots \parallel l_n\,\alpha_n \lhd \emptyset \leq \Pi \lhd \Gamma$.

The program-point coverability problem allows us to characterise "bad-configurations" c_{bad} in terms of program-point labels. We regard a configuration c that covers c_{bad}, in the sense that ($c_{\text{bad}} \leq c$), also as "bad". Using program-point coverability, we can express whether any such configuration is reachable

Example 2. Consider the program in Figures 1 and 2 and call it P. The problem of whether each worker has exclusive access to the shared resource in its critical section is expressible as a program-point coverability problem. A bad configuration is one in which two worker processes are executing the line marked by ?label(" critical "). We can thus see that $(P;?\text{label(" critical ")},?\text{label(" critical ")})$ is an instance of the program-point coverability problem; a no answer implies mutual exclusion, whereas a yes answer would tell us that two worker processes can be simultaneously within their critical section.

The program-point coverability problem is undecidable for unconstrained APCPS. In fact APCPS is Turing powerful: it is straightforward to simulate a system with two synchronising pushdown systems.

3   An Alternative Semantics for APCPS

In this section we present an alternative semantics for APCPS which captures enough information to solve the program-point coverability problem. The key idea is to summarise the effects of the commutative non-terminals on the call stack. In the alternative semantics, rather than keeping track of the contents of the call stack, we precompute the actions of those procedure calls that produce only *commutative* side-effects, i.e. sends, spawns and program point labels, and store them in caches on the call stack. The non-commutative procedure calls, which are left on the call stack, then act as separators for the caches of commutative side-effects. As soon as the top non-commutative non-terminal on the stack is popped, which may be triggered by a concurrency action, the cache just below it is unlocked. The cached actions are made effective instantaneously. Note that this is enough to ensure a precise correspondence between the program-point coverability problem for APCPS and a corresponding coverability problem for our alternative semantics.

An Alternative Semantics. First we introduce a representation of the states of a process. Let $k \in \mathbb{N} \cup \{\infty\}$.

$$TermCache := \mathbb{M}[\Sigma^{\text{com}}] \qquad MixedCache := \mathbb{M}[\Sigma^{\text{com}} \cup \mathcal{N}^{\text{com}}]$$
$$NonTermCache := \mathbb{M}[\mathcal{N}^{\text{com}}] \qquad Cache := TermCache \uplus MixedCache$$
$$CallStack^{\leq k} := (\mathcal{N}^{\neg\text{com}} \cdot Cache)^{\leq k}$$
$$DelayedControl := TermCache \uplus MixedCache \uplus NonTermCache$$
$$NormalControl := (\mathcal{N} \cdot Cache) \uplus (\Sigma \cdot \mathcal{N} \cdot Cache) \uplus (\Sigma \cdot Cache)$$
$$ControlState := NormalControl \uplus DelayedControl$$
$$\gamma, \delta \in Control^{\leq k} := ControlState \cdot CallStack^{\leq k}$$
$$Queue := \mathbb{M}[Msg] \qquad Queues := Chan \rightarrow Queue$$
$$Config^{\leq k} := \mathbb{M}\left[Control^{\leq k}\right] \times Queues$$

Note that we assume the equality $\epsilon = \emptyset$ to simplify notation. We write $Control^{\mathbb{M}} := Control^{\leq \infty}$ and $CallStack^{\mathbb{M}} := CallStack^{\leq \infty}$. [3]

Definition 4 (Alternative Sequential Semantics). Let $\mathcal{G}$ be a PCCFG. We define a transition relation $\rightarrow_{\mathrm{seq}'} \subseteq Control^{\mathbb{M}} \times Control^{\mathbb{M}}$ by induction over the following rules:

If $A \rightarrow B\,C$ is a $\mathcal{G}$-rule, C commutative and $C \rightarrow^*_{\mathrm{seq}} w \in (\mathcal{N}^{\mathrm{com}} \cup \Sigma^{\mathrm{com}})^*$ then

$$A\,M\,\gamma \rightarrow_{\mathrm{seq}'} B\,(\mathbb{M}(w) \oplus M)\,\gamma \tag{7}$$

If $A \rightarrow B\,C$ is a $\mathcal{G}$-rule and C non-commutative then

$$A\,M\,\gamma \rightarrow_{\mathrm{seq}'} B\,C\,M\,\gamma \tag{8}$$

If $A \rightarrow a\,B$ is a $\mathcal{G}$-rule and $a \in \Sigma$ and $B \in \mathcal{N}$ then

$$A\,M\,\gamma \rightarrow_{\mathrm{seq}'} a\,B\,M\,\gamma \tag{9}$$

If $A \rightarrow a$ is a $\mathcal{G}$-rule where $a \in \Sigma \cup \{\epsilon\}$ then

$$A\,M\,\gamma \rightarrow_{\mathrm{seq}'} a\,M\,\gamma \tag{10}$$

where $\gamma \in CallStack^{\mathbb{M}}$, $M \in Cache$, and A, B and C range over non-terminals.

From the alternative sequential semantics, we derive a corresponding alternative concurrent semantics, using the following notation: for $M \in \mathbb{M}[\Sigma^{\mathrm{com}}]$ and $w \in (\Sigma^{\mathrm{com}})^*$

$$\Gamma \oplus \Gamma' := \{c \mapsto \Gamma(c) \oplus \Gamma'(c) \mid c \in Chan\}$$
$$\Gamma(M) := \{c \mapsto \textstyle\sum_{c!m \in M} M(c\,!\,m) \mid c \in Chan\} \quad \Gamma(w) := \Gamma(\mathbb{M}(w))$$
$$\Pi(M) := \{X \mapsto M(\nu X) \mid X \in \mathcal{N}\} \qquad\qquad \Pi(w) := \Pi(\mathbb{M}(w))$$

Definition 5 (Alternative Concurrent Semantics). We define a binary relation $\rightarrow_{\mathrm{con}'}$ over $\mathbb{M}[Control^{\mathbb{M}}] \times (Chan \rightarrow \mathbb{M}[Msg])$ by induction over the following rules:

If $\gamma \in NormalControl \cdot CallStack^{\mathbb{M}}$, $\gamma \rightarrow_{\mathrm{seq}'} \gamma'$ then

$$\gamma \parallel \Pi \lhd \Gamma \rightarrow_{\mathrm{con}'} \gamma' \parallel \Pi \lhd \Gamma \tag{11}$$

If $(c\,?\,m)\,\gamma \in NormalControl \cdot CallStack^{\mathbb{M}}$, $m \in Msg$ then

$$(c\,?\,m)\,\gamma \parallel \Pi \lhd ([m] \oplus q)^c, \Gamma \rightarrow_{\mathrm{con}'} \gamma \parallel \Pi \lhd q^c, \Gamma \tag{12}$$

If $X \in \mathcal{N}$, $(\nu X)\,\gamma \in NormalControl \cdot CallStack^{\mathbb{M}}$ then

$$(\nu X)\,\gamma \parallel \Pi \lhd \Gamma \rightarrow_{\mathrm{con}'} \gamma \parallel X \parallel \Pi \lhd \Gamma \tag{13}$$

If $(c\,!\,m)\,\gamma \in NormalControl \cdot CallStack^{\mathbb{M}}$, $m \in Msg$ then

$$(c\,!\,m)\,\gamma \parallel \Pi \lhd q^c, \Gamma \rightarrow_{\mathrm{con}'} \gamma \parallel \Pi \lhd ([m] \oplus q)^c, \Gamma \tag{14}$$

If $l\,\gamma \in NormalControl \cdot CallStack^{\mathbb{M}}$, $l \in \mathcal{L}$ then

$$l\,\gamma \parallel \Pi \lhd \Gamma \rightarrow_{\mathrm{con}'} \gamma \parallel \Pi \lhd \Gamma \tag{15}$$

---

[3] Defining $Cache$ as a distjoint union enables a definition by cases according to the type of cache, thus rendering $\rightarrow_{\mathrm{con}'}$ monotone with respect to an ordering.

If $M \, X \, \gamma \in DelayedControl \cdot CallStack^{\mathbb{M}}$, $M \in TermCache$, $\Gamma' = \Gamma \oplus \Gamma(M)$, $\Pi' = \Pi \oplus \Pi(M)$ then

$$M \, X \, \gamma \parallel \Pi \lhd \Gamma \rightarrow_{\mathrm{con'}} X \, \gamma \parallel \Pi' \lhd \Gamma' \tag{16}$$

If $M \, \gamma \in DelayedControl \cdot CallStack^{\mathbb{M}}$, $M \in MixedCache$, $\Gamma' = \Gamma \oplus \Gamma(M)$, $\Pi' = \Pi \oplus \Pi(M)$ and $M' = M \upharpoonright (\mathcal{N}^{\mathrm{com}} \cup \mathcal{L})$ then

$$M \, \gamma \parallel \Pi \lhd \Gamma \rightarrow_{\mathrm{con'}} M' \gamma \parallel \Pi' \lhd \Gamma' \tag{17}$$

The alternative semantics precomputes the actions of commutative non-terminals on the call stacks. This is achieved by rule (7) in the alternative sequential semantics. The rules (16) and (17) are the concurrent counterparts; they ensure that the precomputed actions are rendered effective at the appropriate moment. Rule (16) is applicable when the precomputed cache M contains exclusively commutative actions; such a cache denotes a sequence of commutative non-terminals whose computation terminates and generates concurrency actions. Rule (17), on the other hand, handles the case where the cache M contains non-terminals. An interpretation of such a cache is a partial computation of a sequence of commutative non-terminals. In this case rule (17) dispatches all commutative actions and then blocks. It is necessary to consider this case since not all non-terminals have terminating computations. Thus rule (7) may non-deterministically decide to abandon the pre-computation of actions.

We now give a variant of the program-point coverability problem tailored to the alternative semantics and show its correspondence with the program-point coverabilily problem of Section 2.

Verification Problem 2 (Alternative Program-Point Coverability)**.** Given an APCPS $\mathcal{G}$ and a set of program point labels $l_1, \ldots, l_n$, a tuple $(P; l_1, ..., l_n)$ is a yes-instance of the *alternative program-point coverability* problem just if there exist a $\rightarrow_{\mathrm{con'}}$-reachable configuration $\Pi \lhd \Gamma$ such that for every $i \in \{1, \ldots, n\}$ there exists $\lambda_i \, \gamma_i \in \Pi$ such that either $\lambda_i = l_i$, or $\lambda_i = M_i$ and $l_i \in M_i$?

In the appendix we show that the standard semantics *weakly simulates* the alternative semantics for APCPS (Proposition 4). Thus for every configuration reachable in the alternative semantics there is a corresponding configuration reachable in the standard semantics. Owing to the nature of precomputations and caches, it is more difficult to relate runs of the standard semantics to those of the alternative semantics. However, in the appendix, we show that for every run in the standard semantics reaching a configuration, there exists a run in the alternative semantics reaching a corresponding configuration (Proposition 2).

Theorem 1 (Reduction of Program-Point Coverability). *A tuple $(P; l_1, \ldots, l_n)$ is a yes-instance of the program-point coverabililty problem if, and only if, $(P; l_1, \ldots, l_n)$ is a yes-instance of the alternative program-point coverability problem.*

4 APCPS with Shaped Stacks

In this section we present a natural restriction on the shape of the call stacks of APCPS processes. This shape restriction says that, at all times, at most an *a priori* fixed number

of non-commutative non-terminals may reside in the call stack. Because the restriction does not apply to commutative non-terminals, call stacks can grow to arbitrary heights. We show that the alternative semantics for such shape-constrained APCPS gives rise to a well-structured transition system, thus allowing us to show the decidability of the alternative program-point coverability problem.

Definition 6. Define $Reach_{\to_{\mathrm{con'}}} := \{\Pi \lhd \Gamma \mid [S] \lhd \emptyset \to^*_{\mathrm{con'}} \Pi \lhd \Gamma\}$. Let $k \in \mathbb{N}$, we say an APCPS $\mathcal{G}$ *has k-shaped stacks* just if $Reach_{\to_{\mathrm{con'}}} \subseteq Config^{\leq k}$. An APCPS $\mathcal{G}$ *has shaped stacks* just if $\mathcal{G}$ has k-shaped stacks for some $k \in \mathbb{N}$.

It follows from the definition that, in the alternative semantics, processes of an APCPS with k-shaped stacks have the form: $\gamma \, X_1 \, M_1 \, X_2 \, M_2 \cdots X_j \, M_j$ where $\gamma \in ControlState$, $X_i \in \mathcal{N}^{\neg\mathrm{com}}$ and $j \leq k$. Relating this to the standard semantics, processes of an APCPS with k-shaped stacks are always of the form $\alpha \, X_1 \, \beta_1 \, X_2 \, \beta_2 \cdots X_j \, \beta_j$ where $\alpha \in (\mathcal{N} \cup (\Sigma \cdot \mathcal{N}) \cup (\Sigma \cup \{\epsilon\})) \cdot \mathcal{N}^{\mathrm{com}*}$ and $\beta_i \in \mathcal{N}^{\mathrm{com}*}$. It is this shape that lends itself to the name APCPS. Even though the shaped stacks constraint is semantic, the following proposition gives a *syntactic* sufficient condition. (The simple proof is omitted.)

Proposition 1. *Let $\mathcal{G}$ be an APCPS. If there is a well-founded partial order $\geq_{shape}$ such that for every $A \in \mathcal{N}$ and $B \in RHS(A) \cap \mathcal{N}$: (i) $A \geq_{shape} B$, and (ii) $\exists C \in \mathcal{N}^{\neg com} :$ $A \to B\,C$ is a $\mathcal{G}$-rule $\Rightarrow A >_{shape} B$, then $\mathcal{G}$ has shaped stacks.*

Example 3. Proposition 1 tells us that the program in Figures 1 and 2 can be modelled by an APCPS with shaped stacks. Non-tail recursive calls are potentially problematic. In our example the recursive call to setup_network() in the definition of setup_network is non-tail recursive, but only places a send action on the call stack, thus causing no harm. The only other non-tail recursive calls occur in do_task: the call to decompose_task which poses no threat since decompose_task does not invoke do_task again, and the two recursive calls to do_task which either place procedure calls without concurrent actions or send actions on the stack.

4.1   APCPS with Shaped Stacks and Well-Structured Transition Systems

We will now show the decidability of the alternative program-point coverability problem for APCPS with shaped stacks. First we recall the definition of well-structured transition systems [15]. Let $\leq$ be an ordering over a set U; we say $\leq$ is a *well-quasi-order* (wqo) just if for all infinite sequences $u_1, u_2, \ldots$ there exists i, j such that $u_i \leq u_j$. A *well-structured transition system* (WSTS) is a quadruple $(S, \to, \leq, s_0)$ such that $s_0 \in S$, $\leq$ is a wqo over S and $\to \subseteq S \times S$ is monotone with respect to $\leq$, i.e. if $s \to s'$ and $s \leq t$ then there exists t' such that $t \to t'$.

Well-structured transitions systems are an expressive class of infinite state systems that enjoy good model checking properties. A decision problem for WSTS of particular interest to verification is the *coverability problem* i.e. given a state s is it the case that $s_0 \to^* s'$ and $s \leq s'$. For $U \subseteq S$ define the sets $Pred(U) := \{s \mid s \to u, u \in U\}$ and $\uparrow U := \{u' \mid u \leq u', u \in U\}$. For WSTS the coverability problem is decidable [15] provided that for any given $s \in S$ the set $\uparrow Pred(\uparrow \{s\})$ is effectively computable.

Well-quasi-orders can be composed in various ways which makes decision results for WSTS applicable to a wide variety of infinite state models. In the following we recall a few results on the composition of wqos.

(WQO-a) If $(A_i, \leq_i)$ are wqo sets for $i = 1, ..., k$ then $(A_1 \times \cdots \times A_k, \leq_1 \times \cdots \times \leq_k)$ is a wqo set. (*Dickson's Lemma*)

(WQO-b) If A is a finite set then $(A, =)$ is a wqo set.

(WQO-c) If $(A, \leq)$ is a wqo then $(\mathbb{M}[A], \leq_{\mathbb{M}[A]})$ is a wqo set where $M_1 \leq_{\mathbb{M}[A]} M_2$ just if for all $a \in A$ there exists an $a' \geq a$ such that $M_1(a) \leq M_2(a')$ [32].

(WQO-d) If $(A, \leq_A)$ and $(B, \leq_B)$ are wqo sets, then $(A \cdot B, \leq_A \cdot \leq_B)$ is a wqo set, where $\gamma \cdot \gamma' \leq_A \cdot \leq_B \delta \cdot \delta$ just if $\gamma \leq_A \delta$ and $\gamma' \leq_B \delta'$.

(WQO-e) If $(A, \leq_A)$ and $(B, \leq_B)$ are wqo set, then $(A \uplus B, \leq_A \uplus \leq_B)$ is a wqo set, where $a \leq_A \uplus \leq_B b$ just if $a, b \in A$ and $a \leq_A b$ or $a, b \in B$ and $a \leq_B b$.

4.2   A Well-Quasi-Order for the Alternative Semantics

Fix a k. Our goal is to construct a well-quasi-order for $Config^{\leq k}$ which is the first step to showing the alternative semantics gives rise to a WSTS for APCPS with shaped stacks.

We can order the multi-sets $TermCache, NonTermCache, MixedCache$ and $Queue$ with the multi-set inclusion $\leq_{\mathbb{M}}$ which is a well-quasi-order. Since $Chan$ is a finite set and $Queues = Chan \to \mathbb{M}[Msg] \cong \mathbb{M}[Msg]^{|Chan|}$ we obtain a well-quasi-order for $Chan \to \mathbb{M}[Msg]$ using a generalisation of Dickson's lemma. We can then compose the wqo of $TermCache$ and $MixedCache$ to obtain a wqo $\leq_{Cache} := \leq_{TermCache} \uplus \leq_{MixedCache}$ for $Cache$. We order $CallStack^{\leq k}$ as follows: for each $j \in \{1 \ldots k\}$ we define

$$X_1\, M_1\, X_2\, M_2 \cdots X_j\, M_j \leq X_1\, M_1'\, X_2\, M_2' \cdots X_j\, M_j' \quad \text{iff} \quad \forall i : M_i \leq_{Cache} M_i'$$

which gives a well-quasi-order. We obtain a wqo for $DelayedControl$ by composing the wqos of $TermCache, NonTermCache$ and $MixedCache$:

$$\leq_{DelayedControl} := \leq_{TermCache} \uplus \leq_{NonTermCache} \uplus \leq_{MixedCache} \cdot$$

Since Σ and $\mathcal{N}$ are finite sets, $(\Sigma, =_\Sigma)$ and $(\mathcal{N}, =_\mathcal{N})$ are wqo sets, and so, we can compose a wqo for $NormalControl$:

$$\leq_{NormalControl} := (=_\Sigma \cdot \leq_{Cache}) \uplus (=_\Sigma \cdot =_\mathcal{N} \cdot \leq_{Cache}) \uplus (=_\mathcal{N} \cdot \leq_{Cache}) \cdot$$

Similarly we can construct wqos for $ControlState$ and $Control^{\leq k}$ by composition:

$$\leq_{ControlState} := \leq_{NormalControl} \uplus \leq_{DelayedControl}$$

$$\leq_{Control^{\leq k}} := \leq_{ControlState} \cdot \leq_{CallStack^{\leq k}} \cdot$$

As a last step we use (WQO-c) to construct a wqo for $\mathbb{M}\left[Control^{\leq k}\right]$ which then allows us to define a wqo for $Config^{\leq k}$ by $\leq_{Config^{\leq k}} := \leq_{\mathbb{M}[Control^{\leq k}]} \times \leq_{Queues} \cdot$

To prove the decidability of the coverability problem for APCPS with shaped stacks, it remains to show that $\to_{con'}$ is monotonic and $\uparrow Pred(\uparrow \{\gamma\})$ is computable.

Lemma 1 (Monotonicity). *The transition relation $\rightarrow_{con'}$ is monotone with respect to the well-order $\leq_{Config^{\leq k}}$.*

Corollary 1. *The transition system $\left(Config^{\leq k}, \rightarrow_{con'}, \leq_{Config^{\leq k}} \right)$ is a well-structured transition system.*

To see that $\uparrow Pred(\uparrow \{\gamma\})$ is computable is mostly trivial; only predecessors generated by rule (7) are not immediately obvious. Given $M' \in Cache$ we observe that it is enough to be able to compute the set $P_{M'} := \uparrow \{(C, M) \mid C \in \mathcal{N}^{com}, C \rightarrow^*_{seq} w, M'' = M \oplus \mathbb{M}(w), M' \leq_{\mathbb{M}} M''\}$. Now $C \rightarrow^*_{seq} w$ is a computation of a commutative context-free grammar (CCFGs) for which an encoding into Petri nets has been shown by Ganty and Majumdar [17]. Their encoding builds on work by Esparza [11] modelling CCFG in Petri nets. Their translation leverages a recent result [14]: every word of a CCFG has a *bounded-index* derivation i.e. every term of the derivation uses no more than an *a priori* fixed number of occurrences of non-terminals. A budget counter constrains the Petri net encoding of a CCFG to respect boundedness of index; termination of a CCFG computation can be detected by a transition that is only enabled when the full budget is available. This result allows us to compute the set $P_{M'}$ using a backwards coverability algorithm for Petri nets.

Theorem 2. *The alternative program-point coverability problem, and hence the program-point coverability problem, for APCPS with k-shaped stacks are decidable for every $k \geq 0$.*

5   Related Work and Discussion

Partially Commutative Context-Free Grammars (PCCFG). Czerwinski et al. introduced PCCFG as a study in process algebra [7]. They proved that bisimulation is NP-complete for a class of processes extending BPA and BPP [11] where the sequential composition of certain processes is commutative. Bisimulation is defined on the traces of such processes, although there is no synchronisation between processes. In [8] the problem of word reachability for partially commutative context-free languages was shown to be NP-complete.

Asynchronous Procedure Calls. Petri net models for finite state machines that communicate asynchronously via unordered message buffers were first investigated by Mukund et al. [26, 27]. In an influential paper [29] in 2006, Sen and Viswanathan showed that safety verification is decidable for first-order programs with atomic asynchronous methods. Building on this, Jhala and Majumdar [22] constructed a VAS that models such asynchronous programs on-the-fly. Liveness properties, such as fair termination and starvation, of asynchronous programs were extensively studied by Ganty et al. in [18, 17]. In our more general APCPS framework, we may view the asynchronous programs considered by Ganty and Majumdar in [17] as APCPS running a single "scheduler" process. Task bags can be modelled as channels in our setting and the posting of a task can be modelled by sending a message; the scheduling of a procedure call can be simulated as a receive of a non-deterministically selected channel which unlocks a

commutative procedure call defined by rules of types (i) and (ii) and rules of type (iii) where $C \in \mathcal{N}^{\mathrm{com}}$, in the sense of Definition 2. It is thus easy to see that APCPS with shaped stacks subsume programs with asynchronous procedure calls. In light of the fact that their safety verification is EXPSPACE-complete we can infer that the program-point coverability problem for APCPS with shaped stacks is EXPSPACE-hard.

Various extensions of Sen and Viswanathan's model [6] and applications to real-world asynchronous task scheduling systems [19] have been investigated. From the standpoint of message-passing concurrency, a key restriction of many of the models considered is that messages may only be retrieved by a communicating pushdown process when its stack is empty. The aim of this paper is to relax this restriction while retaining decidability of safety verification.

Communicating Pushdown Systems. The literature on communicating pushdown systems is vast. Numerous classes with decidable verification problems have been discovered. Heußner et al. [21] studied a restriction on pushdown processes that communicate asynchronously via FIFO channels: a process may send a message only when its stack is empty, while message retrieval is unconstrained. Several other communicating pushdown systems have been explored: parallel flow graph systems [13], visibly pushdown automata that communicate over FIFO-queues [1], pushdown systems communicating over locks [23], and recursive programs with hierarchical communication [4, 2].

Verification techniques that over-approximate correctness properties of concurrent pushdown systems have been studied [16, 20]. Under-approximation techniques typically impose constraints, such as bounding the number of context switches [31, 24], bounding the number of times a process can switch from a send-mode to receive-mode [3], or allowing symbols pushed onto the stack to be popped only within a bounded number of context switches [30]. Another line of work focuses on pushdown systems that communicate synchronously over channels, restricting model checking to synchronisation traces that fall within a restricted regular language [12]; this approach has been developed into an effective CEGAR method [25].

Future Directions and Conclusion. We have introduced a new class of asynchronously communicating pushdown systems, APCPS, and shown that the program-point coverability problem is decidable and EXPSPACE-hard for the subclass of APCPS with shaped stacks. We plan to investigate the precise complexity of the program-point coverability problem, construct an implementation and integrate it into SOTER [9, 10], a safety verifier for Erlang programs, to study APCPS empirically.

References

[1] D. Babic and Z. Rakamaric. Asynchronously communicating visibly pushdown systems. Technical Report UCB/EECS-2011-108, UC Berkeley, 2011. 5

[2] A. Bouajjani and M. Emmi. Analysis of recursively parallel programs. In *POPL*, pages 203–214, 2012. 5

[3] A. Bouajjani and M. Emmi. Bounded phase analysis of message-passing programs. In *TACAS*, pages 451–465, 2012. 5

[4] A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR*, pages 473–487, 2005. 5

[5] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983. 1

[6] R. Chadha and M. Viswanathan. Decidability results for well-structured transition systems with auxiliary storage. In *CONCUR*, pages 136–150, 2007. 5

[7] W. Czerwinski, S. B. Fröschle, and S. Lasota. Partially-commutative context-free processes. In *CONCUR*, pages 259–273, 2009. 1, 2, 5

[8] W. Czerwinski, P. Hofman, and S. Lasota. Reachability problem for weak multi-pushdown automata. In *CONCUR*, pages 53–68, 2012. 5

[9] E. D'Osualdo, J. Kochems, and C.-H. L. Ong. Soter: an automatic safety verifier for Erlang. In *AGERE! '12*, pages 137–140, 2012. 5

[10] E. D'Osualdo, J. Kochems, and C.-H. L. Ong. Automatic verification of Erlang-style concurrency. In *SAS*, 2013. To Appear. 5

[11] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inform.*, 31(1):13–25, 1997. 4.2, 5

[12] J. Esparza and P. Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL*, pages 499–510, 2011. 2, 5

[13] J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *POPL*, pages 1–11, 2000. 5

[14] J. Esparza, P. Ganty, S. Kiefer, and M. Luttenberger. Parikh's theorem: A simple and direct construction. *CoRR*, abs/1006.3825, 2010. 2, 4.2

[15] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001. 4.1

[16] C. Flanagan and S. Qadeer. Thread-modular model checking. In *SPIN*, pages 213–224, 2003. 5

[17] P. Ganty and R. Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6, 2012. 1, 4.2, 5

[18] P. Ganty, R. Majumdar, and A. Rybalchenko. Verifying liveness for asynchronous programs. In *POPL*, pages 102–113, 2009. 5

[19] G. Geeraerts, A. Heußner, and J.-F. Raskin. Queue-dispatch asynchronous systems. *CoRR*, abs/1201.4871, 2012. 5

[20] T. A. Henzinger, R. Jhala, R. Majumdar, and S. Qadeer. Thread-modular abstraction refinement. In *CAV*, pages 262–274, 2003. 5

[21] A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *FOSSACS*, pages 267–281, 2010. 5

[22] R. Jhala and R. Majumdar. Interprocedural analysis of asynchronous programs. In *POPL*, pages 339–350, 2007. 1, 5

[23] V. Kahlon. Boundedness vs. unboundedness of lock chains: Characterizing decidability of pairwise CFL-reachability for threads communicating via locks. In *LICS*, pages 27–36, 2009. 5

[24] A. Lal and T. W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. *Formal Methods in System Design*, 35(1):73–97, 2009. 5

[25] Z. Long, G. Calin, R. Majumdar, and R. Meyer. Language-theoretic abstraction refinement. In *FASE*, pages 362–376, 2012. 5

[26] M. Mukund, K. N. Kumar, J. Radhakrishnan, and M. A. Sohoni. Towards a characterisation of finite-state message-passing systems. In *ASIAN*, pages 282–299, 1998. 5

[27] M. Mukund, K. N. Kumar, J. Radhakrishnan, and M. A. Sohoni. Robust asynchronous protocols are finite-state. In *ICALP*, pages 188–199, 1998. 5

[28] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000. 1

[29] K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV*, pages 300–314, 2006. 1, 5

[30] S. L. Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, pages 203–218, 2011. 5

[31] S. L. Torre, P. Madhusudan, and G. Parlato. Reducing context-bounded concurrent reachability to sequential reachability. In *CAV*, pages 477–492, 2009. 5

[32] I. Wehrman. Higman's theorem and the multiset order, 2006. URL http://www.cs.utexas.edu/~iwehrman/pub/ms-wqo.pdf. c

A   Proof of Theorem 1

A.1   Direction: $\Leftarrow$

We lift define a function $\mathbb{M}(\!|\cdot|\!)$ over sequences $\mathcal{N}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ in the following way:

$$\mathbb{M}(\!|C_1\cdots C_n|\!) = \left\{ \bigoplus_{i=1}^{n} \mathbb{M}(w_i) \mid C_i \rightarrow^*_{\mathrm{seq}} w_i, w_i \in (\Sigma \cup \mathcal{N})^* /_{\simeq_I} \right\}$$

$$\mathbb{M}(\!|C_1\cdots C_n Z\alpha|\!) = \mathbb{M}(\!|C_1\cdots C_n|\!) \cdot Z \cdot \mathbb{M}(\!|\alpha|\!)$$

$$\mathbb{M}(\!|X\alpha|\!) = X \cdot \mathbb{M}(\!|\alpha|\!)$$

$$\mathbb{M}(\!|a\alpha|\!) = a \cdot \mathbb{M}(\!|\alpha|\!)$$

Let $U, V \subseteq Control^{\mathbb{M}}$, we define $U \rightarrow_{\mathrm{seq}'} V$ just if for all $\gamma' \in V$ there exists a $\gamma \in U$ such that $\gamma \rightarrow_{\mathrm{seq}'} \gamma'$.

Lemma 2. *If $\alpha \rightarrow_{seq} \beta$ such that $\alpha \in \mathcal{N}^*$ then $\mathbb{M}(\!|\alpha|\!) \rightarrow_{seq'} \mathbb{M}(\!|\beta|\!)$.*

Proof. Since $\alpha \rightarrow_{\mathrm{seq}} \beta$ we have $\alpha = X\alpha_0$ and $\beta = \alpha_1\alpha_0$ such that $X \rightarrow \alpha_1$. And so $\mathbb{M}(\!|\alpha|\!) = X\mathbb{M}(\!|\alpha_0|\!)$. We will proceed by case analysis on $X \rightarrow \alpha_1$.

- $X \rightarrow a, a \in \Sigma \cup \{\epsilon\}$.
  Take $a\delta \in a \cdot \mathbb{M}(\!|\alpha_0|\!) = \mathbb{M}(\!|a\alpha_0|\!) = \mathbb{M}(\!|\beta|\!)$. Then $X\delta \in \mathbb{M}(\!|\alpha|\!)$ and $X\delta \rightarrow_{\mathrm{seq}'} a\delta$. Hence $\mathbb{M}(\!|\alpha|\!) \rightarrow_{\mathrm{seq}'} \mathbb{M}(\!|\beta|\!)$.
- $X \rightarrow aA, a \in \Sigma$.
  Take $aA\delta \in a \cdot A \cdot \mathbb{M}(\!|\alpha_0|\!) = \mathbb{M}(\!|aA\alpha_0|\!) = \mathbb{M}(\!|\beta|\!)$, then since $X\delta \in \mathbb{M}(\!|\alpha|\!)$ and $X\delta \rightarrow_{\mathrm{seq}'} aA\delta$. Hence $\mathbb{M}(\!|\alpha|\!) \rightarrow_{\mathrm{seq}'} \mathbb{M}(\!|\beta|\!)$.
- $X \rightarrow AB, B \in \mathcal{N}^{\neg\mathrm{com}}$.
  Suppose $AB\delta \in A \cdot B \cdot \mathbb{M}(\!|\alpha_0|\!) = \mathbb{M}(\!|\alpha_1\alpha_0|\!) = \mathbb{M}(\!|\beta|\!)$, then $X\delta \in \mathbb{M}(\!|\alpha|\!)$ and $X\delta \rightarrow_{\mathrm{seq}'} AB\delta$. Hence $\mathbb{M}(\!|\alpha|\!) \rightarrow_{\mathrm{seq}'} \mathbb{M}(\!|\beta|\!)$.
- $X \rightarrow AB, B \in \mathcal{N}^{\mathrm{com}}$.
  Suppose $AM'\delta \in \mathbb{M}(\!|AB\alpha_0|\!) = \mathbb{M}(\!|\alpha_1\alpha_0|\!) = \mathbb{M}(\!|\beta|\!)$. Then clearly $M' = \mathbb{M}(w) \oplus M$ such that $B \rightarrow^*_{\mathrm{seq}} w$ and $M\delta \in \mathbb{M}(\!|\alpha_0|\!)$. Now then $XM\delta \in \mathbb{M}(\!|\alpha|\!)$ and $XM\delta \rightarrow_{\mathrm{seq}'} A(\mathbb{M}(w) \oplus M)\delta$. Thus $\mathbb{M}(\!|\alpha|\!) \rightarrow_{\mathrm{seq}'} \mathbb{M}(\!|\beta|\!)$.

$\square$

For $U, V \subseteq \mathbb{M}[Control^{\mathbb{M}}]$ define

$$U \parallel V := \{\Pi_0 \parallel \Pi_1 \mid \Pi_0 \in U, \Pi_1 \in V\}$$

Further we define

$$\mathbb{M}(\!|\Pi \parallel \Pi'|\!) := \mathbb{M}(\!|\Pi|\!) \parallel \mathbb{M}(\!|\Pi'|\!)$$

We say that for $U, V \subseteq \mathbb{M}[Control^{\mathbb{M}}]$ $U \lhd \Gamma \rightarrow_{\mathrm{con}'} V \lhd \Gamma'$, just if for all $\Pi' \in V$ there exists $\Pi \in U$ such that $\Pi \lhd \Gamma \rightarrow_{\mathrm{con}'} \Pi' \lhd \Gamma'$. Note this means that if $\mathbb{M}(\!|\alpha|\!) \rightarrow_{\mathrm{seq}'} \mathbb{M}(\!|\beta|\!)$ then clearly $\mathbb{M}(\!|\alpha \parallel \Pi|\!) \lhd \Gamma \rightarrow_{\mathrm{con}'} \mathbb{M}(\!|\beta \parallel \Pi|\!) \lhd \Gamma$ for all Π, Γ.

Lemma 3. *1. If $a \in \Sigma^{com}$ and $a\alpha\alpha' \parallel \Pi \lhd \Gamma \to_{con} \alpha\alpha' \parallel \Pi \oplus \Pi(a) \lhd \Gamma \oplus \Gamma(a)$ where $\alpha \in (\mathcal{N} \cup \{\epsilon\})\mathcal{N}^{com*}$, $\alpha' \in (\mathcal{N}^{\neg com}\mathcal{N}^{com})^*$, then*

$$\mathbb{M}(\!|a\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi \oplus \Pi(a)|\!) \lhd \Gamma \oplus \Gamma(a).$$

2. *If $(c \,?\, m)\alpha\alpha' \parallel \Pi \lhd \Gamma \oplus \Gamma(c \,!\, m) \to_{con} \alpha\alpha' \parallel \Pi \lhd \Gamma$ where $\alpha \in (\mathcal{N} \cup \{\epsilon\})\mathcal{N}^{com*}$, $\alpha' \in (\mathcal{N}^{\neg com}\mathcal{N}^{com})^*$, then*

$$\mathbb{M}(\!|(c \,?\, m)\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \oplus \Gamma(c \,!\, m) \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma.$$

Claim 1. We show that $\mathbb{M}(\!|a\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi'|\!) \lhd \Gamma'$ by case analysis on a:

- $a = c \,!\, m$
  Then $\Pi \oplus \Pi(a) = \Pi$,
  Take $\gamma \parallel \pi \in \mathbb{M}(\!|\alpha\alpha' \parallel \Pi|\!)$ then $(c \,!\, m)\gamma \parallel \pi \in \mathbb{M}(\!|(c \,!\, m)\alpha\alpha' \parallel \Pi|\!)$. Using rule 14 we see that

  $$(c \,!\, m)\gamma \parallel \pi \lhd \Gamma \to_{con'} \gamma \parallel \pi \lhd \Gamma \oplus \Gamma(c \,!\, m).$$

  Hence we conclude $\mathbb{M}(\!|(c \,!\, m)\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi|\!) \lhd \oplus\Gamma(c \,!\, m)$.
- $a = \nu X$
  Then $\Pi \oplus \Pi(\nu X) = \Pi \parallel X$, $\Gamma = \Gamma$
  Take $\gamma \parallel X \parallel \pi \in \mathbb{M}(\!|\alpha\alpha' \parallel X \parallel \Pi|\!) = \mathbb{M}(\!|\alpha\alpha' \parallel \Pi \oplus \Pi(\nu X)|\!)$ then $(\nu X)\gamma \parallel \pi \in \mathbb{M}(\!|(\nu X)\alpha\alpha' \parallel \Pi|\!)$. Using rule 13 we see that

  $$(\nu X)\gamma \parallel \pi \lhd \Gamma \to_{con'} \gamma \parallel X \parallel \pi \lhd \Gamma.$$

  Hence we conclude $\mathbb{M}(\!|(\nu X)\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi \oplus \Pi(\nu X)|\!) \lhd \Gamma$.
- $a = l$
  Then $\Pi \oplus \Pi(l) = \Pi$, $\Gamma \oplus \Gamma(l) = \Gamma$. Take $\gamma \parallel \pi \in \mathbb{M}(\!|\alpha\alpha' \parallel \Pi|\!)$ then $l\gamma \parallel \pi \in \mathbb{M}(\!|l\alpha\alpha' \parallel \Pi|\!)$. Using rule 15 we see that

  $$l\gamma \parallel \pi \lhd \Gamma \to_{con'} \gamma \parallel \pi \lhd \Gamma.$$

  Hence we conclude
  $\mathbb{M}(\!|l\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma$.

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \square$

Claim 2. Take $\gamma \parallel \pi \in \mathbb{M}(\!|\alpha\alpha' \parallel \Pi|\!) = \mathbb{M}(\!|\alpha\alpha' \parallel \Pi|\!)$ then $(c \,?\, m)\gamma \parallel \pi \in \mathbb{M}(\!|(c \,?\, m)\alpha\alpha' \parallel \Pi|\!)$
Then using rule 12 we see that

$$(c \,?\, m)\gamma \parallel \pi \lhd \Gamma \oplus \Gamma(c \,!\, m) \to_{con'} \gamma \parallel \pi \lhd \Gamma.$$

Hence we conclude $\mathbb{M}(\!|(c \,?\, m)\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \oplus \Gamma(c \,!\, m) \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma$.
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \square$

Lemma 4.

If $a_1 \cdots a_n \in \Sigma^{com*}$, $\alpha_i \in \mathcal{N}^{com*}$ and $\alpha' \in (\mathcal{N}^{\neg com}\mathcal{N}^{com})^*$,

$$\alpha_1\alpha' \parallel \Pi(\epsilon) \lhd \Gamma(\epsilon) \to^*_{con} a_1\alpha_2\alpha' \parallel \Pi(\epsilon) \lhd \Gamma(\epsilon)$$
$$\to_{con} \alpha_2\alpha' \parallel \Pi(a_1) \lhd \Gamma(a_1)$$
$$\to^*_{con} \cdots \to^*_{con}$$
$$a_n\alpha_{n+1}\alpha' \parallel \Pi(a_1 \cdots a_{n-1}) \lhd \Gamma(a_1 \cdots a_{n-1})$$
$$\to^*_{con} \alpha_{n+1}\alpha' \parallel \Pi(a_1 \cdots a_n) \lhd \Gamma(a_1 \cdots a_n)$$

then

$$\mathbb{M}(\!|\alpha_1\alpha' \parallel \Pi(\epsilon)|\!) \lhd \Gamma(\epsilon) \to^*_{con'} \mathbb{M}(\!|\alpha_{n+1}\alpha' \parallel \Pi(a_1 \cdots a_n)|\!) \lhd \Gamma(a_1 \cdots a_n).$$

Proof. We prove the claim by induction on n. For $n = 0$ the claim is vacuously true.

For $n = k + 1$, assuming the claim holds for k it is enough to show that if

$$\alpha_{k+1}\alpha' \parallel \Pi(a_1 \cdots a_k) \lhd \Gamma(a_1 \cdots a_k) \to^*_{con} a_{k+1}\alpha_{k+2}\alpha' \parallel \Pi(a_1 \cdots a_k) \lhd \Gamma(a_1 \cdots a_k)$$
$$\to^*_{con} \alpha_{k+2}\alpha' \parallel \Pi(a_1 \cdots a_{k+1}) \lhd \Gamma(a_1 \cdots a_{k+1})$$

then

$$\mathbb{M}(\!|\alpha_{k+1}\alpha' \parallel \Pi(a_1 \cdots a_k)|\!) \lhd \Gamma(a_1 \cdots a_k) \to^*_{con'} \mathbb{M}(\!|\alpha_{k+2}\alpha' \parallel \Pi(a_1 \cdots a_{k+1})|\!) \lhd \Gamma(a_1 \cdots a_{k+1})$$

which we obtain by repeatedly applying Lemma 2 and then Lemma 3. $\qquad\square$

Lemma 5. *If $a\alpha\alpha' \parallel \Pi \lhd \Gamma \to_{con} \alpha\alpha' \parallel \Pi' \lhd \Gamma' \to^*_{con} \alpha' \parallel \Pi'' \lhd \Gamma''$ where $a \in \Sigma$, $\alpha \in \mathcal{N}^{com*}$, $\alpha' \in (\mathcal{N}^{\neg com}\mathcal{N}^{com})^*$, $\alpha \to^*_{seq} w \in \Sigma^*/\!\simeq_I$, $\Pi'' = \Pi' \oplus \Pi(w)$, $\Gamma'' = \Gamma' \oplus \Gamma(w)$ then*

$$\mathbb{M}(\!|a\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi'|\!) \lhd \Gamma' \to_{con'} \mathbb{M}(\!|\alpha' \parallel \Pi''|\!) \lhd \Gamma''.$$

Proof. For $\mathbb{M}(\!|a\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi'|\!) \lhd \Gamma'$ we appeal to Lemma 3.
Now since $\alpha \to^*_{seq} w$ we have $\alpha\alpha' \parallel \Pi' \lhd \Gamma' \to^*_{con} \alpha' \parallel \Pi' \oplus \Pi(w) \lhd \Gamma' \oplus \Gamma(w)$.

Let $M = \mathbb{M}(w)$ and take $\gamma \parallel \pi' \oplus \Pi(M) \lhd \Gamma' \oplus \Gamma(M) \in \mathbb{M}(\!|\alpha' \parallel \Pi' \oplus \Pi(w)|\!) \lhd \Gamma' \oplus \Gamma(w)$
Then note $M\gamma \parallel \pi' \in \mathbb{M}(\!|\alpha\alpha' \parallel \Pi'|\!)$ and using rule 16

$$M\gamma \parallel \pi' \lhd \Gamma' \to_{con'} \gamma \parallel \pi' \oplus \Pi(M) \lhd \Gamma' \oplus \Gamma(M).$$

$\qquad\square$

Lemma 6. *If $a\alpha\alpha' \parallel \Pi \lhd \Gamma \to_{con} \alpha\alpha' \parallel \Pi' \lhd \Gamma' \to^*_{con} w\alpha' \parallel \Pi' \lhd \Gamma' \to^*_{con} w'\alpha' \parallel \Pi'' \lhd \Gamma''$ where $a \in \Sigma$, $\alpha \in \mathcal{N}^{com*}$, $\alpha' \in (\mathcal{N}^{\neg com}\mathcal{N}^{com})^*$, $\alpha'' \in \mathcal{N}^{\neg com*}$ and $\alpha \to^*_{seq} w \in (\Sigma \cup \mathcal{N})^*/\!\simeq_I$, $w' \in \mathcal{N}^*/\!\simeq_I$, $\Pi'' = \Pi' \oplus \Pi(w)$, $\Gamma'' = \Gamma' \oplus \Gamma(w)$ then*

$$\mathbb{M}(\!|a\alpha\alpha' \parallel \Pi|\!) \lhd \Gamma \to_{con'} \mathbb{M}(\!|\alpha\alpha' \parallel \Pi'|\!) \lhd \Gamma' \to_{con'} \mathbb{M}(w') \cdot \mathbb{M}(\!|\alpha'|\!) \parallel \mathbb{M}(\!|\Pi''|\!) \lhd \Gamma''.$$

Proof. For $\mathbb{M}(\!|a\alpha\alpha'\parallel\varPi|\!) \lhd \varGamma \rightarrow_{\text{con}'} \mathbb{M}(\!|\alpha\alpha'\parallel\varPi'|\!) \lhd \varGamma'$ we appeal to Lemma 3. Now since $\alpha \rightarrow^*_{\text{seq}} w$ we have $\alpha\alpha'\parallel\varPi' \lhd \varGamma' \rightarrow^*_{\text{con}} w'\alpha'\parallel\varPi'\oplus\varPi(w) \lhd \varGamma'\oplus\varGamma(w)$.

Let $M' = M(w')$ then

$M'\gamma\parallel\pi'\oplus\varPi(M) \lhd \varGamma'\oplus\varGamma(M) \in \mathbb{M}(w')\cdot\mathbb{M}(\!|\alpha'|\!)\parallel\mathbb{M}(\!|\varPi'\oplus\varPi(w)|\!) \lhd \varGamma'\oplus\varGamma(w)$ and $M\gamma\parallel\pi' \in \mathbb{M}(\!|\alpha\alpha'\parallel\varPi'|\!)$ such that $M = \mathbb{M}(w)$. Using rule 17

$$M\gamma\parallel\pi' \lhd \varGamma' \rightarrow_{\text{con}'} M'\gamma\parallel\pi'\oplus\varPi(M) \lhd \varGamma'\oplus\varGamma(M).$$

$\square$

Lemma 7. *Let* $X \in \mathcal{N}$, $\alpha \in (\mathcal{N}^{\neg com}\mathcal{N}^{com*})^*$, $w \in \Sigma^{com}$ *and* $\beta, \alpha' \in \mathcal{N}^{com*}$

1. *If* $X\beta\alpha \lhd \varGamma(\epsilon)\varGamma(\epsilon) \rightarrow^*_{con} \alpha\parallel\varPi(w) \lhd \varGamma(w)$ *then* $\mathbb{M}(\!|X\beta\alpha|\!) \lhd \varGamma(\epsilon) \rightarrow^*_{con'} \mathbb{M}(\!|\alpha\parallel\varPi(w)|\!) \lhd \varGamma(w)$
2. *If* $X\beta\alpha \lhd \varGamma(\epsilon) \rightarrow^*_{con'} \alpha'\alpha\parallel\varPi(w) \lhd \varGamma(w)$, *then* $\mathbb{M}(\!|X\beta\alpha|\!) \lhd \varGamma(\epsilon) \rightarrow^*_{con} \mathbb{M}(\alpha')\cdot\mathbb{M}(\!|\alpha|\!)\parallel \mathbb{M}(\!|\varPi(w)|\!) \lhd \varGamma(w)$

Claim 1. Then $X\beta\alpha \rightarrow^*_{\text{seq}} a\alpha_0\alpha$ where $a \in \Sigma^{\text{com}}\cup\{\epsilon\}$ such that $a\alpha_0 \rightarrow^*_{\text{seq}} w$ so by Lemma 2 $\mathbb{M}(\!|X\beta\alpha|\!) \rightarrow^*_{\text{seq}'} \mathbb{M}(\!|a\alpha_0\alpha|\!)$. By Lemma 5 $\mathbb{M}(\!|a\alpha_0\alpha|\!) \lhd \varGamma(\epsilon) \rightarrow^*_{\text{con}'} \mathbb{M}(\!|\alpha\parallel\varPi(w)|\!) \lhd \varGamma(w)$ and clearly also $\mathbb{M}(\!|X\beta\alpha|\!) \lhd \varGamma(\epsilon) \rightarrow^*_{\text{con}'} \mathbb{M}(\!|a\alpha_0\alpha|\!)\parallel\varPi(w) \lhd \varGamma(w)$.

$\square$

Claim 2. Then $X\beta\alpha \rightarrow^*_{\text{seq}} a\alpha_0\alpha$ where $a \in \Sigma^{\text{com}}\cup\{\epsilon\}$ such that $a\alpha_0 \rightarrow^*_{\text{seq}} w\alpha'$ so by Lemma 2 $\mathbb{M}(\!|X\beta\alpha|\!) \rightarrow^*_{\text{seq}'} \mathbb{M}(\!|a\alpha_0\alpha|\!)$. By Lemma 6 $\mathbb{M}(\!|a\alpha_0\alpha|\!) \lhd \varGamma(\epsilon) \rightarrow^*_{\text{con}'} \mathbb{M}(\!|\alpha'\alpha\parallel\varPi(w)|\!) \lhd \varGamma(w)$ and clearly also $\mathbb{M}(\!|X\alpha|\!) \lhd \varGamma(\epsilon) \rightarrow^*_{\text{con}'} \mathbb{M}(\!|a\alpha_0\alpha|\!)\parallel\varPi(w) \lhd \varGamma(w)$.

$\square$

Lemma 8. *Let* $X \in \mathcal{N}^{\neg com}$, $\beta, \beta' \in \mathcal{N}^{com*}$ *and* $\alpha, \alpha' \in (\mathcal{N}^{\neg com}\mathcal{N}^{com*})^*$.

1. *If* $X\beta\alpha \lhd \varGamma(\epsilon) \rightarrow^*_{con} \alpha'\alpha\parallel\varPi(w) \lhd \varGamma(w)$ *then* $\mathbb{M}(\!|X\beta\alpha|\!) \lhd \varGamma(\epsilon) \rightarrow^*_{con'} \mathbb{M}(\!|\alpha'\alpha\parallel\varPi(w)|\!) \lhd \varGamma(w)$
2. *If* $X\beta\alpha \lhd \varGamma(\epsilon) \rightarrow^*_{con} (c\,?\,m)\beta'\alpha'\alpha\parallel\varPi(w) \lhd \varGamma(w)$, *then* $\mathbb{M}(\!|X\alpha\parallel\varPi(w)|\!) \lhd \varGamma(\epsilon) \rightarrow^*_{con'} \mathbb{M}(\!|(c\,?\,m)\beta'\alpha'\alpha|\!) \lhd \varGamma(\epsilon)$.

Claim 1. Then

$$X\beta\alpha \lhd \varGamma(\epsilon) \rightarrow^*_{\text{con}} X'\alpha'\alpha\parallel\varPi(w_0) \lhd \varGamma(w_0)$$

such that $X' \rightarrow^*_{\text{seq}} a\alpha_0$ where $a \in \Sigma^{\text{com}}\cup\{\epsilon\}$, $\alpha_0 \in \mathcal{N}^{\text{com}*}$ such that $a\alpha_0 \rightarrow^*_{\text{seq}} w_1$ and $w = w_0w_1$. By Lemma 4

$$\mathbb{M}(\!|X\beta\alpha|\!) \lhd \varGamma(\epsilon) \rightarrow^*_{\text{con}'} \mathbb{M}(\!|X'\alpha'\alpha\parallel\varPi(w_0)|\!) \lhd \varGamma(w_0)$$

Then the proof of Lemma 7 Claim 1 applies to give the result. $\square$

Claim 2. Then

$$X\beta\alpha \lhd \varGamma(\epsilon) \rightarrow^*_{\text{con}'} X'X''\beta'\alpha'\alpha\parallel\varPi(w_0) \lhd \varGamma(w_0)$$

such that $X'' \rightarrow^*_{\text{seq}} c\,?\,m$, and $X' \rightarrow^*_{\text{seq}} w_1$ where $w = w_0w_1$. Hence

$$X'X''\beta'\alpha'\alpha\parallel\varPi(w_0) \lhd \varGamma(w_0) \rightarrow^*_{\text{con}'} (c\,?\,m)\beta'\alpha'\alpha\parallel\varPi(w) \lhd \varGamma(w).$$

so by Lemma 4 and Lemma 2.

$$\mathbb{M}(\!(X\beta\alpha)\!) \lhd \varGamma(\epsilon) \to^*_{\text{con}'} \mathbb{M}(\!((c\,?\,m)\beta'\alpha'\alpha \parallel \varPi(w))\!) \lhd \varGamma(w).$$

$\square$

For $\alpha_1, \ldots, \alpha_m \in \mathcal{N}^{\text{com}*}$ and $Z_1, \ldots, Z_{m-1} \in \mathcal{N}^{\neg\text{com}}$ define

$$\overline{\mathbb{M}}(\alpha_1 Z_1 \cdots \alpha_{m-1} Z_{m-1} \alpha_m) := \mathbb{M}(\alpha_1) Z_1 \cdots \mathbb{M}(\alpha_{m-1}) Z_{m-1} \mathbb{M}(\alpha_m)$$

$$\overline{\mathbb{M}}(\varPi \parallel \varPi') := \overline{\mathbb{M}}(\varPi) \parallel \overline{\mathbb{M}}(\varPi')$$

Proposition 2. *If $S \lhd \varGamma(\epsilon) \to^*_{\text{con}} \varPi' \lhd \varGamma'$ then $\overline{\mathbb{M}}(S) \lhd \varGamma(\epsilon) \to^*_{\text{con}'} \overline{\mathbb{M}}(\varPi') \lhd \varGamma'$*

Proof. Let $\varPi^f \in \mathbb{M}[Control]$ and define the set $P_{\varPi^f} = \{\alpha \mid \exists \varPi.\alpha \parallel \varPi = \varPi^f\}$. further define the set of configurations $P := \{\varPi \lhd \varGamma \mid \forall \alpha \in \varPi, \alpha \in \mathcal{N}(\mathcal{N}^{\neg\text{com}}\mathcal{N}^{\text{com}*})^* \cup \Sigma^{\neg\text{com}}\mathcal{N}^{\text{com}*}(\mathcal{N}^{\neg\text{com}}\mathcal{N}^{\text{com}*})^* \cup \mathcal{N} \cup P_{\varPi^f}\}$.

Now suppose that for some $\varPi, \varPi'$ and $\varGamma, \varGamma'$

$$\varPi \lhd \varGamma := \varPi_0 \lhd \varGamma_0 \to^*_{\text{con}} \varPi_1 \lhd \varGamma_1 \to^*_{\text{con}} \cdots \to^*_{\text{con}} \varPi_n \lhd \varGamma_n =: \varPi' \lhd \varGamma'$$

such that $\varPi_i \lhd \varGamma_i \in P$ for $i = 0, \ldots n$. Without loss of generality we can assume that for all $i = 0, \ldots, n$, $\varPi_i = \varPi_i^a \parallel \varPi_i^f$ such that for all $\alpha \in \varPi_i^f$ we have $\alpha \in P_{\varPi^f}$ and α is not involved in any transitions in $\varPi_i \lhd \varGamma_i \to^*_{\text{con}} \varPi_n \lhd \varGamma_n$. Note that we are not loosing generality, since a reduction $\alpha \parallel \varPi \to^*_{\text{con}} \alpha \parallel \varPi'$ can either be pre-empted or goes through a process state in $\Sigma^{\neg\text{com}}\mathcal{N}^{\text{com}*}(\mathcal{N}^{\neg\text{com}}\mathcal{N}^{\text{com}*})^*$. Note this also means that $\varPi_{i+1}^f = \varPi_i^f \parallel \varPi_i'^f$. We further assume w.l.o.g that for each i it is the case that $\varPi_i^a = \alpha \parallel \varPi_i'$ and $\varPi_{i+1}^a = \alpha' \parallel \varPi_i' \oplus \varPi(w)$ and $\varGamma_{k+1} \oplus \varGamma(w') = \varGamma_k \oplus \varGamma(w)$ for some $w \in \Sigma^{\text{com}*}$ and $w' \in \{\epsilon\} \cup \Sigma^{\text{com}}$, i.e. during each $\varPi_i^a \lhd \varGamma_i \to^*_{\text{con}} \varPi_{i+1}^a \lhd \varGamma_{i+1}$ only one process makes progress (note this can be achieved by delaying receptions and performing sends and spawns as early as possible) and none of the intermediate steps are configurations of P.

We will prove by induction on n:

$$\mathbb{M}(\!(\varPi_0^a)\!) \parallel \tilde{\varPi}_0^f \lhd \varGamma_0 \to^*_{\text{con}'} \mathbb{M}(\!(\varPi_1^a)\!) \parallel \tilde{\varPi}_1^f \lhd \varGamma_1 \to^*_{\text{con}'} \cdots \to^*_{\text{con}'} \mathbb{M}(\!(\varPi_n)\!) \parallel \tilde{\varPi}_n^f \lhd \varGamma_n$$

where for all $i = 0, \ldots n$ and $\alpha \in \varPi_i^f$ we have either $\varPi_i^f(\alpha) = \tilde{\varPi}_i^f(\mathbb{M}(\!(\alpha)\!))$ or $\varPi_i^f(\alpha) = \tilde{\varPi}_i^f(\mathbb{M}(\alpha_0) \cdot \mathbb{M}(\!(\alpha_1)\!))$, $\alpha = \alpha_0 \alpha_1$.

- $n = 0$.
  The claim holds trivially.
- $n = k + 1$, assuming the claim holds for k.
  To prove the inductive claim we need to show that from $\varPi_k = \alpha \parallel \varPi_k'$, $\varPi_{k+1} = \alpha' \parallel \varPi_k' \oplus \varPi(w)$ and $\varGamma_{k+1} \oplus \varGamma(w') = \varGamma_k \oplus \varGamma w$ where $\varPi_k \lhd \varGamma_k \to^*_{\text{con}} \varPi_{k+1} \lhd \varGamma_{k+1}$, we can infer $\mathbb{M}(\!(\varPi_k)\!) \lhd \varGamma_k \to^*_{\text{con}} \mathbb{M}(\!(\varPi_{k+1})\!) \lhd \varGamma_{k+1}$. We will do so by a case analysis on the shape of α and α'.
  - $\alpha, \alpha' \in (\mathcal{N}^{\neg\text{com}}\mathcal{N}^{\text{com}*})^*$
    Then $\alpha = X\alpha_0\alpha_1$, $X \in \mathcal{N}^{\neg\text{com}}$, $\alpha_0 \in \mathcal{N}^{\text{com}*}$, $\alpha_1 \in (\mathcal{N}^{\neg\text{com}}\mathcal{N}^{\text{com}*})^*$ and $\alpha' = \alpha_0'\alpha_1$ where $\alpha_0' \in \epsilon \cup (\mathcal{N}^{\neg\text{com}}\mathcal{N}^{\text{com}*})^*$, i.e. either we increase the call-stack or we pop one non-commutative non-terminal off the call-stack. Otherwise we would end up either in an intermediate configuration in P or in a different case.

      ∗ Case $\alpha_0' = \epsilon$.

        Then $X\alpha_0\alpha_1 \parallel \Pi_k' \lhd \Gamma_k \to_{\mathrm{con}}^* X'\alpha_2\alpha_0\alpha_1 \parallel \Pi_k' \lhd \Gamma_k$ such that $X' \in \mathcal{N}^{\mathrm{com}}$, $\alpha_2 \in \mathcal{N}^{\mathrm{com}*}$ and $X'\alpha_2\alpha_0\alpha_1 \parallel \Pi_k' \lhd \Gamma_k \to_{\mathrm{con}}^* w\alpha_1 \parallel \Pi_k' \lhd \Gamma_k \to_{\mathrm{con}}^* \alpha_1 \parallel \Pi_k' \oplus \Pi(w) \lhd \Gamma_k \oplus \Gamma(w)$, where $w \in \Sigma^{\mathrm{com}*}$ such that $\Pi_{k+1} = \alpha' \parallel \Pi_k' \oplus \Pi(w)$ and $\Gamma_{k+1} = \Gamma_k \oplus \Gamma w$. Lemma 3 then allows us to conclude that $\mathbb{M}(\!|X\alpha_0\alpha_1 \parallel \Pi_k'|\!) \lhd \Gamma_k \to_{\mathrm{con}'}^* \mathbb{M}(\!|X'\alpha_2\alpha_0\alpha_1 \parallel \Pi_k'|\!) \lhd \Gamma_k$ and Lemma 7.1 gives us $\mathbb{M}(\!|X'\alpha_2\alpha_0\alpha_1 \parallel \Pi_k'|\!) \lhd \Gamma_k \to_{\mathrm{con}'}^* \mathbb{M}(\!|\alpha_1 \parallel \Pi_k' \oplus \Pi(w)|\!) \lhd \Gamma_k \oplus \Gamma(w) = \Pi_{k+1} \lhd \Gamma_{k+1}$.

      ∗ Case $\alpha_0' \neq \epsilon$.

        Follows directly from Lemma 8.1

- $\alpha \in \Sigma^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ and $\alpha' \in (\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$

  Follows from Lemma 5.

- $\alpha \in (\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ and $\alpha' \in \Sigma^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$

  Follows from Lemma 8.2

- $\alpha \in \mathcal{N}$ and $\alpha' \in (\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$

  We can assume that $\alpha \in \mathcal{N}^{\mathrm{com}}$ since otherwise a case above already applies. By the definition of $\mathcal{N}^{\mathrm{com}}$ we can thus infer that $\alpha' = \epsilon$ since otherwise α would not be commutative. Thus Lemma 7.1 applies.

- $\alpha \in \mathcal{N}$ and $\alpha' \in \Sigma^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$

  There is nothing to prove for this case as, similarly to the case above, either $\alpha \in \mathcal{N}^{\neg\mathrm{com}}$ and so a case above applies or $\alpha \in \mathcal{N}^{\mathrm{com}}$ but then $\alpha' \notin \Sigma^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ which is impossible; so the former must be the case.

- $\alpha \in (\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ and $\alpha' \in P_{\Pi_f}$

  If $\alpha' \in (\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^* \cup \Sigma^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ the above cases apply. Otherwise it must be the case that $\alpha' \in \mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^* \cup \Sigma^{\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$.

     ∗ $\alpha' \in \mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$

       So it must be the case that $\alpha = X\alpha_0\alpha_1$, $X \in \mathcal{N}^{\neg\mathrm{com}}$, $\alpha_0 \in \mathcal{N}^{\mathrm{com}*}$ $\alpha_1 \in (\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ and $\alpha' = \alpha_0'\alpha_1'\alpha_1$ where $\alpha_1' \in (\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$, $\alpha_0' \in \mathcal{N}^{\mathrm{com}*}$ Lemma 7.2 applies to give

  $$\mathbb{M}(\!|X\alpha_0\alpha_1 \parallel \Pi_k|\!) \lhd \Gamma_k \to_{\mathrm{con}'}^* \mathbb{M}(\alpha_0') \cdot \mathbb{M}(\!|\alpha_1'\alpha_1|\!) \parallel \mathbb{M}(\!|\Pi_k|\!) \parallel \mathbb{M}(\!|\Pi(w)|\!) \lhd \Gamma_k \oplus \Gamma(w)$$

     ∗ $\alpha' \in \Sigma^{\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$

       Follows from Lemma 4

- $\alpha \in \Sigma^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ and $\alpha' \in P_{\Pi_f}$

  Unless $\alpha' \in \mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^* \cup \Sigma^{\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ this case is covered by a case above. The remaining follows from Lemma 6.

- $\alpha \in \mathcal{N}$ and $\alpha' \in P_{\Pi_f}$

  Unless $\alpha' \in \mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^* \cup \Sigma^{\mathrm{com}}\mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ this case is covered by a case above. The remaining follows from Lemma 2 and Lemma 4

This concludes the proof of the inductive step.

Now we apply the above for the case that $\Pi_0 \lhd \Gamma_0 = S \lhd \Gamma(\epsilon)$ and $\Pi^f := \Pi'$. We can then see that $\mathbb{M}(\!|S|\!) \lhd \Gamma(\epsilon) \to_{\mathrm{con}'}^* \tilde{\Pi}^f \lhd \Gamma'$.

Then since for all $\alpha \in \mathcal{N}^{\mathrm{com}*}(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^*$ it is the case that $\overline{\mathbb{M}}(\alpha) \in \mathbb{M}(\!|\alpha|\!)$ and further for $\alpha_0 \in \mathcal{N}^{\mathrm{com}*}$, $\alpha_1(\mathcal{N}^{\neg\mathrm{com}}\mathcal{N}^{\mathrm{com}*})^* \overline{\mathbb{M}}(\alpha_0\alpha_1) \in \mathbb{M}(\alpha_0) \cdot \mathbb{M}(\!|\alpha_1|\!)$ we can

deduce from the definition of $\rightarrow_{con'}$ on sets of configurations that

$$\overline{\mathbb{M}}(S) \rightarrow^*_{con'} \overline{\mathbb{M}}(\Pi') \lhd \Gamma'$$

which concludes the proof. $\square$

A.2   Direction: $\Rightarrow$

$$\lceil M \rceil = \left\{ \left\{ C_i \mapsto \sum_{(C_i, w) \in M'} M'(C_i, w) \right\} \;\middle|\; \begin{array}{l} M = \bigoplus_{(C_i, w) \in M'} \overset{M'(C_i, w)}{\underset{j=1}{\bigoplus}} \mathbb{M}(w), \\[2mm] M' \in \mathbb{M}[\{(C, w) \mid C \rightarrow^*_{seq} w, C \in \mathcal{N}\}] \end{array} \right\}$$

$$[\![M]\!] = \{\alpha \in Control \mid \mathbb{M}(\alpha) = \lceil M \rceil\}$$

$$[\![\gamma_1 \cdots \gamma_n]\!] = [\![\gamma_1]\!] \cdots [\![\gamma_n]\!] \text{ where } \gamma_i \in \mathcal{N} \cup Cache \cup \Sigma$$

Define for $V, W \subseteq \mathbb{M}[Control]$

$$V \parallel W = \{\Pi \parallel \Pi' \mid \Pi \in V, \Pi' \in W\}$$

and $\Pi, \Pi' \in \mathbb{M}[Control^{\mathbb{M}}]$

$$[\![\Pi \parallel \Pi']\!] = [\![\Pi]\!] \parallel [\![\Pi']\!]$$

Definition 7 (Simulation Relation). Let $R \subseteq S \times S'$ where $(S, \rightarrow_S), (S', \rightarrow_{S'})$ are transition systems. We say R is a *weak (S, S')-simulation* just if

$$(q, p) \in R \text{ and } q \rightarrow_S q' \Rightarrow p \rightarrow^*_{S'} p' \text{ and } (q', p') \in R.$$

Let $\preccurlyeq_S := \{(\gamma, \alpha) \mid \alpha \in [\![\gamma]\!]\} \subseteq Control^{\mathbb{M}} \times Control$ and $\preccurlyeq_C := \{(\Pi_1 \lhd \Gamma, \Pi_2 \lhd \Gamma) \mid \Pi_2 \in [\![\Pi_1]\!]\}$.

Proposition 3 (Sequential Simulation). $\preccurlyeq_S$ *is a weak simulation relation.*

Proof. Let $\gamma \in Control^M$ and $\alpha \in Control$ such that $\alpha \in [\![\gamma]\!]$ and $\gamma \rightarrow_{seq'} \gamma'$.

Since $\gamma \rightarrow_{seq'} \gamma'$ we know that $\gamma = XM\gamma_0$ and $\gamma' = \gamma_1\gamma_0$. Hence by definition of $[\![\gamma]\!]$ it must be that $\alpha = X\beta\alpha_0$ such that $\beta \in [\![M]\!]$ and $\alpha_0 \in [\![\gamma_0]\!]$.

We will prove that there exists a $\alpha' \succcurlyeq_S \gamma'$ such that $\alpha \rightarrow_{seq} \alpha'$ by case analysis on the type of rule used for $\gamma \rightarrow_{seq'} \gamma'$.

Claim 1 and 2: $X \rightarrow a \in G$, $a \in \Sigma \cup \{\epsilon\}$. *(trivial)*
Clearly $X\beta\alpha_0 \rightarrow_{seq} a\beta\alpha_0$, and $\gamma_1 = aM$. Clearly $a\beta\alpha_0 =: \alpha' \in [\![aM\gamma_0]\!] = [\![\gamma']\!]$ and so $\alpha' \succcurlyeq_S \gamma'$.

Claim 3: $X \rightarrow aA \in G$. *(trivial)*
Then $X\beta\alpha_0 \rightarrow_{seq} aA\beta\alpha_0$ and $\gamma_1 = aAM$. Clearly $aA\beta\alpha_0 =: \alpha' \in [\![aAM\gamma_0]\!] = [\![\gamma']\!]$ and so $\gamma' \preccurlyeq_S \alpha'$.

Claim 4: $X \to AB \in G$, $B \in \mathcal{N}^{\mathrm{com}}$, $B \to_{\mathrm{seq}}^{*} w$. *(non-trivial)*

Then $X\beta\alpha_0 \to_{\mathrm{seq}} AB\beta\alpha_0$. To prove $A\,B\,\beta\alpha_0 =: \alpha' \in [\![A(\mathbb{M}(w) \oplus M)\gamma_0]\!] = [\![\gamma']\!]$ we need to show that $B\,\beta \in [\![\mathbb{M}(w) \oplus M]\!]$. Since $\beta \in [\![M]\!]$ we know that $\mathbb{M}(\beta) = \lceil M \rceil$. It remains to prove $\mathbb{M}(B) \oplus \mathbb{M}(\beta) \in \lceil \mathbb{M}(w) \oplus M \rceil$. Since $\mathbb{M}(\beta) \in \lceil M \rceil$ there exists $M' \in \mathbb{M}[\{(C,w) \mid C \to_{\mathrm{seq}}^{*} w, C \in \mathcal{N}\}]$ such that

$$\mathbb{M}(\beta) = \left\{ C_i \mapsto \sum_{(C_i,w) \in M'} M'(C_i,w) \right\}$$

and $M = \bigoplus_{(C_i,w) \in M'} \bigoplus_{j=1}^{M'(C_i,w)} \mathbb{M}(w)$. Then writing $M'_B := M' \oplus \mathbb{M}((B,w))$ it is the case that $M'_B \in \mathbb{M}[\{(C,w) \mid C \to_{\mathrm{seq}}^{*} w, C \in \mathcal{N}\}]$ and

$$M \oplus \mathbb{M}(w) = \left(\bigoplus_{(C_i,w') \in M'} \bigoplus_{j=1}^{M'(C_i,w')} \mathbb{M}(w') \right) \oplus \mathbb{M}(w)$$

$$= \bigoplus_{(C_i,w') \in M'_B} \bigoplus_{j=1}^{M'_B(C_i,w')} \mathbb{M}(w')$$

Thus we can conclude that

$$f_B := \left\{ C_i \mapsto \sum_{(C_i,w) \in M'_B} M'_B(C_i,w) \right\} \in \lceil \mathbb{M}(w) \oplus M \rceil$$

and since $M'_B(B,w) = M'(B,w) + 1$ and $M'_B(B',w') = M'(B',w')$ if either $B \neq B'$ or $w \neq w'$, it is the case that $f_B(B) = \mathbb{M}(\beta)(B) + 1$ and $f_B(C) = \mathbb{M}(\gamma)(C)$ if $C \neq B$. Hence $f_B = \mathbb{M}(B) \oplus \mathbb{M}(\beta)$ and so $\mathbb{M}(B) \oplus \mathbb{M}(\beta) \in \lceil \mathbb{M}(w) \oplus M \rceil$ which implies $\alpha' \succcurlyeq_{\mathrm{S}} \gamma'$ and concludes the proof of this case.

Claim 5: $X \to AB \in G$, $B \in \mathcal{N}^{\neg\mathrm{com}}$. *(trivial)*

Then $X\beta\alpha_0 \to_{\mathrm{seq}} AB\beta\alpha_0$ and $\gamma_1 = ABM$. Clearly $AB\beta\alpha_0 = \alpha' \in [\![ABM\gamma_0]\!] = [\![\gamma']\!]$ and so $\alpha' \succcurlyeq_{\mathrm{S}} \gamma'$.

The claim holds for all cases which concludes the proof. $\qquad\square$

Proposition 4 (Concurrent Simulation). $\preccurlyeq_C$ *is a weak simulation relation.*

Proof. Let $\Pi_1 \in \mathbb{M}[Control^{\mathbb{M}}]$, $\Pi_2 \in \mathbb{M}[Control]$ and $\Gamma, \Gamma' \in Chan \to \mathbb{M}[Msg]$ such that $\Pi_1 \lhd \Gamma \preccurlyeq_C \Pi_1' \lhd \Gamma$ and suppose that $\Pi_1 \lhd \Gamma \to_{\mathrm{con}'} \Pi_1' \lhd \Gamma'$.

We will prove that there exists a Π_2' such that $\Pi_1' \lhd \Gamma' \preccurlyeq_C \Pi_2' \lhd \Gamma'$ and $\Pi_2 \lhd \Gamma \to_{\mathrm{con}}^{*} \Pi_2' \lhd \Gamma'$ by case analysis on the rule used for $\Pi_1 \lhd \Gamma \to_{\mathrm{con}'} \Pi_1' \lhd \Gamma'$.

- Rule 11

  Follows immediately by Proposition 3.
- Rule 12

  Then $\Pi_1 = (c\,?\,m)\gamma \parallel \Pi_1^0$ and $\Gamma = \Gamma' \oplus \Gamma(c\,!\,m)$ and $\Pi_1' = \gamma \parallel \Pi_1^0$. Hence $\Pi_2 = (c\,?\,m)\alpha \parallel \Pi_2^0 \in [\![\Pi_1]\!]$ with $\alpha \in [\![\gamma]\!]$ and so $\alpha \parallel \Pi_2^0 \in [\![\Pi_1']\!]$. And using rule 3 $(c\,?\,m)\,\alpha \parallel \Pi_2^0 \lhd \Gamma' \oplus \Gamma(c\,!\,m) \to_{\mathrm{con}} \alpha \parallel \Pi_2^0 \lhd \Gamma'$.

- Rule 13

  Then $\Pi_1 = (\nu X)\,\gamma \parallel \Pi_1^0$, $\Pi_1' = \gamma \parallel X \parallel \Pi_1^0$ and $\Gamma' = \Gamma$. Hence $\Pi_2 = (\nu X)\,\alpha \parallel \Pi_2^0 \in [\![\Pi_1]\!]$ with $\alpha \in [\![\gamma]\!]$ and so $\alpha \parallel X \parallel \Pi_2^0 \in [\![\Pi_1']\!]$. By rule 6 $(\nu X)\,\alpha \parallel \Pi_2^0 \lhd \Gamma \rightarrow_{\mathrm{con}} \alpha \parallel X \parallel \Pi_2^0 \lhd \Gamma$.

- Rule 14

  Then $\Pi_1 = (c_j\,!\,m)\,\gamma \parallel \Pi_1^0$, $\Pi_1' = \gamma \parallel \Pi_1^0$ and $\Gamma' = \Gamma \oplus \Gamma(c\,!\,m)$. Then $\Pi_2 = (c_j\,!\,m)\,\alpha \parallel \Pi_2^0 \in [\![\Pi_1]\!]$ with $\alpha \in [\![\gamma]\!]$ and so $\alpha \parallel \Pi_2^0 \in [\![\Pi_1']\!]$. By rule 14 we can see $(c_j\,!\,m)\,\alpha \parallel \Pi_2^0 \lhd \Gamma \rightarrow_{\mathrm{con}} \alpha \parallel \Pi_2^0 \lhd \Gamma \oplus \Gamma(c\,!\,m)$.

- Rule 15

  Then $\Pi_1 = l\,\gamma \parallel \Pi_1^0$, $\Pi_1' = \gamma \parallel \Pi_1^0$ and $\Gamma' = \Gamma$. Then $\Pi_2 = l\,\alpha \parallel \Pi_2^0 \in [\![\Pi_1]\!]$ with $\alpha \in [\![\gamma]\!]$ and $\alpha \parallel \Pi_2^0 \in [\![\Pi_1']\!]$. By rule 5 $l\,\alpha \parallel \Pi_2^0 \lhd \Gamma \rightarrow_{\mathrm{con}} \alpha \parallel \Pi_2^0 \lhd \Gamma$.

- Rule 16

  Then $\Pi_1 = M\,X\,\gamma \parallel \Pi_1^0$ such that $M \in \mathit{TermCache}$, $\Gamma' = \Gamma \oplus \Gamma(M)$, $X \in \mathcal{N}^{\neg\mathrm{com}}$ and $\Pi_1' = X\,\gamma \parallel \Pi_1^0 \parallel \Pi(M)$. Also $\Pi_2 = \beta\,X\,\alpha \parallel \Pi_2^0 \in [\![\Pi_1]\!]$ and $\beta\,X\,\alpha \in [\![M\,X\,\gamma]\!]$.

  Hence $X\,\alpha \in [\![X\,\gamma]\!]$ and $\beta \in [\![M]\!]$ and hence $\mathbb{M}(\beta) = \lceil M \rceil$. Thus $\beta \rightarrow_{\mathrm{seq}}^* w$, $w \in \Sigma^{\mathrm{com}*}$ such that $\mathbb{M}(w) = M$. Hence using rules 4, 6 and 2 repeatedly we can see that $\beta\,X\,\alpha \parallel \Pi_2^0 \lhd \Gamma \rightarrow_{\mathrm{con}}^* X\,\alpha \parallel \Pi_2^0 \parallel \Pi(w) \lhd \Gamma \oplus \Gamma(w) = X\,\alpha \parallel \Pi_2^0 \parallel \Pi(M) \lhd \Gamma \oplus \Gamma(M) =: \Pi_2'$. and $\Pi_2' \in [\![\Pi_1']\!]$.

- Rule 17

  Then $\Pi_1 = M\,X\,\gamma \parallel \Pi_1^0$ and $\Pi_1' = M'\,X\,\gamma \parallel \Pi_1^0 \parallel \Pi(M)$ such that $m \in \mathit{MixedCache}$, $M' \in \mathit{NonTermCache}$, $\Gamma' = \Gamma \oplus \Gamma(M)$ and $X \in \mathcal{N}^{\neg\mathrm{com}}$.

  Also $\Pi_2 = \beta\,X\,\alpha \parallel \Pi_2^0 \in [\![\Pi_1]\!]$ and $\beta\,X\,\alpha \in [\![M\,X\,\gamma]\!]$. Thus $\beta \in [\![M]\!]$ and hence $\beta \rightarrow_{\mathrm{seq}}^* w$, $w \in (\Sigma^{\mathrm{com}} \cup \mathcal{N}^{\mathrm{com}})^*$ such that $\mathbb{M}(w) = M$. Then $w \simeq_I w_0 w_1$ such that $w_0 \in \Sigma^{\mathrm{com}*}$ and $w_1 \in \mathcal{N}^{\mathrm{com}}$ and $M' = \mathbb{M}(w_1)$.

  Hence $w_1\,X\,\alpha \in [\![M'\,X\,\gamma]\!]$ and thus $w_1 X\,\alpha \parallel \Pi_2^0 \oplus \Pi(M) \lhd \Gamma \oplus \Gamma(M) := \Pi_2' \in [\![\Pi_1']\!]$.

  Using rules 4, 6 and 2 repeatedly we can see that $\beta\,X\,\alpha \parallel \Pi_2^0 \lhd \Gamma \rightarrow_{\mathrm{con}}^* w_1 X\,\alpha \parallel \Pi_2^0 \parallel \Pi(w_0) \lhd \Gamma \oplus \Gamma(w_0) = w_1 X\,\alpha \parallel \Pi_2^0 \parallel \Pi(w) \lhd \Gamma \oplus \Gamma(w) = w_1 X\,\alpha \parallel \Pi_2^0 \parallel \Pi(M) \lhd \Gamma \oplus \Gamma(M) = \Pi_2'$.

Hence the claim holds in all cases and thus we can conclude that $\preccurlyeq_C$ is a weak simulation relation. $\qquad\square$

Corollary 2. *Given an ACPS P if $S \lhd \Gamma(\epsilon) \rightarrow_{con'}^* \Pi \lhd \Gamma$ then $S \lhd \Gamma(\epsilon) \rightarrow_{con}^* \Pi' \lhd \Gamma$ such that $\Pi \lhd \Gamma \preccurlyeq_S \Pi' \lhd \Gamma$.*

Proof. Follows trivially by induction from Proposition 4. $\qquad\square$

Theorem 1 (Reduction of Program-Point Coverability). *$(P; l_1, \ldots, l_n)$ is a yes-instance of Program-Point Coverabililty problem iff $(P; l_1, \ldots, l_n)$ is a yes-instance of Alternative Program-Point Coverability problem.*

Proof. We will first prove the $\Rightarrow$-direction. Let $(P; l_1, \ldots, l_n)$ be a yes-instance of the Program-Point Coverabililty problem then a configuration $l_1 \alpha_1 \parallel \cdots \parallel l_n \alpha_n \parallel \Pi \lhd \Gamma$ for some $\alpha_1, \ldots, \alpha_n \in (\Sigma \cup \mathcal{N})^*/_{\simeq_I}$ is $\rightarrow_{\mathrm{con}}$ reachable. By Proposition 2 $\overline{\mathbb{M}}(l_1 \alpha_1 \parallel \cdots \parallel l_n \alpha_n) \parallel \overline{\mathbb{M}}(\Pi) \lhd \Gamma$ is reachable for $\rightarrow_{\mathrm{con'}}$ and thus $(P; l_1, \ldots, l_n)$ is a yes-instance of the Alternative Program-Point Coverabililty problem.

For the $\Leftarrow$-direction let $(P; l_1, \ldots, l_n)$ be a yes-instance of the Alternative Program-Point Coverabililty problem. Then a configuration $\gamma_1 \parallel \cdots \parallel \gamma_n \parallel \Pi \lhd \Gamma$ is $\to_{\mathrm{con}'}$ reachable and for $i = 1, \ldots, n$ either $\gamma_i = l_i \gamma_i'$ or $\gamma_i = M_i \gamma_i'$ such that $l_i \in M_i$. By Proposition 4 we can conclude that $\alpha_1 \parallel \cdots \parallel \alpha_n \parallel \Pi' \lhd \Gamma$ is $\to_{\mathrm{con}}$ reachable such that $\gamma_1 \parallel \cdots \parallel \gamma_n \parallel \Pi \lhd \Gamma \preccurlyeq_{\mathrm{C}} \alpha_1 \parallel \cdots \parallel \alpha_n \parallel \Pi' \lhd \Gamma$. That means for $i = 1, \ldots, n$ either $\alpha_i = l_i \alpha_i'$ or $\alpha_i = \beta_i \alpha_i'$ such that $\beta_i \in \mathcal{N}^{\mathrm{com}*}$ and $\beta_i \to_{\mathrm{seq}}^* w_i^0 w_i^1$ such that $\mathbb{M}(w_i^1) = M_i$ and $w_i^0 \in \Sigma^{\mathrm{com}*}$. It follows, by using $\simeq_I$ where necessary and choosing rewrite rules to expose l_i, that $\beta_i \to_{\mathrm{seq}}^* w'^0_i l_i \beta_i'$ where $w'^0_i \in \Sigma^{\mathrm{com}*}$ and $\beta_i \in \mathcal{N}^{\mathrm{com}*}$. Hence $\alpha_1 \parallel \cdots \parallel \alpha_n \parallel \Pi' \lhd \Gamma \to_{\mathrm{con}}^* \alpha_1'' \parallel \cdots \parallel \alpha_n'' \parallel \Pi' \lhd \Gamma \oplus \Gamma(w'^0_0 \cdots w'^0_n)$ where either $\alpha_i'' = \alpha_i = l_i \alpha_i$ and $w'^0_i = \epsilon$ or $\alpha_i'' = l_i \beta_i'$. Thus we can conclude that $(P; l_1, \ldots, l_n)$ is a yes-instance for the Program-Point Coverability problem. $\square$

B   Proof of Lemma 1

Lemma 9 (Sequential Monotonicity). *The transition relation $\to_{\mathrm{seq}'}$ is monotone with respect to $\leq_{Control^{\leq k}}$.*

Proof. Suppose $\gamma, \gamma', \delta \in Control$ such that $\gamma \leq \delta$ and $\gamma \to_{\mathrm{seq}'} \gamma'$. We will show that there $\exists \delta'$ such that $\delta \to_{\mathrm{seq}'} \delta'$ and $\gamma \leq \delta'$. We conclude from the definition of $\leq_{Control^{\leq k}}$ on $Control^{\leq k}$ and the fact that $\gamma \to_{\mathrm{seq}'} \gamma'$ that $\gamma = X_1 M_1 X_2 M_2 \cdots X_j M_j$ and $\delta = X_1 M_1' X_2 M_2' \cdots X_j M_j'$ with $M_i \leq_{Cache} M_i'$ for $1 \leq i \leq j \leq k$. Our proof will be by case analysis on $\gamma \to_{\mathrm{seq}'} \gamma'$.

- $\gamma \to_{\mathrm{seq}'} \gamma'$ using Rule 7.
  Thus there is a $X_1 \to BC$ rule and $C \to_{\mathrm{seq}}^* w$ and $\gamma' = B(\mathbb{M}(w) \oplus M_1) X_2 M_2 \cdots X_j M_j$. Hence $\delta \to_{\mathrm{seq}'} B(\mathbb{M}(w) \oplus M_1') X_2 M_2' \cdots X_j M_j' =: \delta'$. Clearly $(\mathbb{M}(w) \oplus M_1) \leq_{Cache} (\mathbb{M}(w) \oplus M_1')$ and thus $\gamma' \leq_{Control^{\leq k}} \delta'$.
- $\gamma \to_{\mathrm{seq}'} \gamma'$ using Rule 8.
  Thus there is a $X_1 \to BC$ rule, $C \in \mathcal{N}^{\neg\mathrm{com}}$ and $\gamma' = B C M_1 X_2 M_2 \cdots X_j M_j$. Further since γ' in $Control^{\leq k}$ it is the case that $j < k$. Hence $\delta \to_{\mathrm{seq}'} B C M_1' X_2 M_2' \cdots X_j M_j' =: \delta', \delta \in Control^{\leq k}$ since $j < k$ and obviously $\gamma' \leq_{Control^{\leq k}} \delta'$.
- $\gamma \to_{\mathrm{seq}'} \gamma'$ using Rule 9.
  Thus there is a $X_1 \to aB$ rule, $a \in \Sigma \cup \{\epsilon\}$ and $\gamma' = a B M_1 X_2 M_2 \cdots X_j M_j$. Hence $\delta \to_{\mathrm{seq}'} a B M_1' X_2 M_2' \cdots X_j M_j' =: \delta'$ and obviously $\gamma' \leq_{Control^{\leq k}} \delta'$.
- $\gamma \to_{\mathrm{seq}'} \gamma'$ using Rule 10.
  Thus there is a $X_1 \to a$ rule, $a \in \Sigma \cup \{\epsilon\}$ and $\gamma' = a M_1 X_2 M_2 \cdots X_j M_j$. Hence $\delta \to_{\mathrm{seq}'} a M_1' X_2 M_2' \cdots X_j M_j' =: \delta'$ and obviously $\gamma' \leq_{Control^{\leq k}} \delta'$.

$\square$

Lemma 10 (Monotonicity). *The transition relation $\to_{\mathrm{con}'}$ is monotone with respect to $\leq_{Config}$.*

Proof. Suppose $\Pi_1 \lhd \Gamma_1, \Pi_1' \lhd \Gamma_1', \Pi_2 \lhd \Gamma_2 \in Config$ such that $\Pi_1 \lhd \Gamma_1 \leq_{Config} \Pi_2 \lhd \Gamma_2$ and $\Pi_1 \lhd \Gamma_1 \to_{\mathrm{con}'} \Pi_1' \lhd \Gamma_1'$. We will show that there $\exists \Pi_2'$ such that $\Pi_2 \lhd \Gamma_2 \to_{\mathrm{con}'} \Pi_2' \lhd \Gamma_2'$ and $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$. Since $\Pi_1 \lhd \Gamma_1 \leq_{Config} \Pi_2 \lhd \Gamma_2$ and

$\Pi_1 \lhd \Gamma \to_{\text{con}'} \Pi_1' \lhd \Gamma'$, we can infer the components of the configuration involved in the latter transition. That means $\Pi_1 = \gamma \parallel \Pi_1^0$ and $\Pi_2 = \delta \parallel \Pi_2^0$ such that $\Pi_1^0 \leq_{\mathbb{M}[Control^{\leq k}]} \Pi_2^0$, $\Gamma_1 \leq_{Queues} \Gamma_2$, $\gamma, \delta \in Control^{\leq k}$ and $\gamma \leq_{Control^{\leq k}} \delta$. Our proof will be by case analysis on $\Pi_1 \lhd \Gamma \to_{\text{con}'} \Pi_1' \lhd \Gamma_1'$.

- $\Pi_1 \lhd \Gamma_1 \to_{\text{con}'} \Pi_1' \lhd \Gamma_1'$ using 11.
  This follows immediately by Lemma 9.
- $\Pi_1 \lhd \Gamma_1 \to_{\text{con}'} \Pi_1' \lhd \Gamma_1'$ using 12.
  Thus we can conclude (i) $\gamma = c\,?\,m\gamma'$, (ii) $\Pi_1' = \gamma' \parallel \Pi_1^0$, (iii) $\Gamma_1 = \Gamma_1' \oplus \Gamma(c\,!\,m)$. Further since $\Pi_1 \lhd \Gamma_1 \leq_{Config} \Pi_2 \lhd \Gamma_2$ we infer (iv) $\delta = c\,?\,m\delta'$ with $\gamma' \leq_{Control^{\leq k}} \delta'$ and (v) $\Gamma_2 = \Gamma_2' \oplus \Gamma(c\,!\,m)$.
  Then we have $\Pi_2 \lhd \Gamma_2 \to_{\text{con}'} \delta' \parallel \Pi_2^0 \lhd \Gamma_2'$.
  Writing $\Pi_2' := \delta' \parallel \Pi_2^0$ it remains to show $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$.
  Now (a) $\gamma' \leq_{\mathbb{M}[Control^{\leq k}]} \delta'$, (b) $\Pi_1^0 \leq_{\mathbb{M}[Control^{\leq k}]} \Pi_2^0$ by assumption and (c) since $\Gamma_1 \leq_{Queue} \Gamma_2$ and clearly $\Gamma_1' \leq_{Queue} \Gamma_2'$.
  Hence we conclude $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$.
- $\Pi_1 \lhd \Gamma_1 \to_{\text{con}'} \Pi_1' \lhd \Gamma_1'$ using 13.
  Thus we can conclude (i) $\gamma = (\nu X)\,\gamma'$, (ii) $\Pi_1' = \gamma' \parallel \Pi_1^0 \parallel X$, (iii) $\Gamma_1' = \Gamma_1$. Further since $\Pi_1 \lhd \Gamma_1 \leq_{Config} \Pi_2 \lhd \Gamma_2$ we infer (iv) $\delta = (\nu X)\,\delta'$ with $\gamma' \leq_{Control^{\leq k}} \delta'$.
  Then we have $\Pi_2 \lhd \Gamma_2 \to_{\text{con}'} \delta' \parallel \Pi_2^0 \parallel X \lhd \Gamma_2$.
  Writing $\Pi_2' := \delta' \parallel \Pi_2^0 \parallel X$ it remains to show $\Pi_1' \lhd \Gamma_1 \leq_{Config} \Pi_2' \lhd \Gamma_2$.
  Now (a) $\gamma' \leq_{\mathbb{M}[Control^{\leq k}]} \delta'$, (b) $\Pi_1^0 \leq_{\mathbb{M}[Control^{\leq k}]} \Pi_2^0$ by assumption and (c) clearly $X \leq_{\mathbb{M}[Control^{\leq k}]} X$.
  Hence we conclude $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$.
- $\Pi_1 \lhd \Gamma_1 \to_{\text{con}'} \Pi_1' \lhd \Gamma_1'$ using 14.
  Thus we can conclude (i) $\gamma = c\,!\,m\,\gamma'$, (ii) $\Pi_1' = \gamma' \parallel \Pi_1^0$, (iii) $\Gamma_1' = \Gamma_1 \oplus \Gamma(c\,!\,m)$. Further since $\Pi_1 \lhd \Gamma_1 \leq_{Config} \Pi_2 \lhd \Gamma_2$ we infer (iv) $\delta = c\,!\,m\,\delta'$ with $\gamma' \leq_{Control^{\leq k}} \delta'$.
  Then we have $\Pi_2 \lhd \Gamma_2 \to_{\text{con}'} \delta' \parallel \Pi_2^0 \lhd \Gamma_2 \oplus \Gamma(c\,!\,m)$.
  Writing $\Pi_2' := \delta' \parallel \Pi_2^0$ and $\Gamma_2' := \Gamma_2 \oplus \Gamma(c\,!\,m)$ it remains to show $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$.
  Now (a) $\gamma' \leq_{\mathbb{M}[Control^{\leq k}]} \delta'$, (b) $\Pi_1^0 \leq_{\mathbb{M}[Control^{\leq k}]} \Pi_2^0$ by assumption and (c) since $\Gamma_1 \leq_{Queues} \Gamma_2$, $\oplus$ and $\Gamma(\cdot)$ monotonic we have $\Gamma_1 \oplus \Gamma(c\,!\,m) \leq_{Queues} \Gamma_2 \oplus \Gamma(c\,!\,m)$.
  Hence we conclude $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$.
- $\Pi_1 \lhd \Gamma_1 \to_{\text{con}'} \Pi_1' \lhd \Gamma_1'$ using 15.
  Thus we can conclude (i) $\gamma = l\,\gamma'$, (ii) $\Pi_1' = \gamma' \parallel \Pi_1^0$, (iii) $\Gamma_1' = \Gamma_1$. Further since $\Pi_1 \lhd \Gamma_1 \leq_{Config} \Pi_2 \lhd \Gamma_2$ we infer (iv) $\delta = l\,\delta'$ with $\gamma' \leq_{Control^{\leq k}} \delta'$.
  Then we have $\Pi_2 \lhd \Gamma_2 \to_{\text{con}'} \delta' \parallel \Pi_2^0 \lhd \Gamma_2$.
  Writing $\Pi_2' := \delta' \parallel \Pi_2^0$ it is trivial to see $\Pi_1' \lhd \Gamma_1 \leq_{Config} \Pi_2' \lhd \Gamma_2$.
- $\Pi_1 \lhd \Gamma_1 \to_{\text{con}'} \Pi_1' \lhd \Gamma_1'$ using 16.
  Thus we can conclude (i) $\gamma = M_1 X \gamma_1$, (ii) $\Pi_1' = \gamma' \parallel \Pi_1^0 \parallel \Pi(M_1)$, (iii) $\gamma' = X\gamma_1$ and (iv) $\Gamma_1' = \Gamma_1 \oplus \Gamma(M_1)$. Further since $\Pi_1 \lhd \Gamma_1 \leq_{Config} \Pi_2 \lhd \Gamma_2$ we infer (v) $\delta = M_2 X \delta_1$ with $M_1 \leq_{Cache} M_2$ and $X\gamma_1 \leq_{Control^{\leq k}} X\delta_1$.
  Let $\delta' := X\delta_1$ then clearly $\gamma' \leq_{Control^{\leq k}} \delta'$ and we have $\Pi_2 \lhd \Gamma_2 \to_{\text{con}'} \delta' \parallel \Pi_2^0 \parallel \Pi(M_2) \lhd \Gamma_2 \oplus \Gamma(M_2)$.

Writing $\Pi_2' := \delta' \parallel \Pi_2^0 \parallel \Pi(M_2)$ and $\Gamma_2' := \Gamma_2 \oplus \Gamma(M_2)$ we will now show $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$.

Now (a) $\gamma' \leq_{\mathbb{M}[Control^{\leq k}]} \delta'$, (b) $\Pi_1^0 \leq_{\mathbb{M}[Control^{\leq k}]} \Pi_2^0$ by assumption, (c) since $M_1 \leq_{Cache} M_2$ and since $\Pi(\cdot)$ is clearly monotonic $\Pi(M_1) \leq_{\mathbb{M}[Control^{\leq k}]} \Pi(M_2)$ and (d) lastly since $\Gamma(\cdot)$ is monotonic we can conclude $\Pi(M_1) \leq_{Queues} \Pi(M_2)$. Hence we conclude $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$.

- $\Pi_1 \lhd \Gamma_1 \rightarrow_{con'} \Pi_1' \lhd \Gamma_1'$ using 17.
  Thus we can conclude (i) $\gamma = M_1 X \gamma_1$ with $M_1 \in MixedCache$, (ii) $\Pi_1' = \gamma' \parallel \Pi_1^0 \parallel \Pi(M_1)$, (iii) $\gamma' = M_1' X \gamma_1$ with $M_1' \in NonTermCache$ and $M_1' = M_1 \upharpoonright (\mathcal{N} \cup \mathcal{L})$, (iv) $\Gamma_1' = \Gamma_1 \oplus \Gamma(M_1)$. Further since $\Pi_1 \lhd \Gamma_1 \leq_{Config} \Pi_2 \lhd \Gamma_2$ we infer (v) $\delta = M_2 X \delta_1$ with $M_1 \leq_{Cache} M_2$ and $X \gamma_1 \leq_{Control^{\leq k}} X \delta_1$.
  Let $M_2' := M_2 \upharpoonright (\mathcal{N} \cup \mathcal{L})$ and $\delta' := M_2' X \delta_1$ then we have $\Pi_2 \lhd \Gamma_2 \rightarrow_{con'} \delta' \parallel \Pi_2^0 \parallel \Pi(M_2) \lhd \Gamma_2 \oplus \Gamma(M_2)$.
  Writing $\Pi_2' := \delta' \parallel \Pi_2^0 \parallel \Pi(M_2)$ and $\Gamma_2' := \Gamma_2 \oplus \Gamma(M_2)$ we will now show $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$.
  Now since $\cdot \upharpoonright \cdot$ is monotonic in the first argument and $M_1 \leq_{Cache} M_2$ we conclude $M_1' \leq_{Cache} M_2'$ and thus (a) $\gamma' \leq_{\mathbb{M}[Control^{\leq k}]} \delta'$, (b) $\Pi_1^0 \leq_{\mathbb{M}[Control^{\leq k}]} \Pi_2^0$ by assumption, (c) and since $\Pi(\cdot)$ is monotonic $\Pi(M_1) \leq_{\mathbb{M}[Control^{\leq k}]} \Pi(M_2)$ and (d) lastly since $\Gamma(\cdot)$ is monotonic we can conclude $\Pi(M_1) \leq_{Queues} \Pi(M_2)$. Hence we conclude $\Pi_1' \lhd \Gamma_1' \leq_{Config} \Pi_2' \lhd \Gamma_2'$.

$\square$

Corollary 3. *The transition system* $(\mathbb{M}[Control^{\leq k}] \times (Chan \rightarrow \mathbb{M}[Msg]), \rightarrow_{con'}, \leq)$ *is a well-structured transition system.*

Proof. Follows immediately from Lemma 10. $\square$

Theorem 3. *The Program-Point Coverability problem for unbounded spawning k-ACPS is decidable.*

Proof. By Theorem 1 it suffices to show that the Alternative Program-Point Coverability problem is decidable, which follows from Corollary 3 and the fact that the set

$$U := \uparrow \{\widehat{l_1} X_1^1 \cdots X_{j_1}^1 \parallel \cdots \parallel \widehat{l_n} X_n^1 \cdots X_n^{j_n} \lhd \Gamma([]) \mid X_i^j \in \mathcal{N}^{\neg com}, \widehat{l_i} = l_i \text{ or } [l_i] \text{ and } 0 \leq j_i \leq k\}$$

is upward-closed and $(P; l_1, ..., l_n)$ is a yes-instance for the Alternative Program-Point Coverability problem iff an element of U is $\rightarrow_{con'}$-reachable. $\square$

C   Notation

M	multiset
μ, ν	general sequence
m	message
c	channel
A, B, C, X, Y, Z	non-terminal i.e. element of $\mathcal{N}$
l	label
a	terminal i.e. element of Σ
w	word over Σ
α, β	word over $(\Sigma \cup \mathcal{N})$
γ, δ	word in $Control^{\mathbb{M}}$
i, j, n, k	integer
Γ	channel
Π	set of processes
U, V, W	set
I	Independence relation
D	dependence relation
R	relation
u, v	general element
$\rightarrow_{\mathrm{seq}}$	standard sequential semantics
$\rightarrow_{\mathrm{con}}$	standard concurrent semantics
$\rightarrow_{\mathrm{seq'}}$	alternative sequential semantics
$\rightarrow_{\mathrm{con'}}$	alternative concurrent semantics