Constrained Dynamic Tree Networks*

Matthew Hague and Vincent Penelle

[1] Royal Holloway, University of London
[2] Royal Holloway, University of London and Université Paris-Est, LIGM (CNRS UMR 8049), UPEM, CNRS

Abstract

We generalise Constrained Dynamic Pushdown Networks, introduced by Bouajjani *et al.*, to Constrained Dynamic Tree Networks. In this model, we have trees of processes which may monitor their children. We allow the processes to be defined by any computation model for which the alternating reachability problem is decidable. We address the problem of symbolic reachability analysis for this model. More precisely, we consider the problem of computing an effective representation of their reachability sets using finite state automata. We show that backward reachability sets starting from regular sets of configurations are always regular. We provide an algorithm for computing backward reachability sets using tree automata.

1 Introduction

Bouajjani *et al.* [2] defined Constrained Dynamic Networks of Pushdown Systems: a model of concurrent computation where configurations of processes are tree structures, and each process is given by a pushdown system. During an execution, new child processes can be created, and a parent can test the states of its children before performing an execution step. They considered the global backwards reachability problem for these systems. That is, given a regular set of target configurations, compute the set of configurations that can reach the target set. They showed that, under a *stability* constraint, this backwards reachability set is regular and computable.

The stability constraint requires that once a test a parent may make on its children is satisfied, then it will remain satisfied, even if the children continue their execution. In the simplest case, this allows a parent to test for termination in a given state of its children. In general, this constraint allows a parent to (repeatedly) test whether its children have passed certain stages of execution (and their state in doing so).

We show here that Bouajjani *et al.*'s result is not dependent on the processes in the tree being modelled by pushdown systems. In fact, all that is required is that the *alternating reachability problem* is decidable for the systems labelling the nodes in the tree. Intuitively, in the alternating reachability problem, some steps during the run of the system may be required to split into separate paths. From the initial state, we ask whether all paths of the execution reach a given final state.

Thus, we introduce *Constrained Dynamic Tree Networks*, which are tree networks of processes as before, but the individual processes can be labelled by any system for which the alternating reachability problem is decidable.

One particular instance of interest is the case of networks of *collapsible pushdown systems* [13]. Collapsible pushdown systems are a generalisation of pushdown systems that are known to be equi-expressive with *Higher-Order Recursion Schemes*. The alternating reachability problem is known to be decidable for these systems [29]. In fact, the backwards reachability

*This work was supported by the Engineering and Physical Sciences Research Council [EP/K009907/1].

sets of alternating collapsible pushdown systems are also known to be computable and regular [4]. Thus, we obtain a new model of concurrent higher-order programs for which the backwards reachability sets are also computable and regular. An advantage of our approach is that we do not need to consider the technical difficulties of reasoning about collapsible pushdown systems. The proof presented here only needs to take care of the concurrent aspects of the computations. Thus, we obtain results for quite complex systems with a relatively modest proof.

Modern day programming increasingly embraces higher-order programming, both via the inclusion of higher-order constructs in languages such as C++, JavaScript and Python, but also via the importance of *callbacks* in highly popular technologies such as jQuery and Node.js. For example, to read a file in Node.js, one would write

```
fs.readFile('f.txt', function (err, data) { ..use data.. });
```

In this code, the call to `readFile` spawns a new thread that asynchronously reads `f.txt` and sends the `data` to the function argument. This function will have access to, and frequently use, the closure information of the scope in which it appears. The rest of the program runs *in parallel* with this call. This style of programming is fundamental to both jQuery and Node.js programming, as well as being a popular for programs handling input events or slow IO operations such as fetching remote data or querying databases (e.g. HTML5's indexedDB).

Analysing such programs is a challenge for verification tools which usually do not model higher-order recursion, or closures, accurately. However, several higher-order model-checking tools have been recently developed. This trend was pioneered by Kobayashi *et al.* [16]. The feasibility of higher-order model-checking in practice has been demonstrated by numerous higher-order model-checking tools [15, 17, 19, 27, 5, 6, 31]. Since all of these tools can handle the alternating reachability problem, it is possible that our techniques may be used to provide model checking tools for concurrent higher-order programs.

Our construction follows Bouajjani *et al.* and uses a *saturation* method to construct a regular representation of the backwards reachability set. However, our automaton representation is different: it separates the representation of the system states from the tree structure. We also use different techniques to prove correctness of the construction. In particular, our soundness proof works by defining and showing soundness of each transition of the automaton, rather than dissecting complete runs. This is an application of a technique first used for a saturation technique for solving parity games over pushdown systems [14].

1.1 Related Work

Dynamic pushdown networks have been well-studied. Often these are studied without the process tree structure or constraints allowing a parent to inspect its children [24, 35]. Various decidability-preserving locking techniques have also been investigated [22]. Some of these works also allow the tree structure to be taken into account [21, 28]. Touili and Atig have also considered communication structures that are not necessarily trees [37]. However, these works consider pushdown networks only.

There has been some work studying concurrent variants of recursion scheme model checking, including a context-bounded algorithm for recursion schemes [18], and further underapproximation methods such as phase-bounded, ordered, and scope-bounding [11, 34]. These works allow only a fixed number of threads.

Dynamic thread creation is permitted by both Yasukata *et al.* [38] and by Chadha and Viswanathan [7]. In Yasukata *et al.*'s model, recursion schemes may spawn and join threads.

Communication is permitted only via nested locks. Their work is a generalisation of results for order-1 pushdown systems [10]. Chadha and Viswanathan allow threads to be spawned, but only one thread runs at a time, and must run to completion. Moreover, the tree structure is not maintained.

The saturation technique was popularised by Bouajjani *et al.* [1] for the analysis of pushdown systems, which was implemented in the successful Moped tool [33, 36]. Saturation methods also exist for *ground tree rewrite systems* and related systems [23, 3, 25], though use different techniques.

Ground tree rewrite systems may also be generalised to trees where the nodes are labelled by higher-order stacks. Penelle proves decidability of first order logic with reachability over such systems [30]. However, this result does not allow nodes to have an unbounded number of direct children, and does not consider collapsible stacks in their full generality.

Finally, there is various research into meta-results on the analysis of concurrent systems, where the concurrent structure is the object of the research. Recent work by La Torre *et al.* has shown that parameterised safety analysis is possible of asynchronous networks of shared-memory systems [20], provided, amongst other constraints, the downwards closure of the system is computable. Collapsible pushdown systems are known to have these properties [8, 12, 39]. Other works have studied multi-stack pushdown systems and offered either bounded tree-width [26], or split-width [9], as explanations for decidability. However, these results have not been extended to higher orders.

2 Alternating transition system

We define the notion of alternating transition system. Given a (possibly infinite) set of elements Γ and operations θ over Γ, and a finite set of states $\mathbb{S}$ and final states $\mathbb{F}$, an alternating transition system has transitions of the form $s \xrightarrow{\theta} S$. A label γ is always accepted from a final state. When we apply an alternating transition $s \xrightarrow{\theta} S$, to accept an element γ from s, we need to accept $\theta(\gamma)$ from every state in S.

In the following, we consider Γ to be a set, and Ops a set of operations $\theta : \Gamma \to \Gamma$ over Γ. Note, we do not require that θ is defined over all elements of Γ. We also define the special operation Id such that $\mathrm{Id}(\gamma) = \gamma$ for all $\gamma \in \Gamma$.

Definition 1 (Alternating transition systems over Γ, Ops)**.** *An alternating transition system over Γ, Ops is a tuple $\mathcal{N} = (\mathbb{S}, \mathbb{F}, \eta)$, where:*

- *$\mathbb{S}$ is a finite set of states,*
- *$\mathbb{F} \subseteq \mathbb{S}$ is the set of final states,*
- *$\eta \subseteq \mathbb{S} \times \mathrm{Ops} \times 2^{\mathbb{S}}$ is the set of transitions.*

Given $\gamma \in \Gamma$ and $s \in \mathbb{S}$, we say that γ is accepted from s, denoted $\gamma \vdash s$, if s is final, or if there is a transition $\nu = (s, \theta, S)$ such that $\theta(\gamma)$ is defined and $\theta(\gamma) \vdash s'$ for every $s' \in S$.

In all the following, we suppose that for every alternating transition system $\mathcal{N}$, given $\gamma \in \Gamma$ and $s \in \mathbb{S}$, we can decide whether $\gamma \vdash s$.

Example 1. *An alternating pushdown system with stack alphabet Σ is an alternating transition system with $\Gamma = \Sigma^*$ and the set of operations $\mathrm{Ops} = \{(a, u) \mid a \in \Sigma, u \in \Sigma^*\}$. where for all*

$w \in \Sigma^*$

$$(a,u)(w) = \begin{cases} uv & w = av \\ \mathit{undefined} & \mathit{otherwise.} \end{cases}$$

Note here that we represent a stack as a word, with the top of the stack appearing on the left.

3 Constrained Dynamic Tree Networks

We define constrained dynamic tree networks (CDTNs), which allow process trees with dynamic thread creation and parents to inspect the states of their children.

Definition 2 (Constrained Dynamic Tree Network over Γ)**.** *A constrained dynamic network over Γ is a tuple $\mathcal{M} = (\mathrm{P}, \mathrm{F}, \delta)$ with:*

- P *is a finite set of states,*
- $\mathrm{F} \subseteq \mathrm{P}$ *is the set of final states,*
- δ *a finite set of transitions of the following form:*

  C1 $\phi : p \xrightarrow{\theta} p_a$*, with* $\theta \in \mathrm{Ops}, p, p_a \in \mathrm{P}$*, and* ϕ *is a regular language over* P*.*

  C2 $\phi : p \to p_a \rhd p_b$*, with* $p, p_a, p_b \in \mathrm{P}$*, and* ϕ *is a regular language over* P*.*

An $\mathcal{M}$-configuration is a tree labelled by $\mathrm{P} \times \Gamma$. That is, a configuration is either a leaf node $(p,\gamma)(\emptyset)$ or a tree $(p,\gamma)(t_1, \cdots, t_m)$ with root (p,γ) and children $t_1, \cdots, t_m$ where $p \in \mathrm{P}$, $\gamma \in \Gamma$, and for each $1 \leq i \leq m$ we have that t_i is an $\mathcal{M}$-configuration. We write $C[t]$ to denote a configuration containing t as a sub-tree. Furthermore, we define

$$S((p,\gamma)(t_1, \cdots, t_m)) = p$$

to extract the *internal* state of the root node of a configuration.

Transitions of the form C1 apply θ to a node, while transitions of the form C2 create a new child process. That is, the application of a transition of the form C1 to a configuration $C[(p,\gamma)(t_1, \cdots, t_m)]$ yields the configuration $C[(p_a, \theta(\gamma))(t_1, \cdots, t_m)]$, if $\theta(\gamma)$ is defined and $S(t_1) \cdots S(t_m) \in \phi$. The application of a transition of the form C2 to a configuration $C[(p,\gamma)(t_1, \cdots, t_m)]$ yields the configuration $C[(p_a,\gamma)(t_1, \cdots, t_m, (p_b,\gamma)(\emptyset))]$ if $S(t_1) \cdots S(t_m) \in \phi$.

3.1 Stability constraint

We give the restriction on child constraints ϕ that allows us to preserve decidability of reachability for constrained dynamic tree networks. Intuitively, this constraint asserts that once a constraint ϕ is satisfied, it will remain satisfied even if its children progress. We first define the notion of a *stable* constraint.

Definition 3 (Stability relation)**.** *Given an alphabet Σ and a binary relation ρ over Σ, we say that a language L is ρ-stable if for every $a, b \in \Sigma$, $u, v \in \Sigma^*$, $uav \in L \wedge \rho(a,b) \Rightarrow ubv \in L$.*

Given a CDTN $\mathcal{M} = (\mathrm{P}, \mathrm{F}, \delta)$ we define

$$\rho_\tau = \{(p,p') \mid \exists \phi : p \xrightarrow{\theta} p' \in \delta \vee \exists \phi : p \to p' \rhd p'' \in \delta\} \ .$$

We say $\mathcal{M}$ is ρ_τ-stable iff for all $\phi : p \xrightarrow{\theta} p_a \in \delta$ and $\phi : p \to p_a \rhd p_b \in \delta$ we have ϕ is ρ_τ-stable.

3.2 Automaton

We now define a notion of tree automata over the configurations of a constrained dynamic tree network. As these configurations can have an unbounded arity, we need to have an automaton model which can deal with unbounded arity, thus we use an adapted version of hedge automata. Transitions of our automata are thus of the form $p(L) \to q$, meaning they can rewrite a tree to a state q, if

- the internal state of its root is p,
- the i^{th} son of its root can be rewritten to the state q_i, and
- $q_1 \cdots q_m$ is in the regular language L (if the node has m sons).

Moreover, the automaton as well checks that the element of the root is accepted by an alternating transition system which is bound to the rule (more precisely, we will use a single alternating transition system for the whole automaton, which has a unique initial state for each rule of the automaton). In the following definition, let $\mathrm{Rec}(\mathcal{Q})$ be the set of regular languages over alphabet $\mathcal{Q}$.

Definition 4 ($\mathcal{M}$-automaton)**.** *Given a CDTN $\mathcal{M}$, an $\mathcal{M}$-automaton is a tuple $\mathcal{A} = (\mathcal{Q}, \mathcal{F}, \Delta, \mathcal{N})$, where:*

- *$\mathcal{Q}$ is a final set of states,*
- *$\mathcal{F} \subseteq \mathcal{Q}$ the set of final states,*
- *$\Delta \subseteq \mathrm{P} \times \mathrm{Rec}(\mathcal{Q}) \times \mathcal{Q}$ a finite set of transitions of the form $p(L) \to q$,*
- *$\mathcal{N} = (\mathbb{S}, \mathbb{F}, \eta)$ an alternating transition system over Γ, such that for every $r \in \Delta$, there is a unique state $s_r \in \mathbb{S}$. Without loss of generality, we as well suppose that these states have no incoming transition[1]. We as well supose that these states are not final[2].*

An $\mathcal{A}$-configuration is a tree labelled by $(\mathrm{P} \times \Gamma) \cup \mathcal{Q}$, such that only leaves can be labelled by $\mathcal{Q}$. Let $\mathcal{T}(\mathrm{P} \times \Gamma)$ be the set of such trees. Given a transition $r = p(L) \to q$ and two $\mathcal{A}$-configurations t and t', we have $t \xrightarrow{r} t'$ if and only if $t = C[(p, \gamma)(q_1, \cdots, q_m)]$, $t' = C[q]$, $q_1 \cdots q_m \in L$ and $\gamma \vdash s_r$.

Let $\xrightarrow[\Delta]{*}$ be the transitive closure of $\left(\bigcup_{r \in \Delta} \xrightarrow{r}\right)$. The set of $\mathcal{M}$-configurations recognised by $\mathcal{A}$ from the state q is $\mathcal{L}_q(\mathcal{A}) = \{t \in \mathcal{T}(\mathrm{P} \times \Gamma) \mid t \xrightarrow[\Delta]{*} q\}$.

Example 2. *We can accept regular sets of pushdown networks as defined by Bouajjani et al. [2] by defining the word automata used to recognise pushdown stacks as alternating transition systems with operations of the form (a, ε), where ε is the empty word, and operations have the same semantics as in Example 1. That is, each operation consumes the leftmost character of the word representation of the stack. For this we will need an explicit end-of-stack marker.*

[1] If it is not the case, we create a copy of these states on which we conserve all the transition as an "internal state", and remove the incoming transitions to these states.

[2] If so, for a state s_r, we create a new final state s and add the transition $s_r \xrightarrow{\mathrm{Id}} s$, and remove s_r from the set of final states.

4 Backwards Reachability

In this section, we show that the we can compute the backwards reachability set of CDTNs. That is, if a CDTN $\mathcal{M}$ is ρ_δ-stable, then the set of predecessors of a regular set is regular.

Theorem 1. *Given $\mathcal{M}$ a ρ_δ-stable CDTN and $\mathcal{A}$ an $\mathcal{M}$-automaton, it is possible to compute a $\mathcal{M}$-automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$.*

To prove this theorem, we construct an automaton $\mathcal{A}'$ from $\mathcal{A}$ and $\mathcal{M}$. We construct the automaton in two steps.

4.1 The automaton $\mathcal{A}_p$.

The first step consists in adding in the states of the automaton the information of what was the internal state of the root of the $\mathcal{M}$-configuration that was reduced to this state. Informally, we replace every transition $p(L) \to q$ with $p(L) \to (q,p)$, so given an $\mathcal{M}$-configuration t such that if $t \xrightarrow[\Delta]{*} q$, we have $t \xrightarrow[\Delta_p]{*} (q, S(t))$. This will be useful in the actual construction of $\mathcal{A}'$, as to inversely apply $\mathcal{M}$-rules, we will need to check if the constraint of the rule is satisfied, which will be given by this information (using the stability property, as we remember the final state of the root of each son). More formally, we will also need to adapt the constraint L and to add states to the inner alternating transition system.

We define $\mathcal{A}_p = (\mathcal{Q} \times \mathrm{P}, \mathcal{F} \times \mathrm{P}, \Delta_p, \mathcal{N}_p)$, where

$$\Delta_p = \left\{ p(L_\mathrm{P}) \to (q,p) \;\middle|\; \begin{array}{c} p(L) \to q \in \Delta, \\ L_\mathrm{P} = \{(q_1,p_1)\cdots(q_m,p_m) \mid q_1 \cdots q_m \in L, p_1, \cdots, p_m \in \mathrm{P}\} \end{array} \right\}$$

and $\mathcal{N}_p = (\mathbb{S}_p, \mathbb{F}_p, \eta_p)$, with

- $\mathbb{S}_p = \mathbb{S} \backslash \{s_r \mid r \in \Delta\} \cup \{s_r \mid r \in \Delta_p\}$,
- $\mathbb{F}_p = \mathbb{F} \cap \mathbb{S}_p \cup \{s_r \mid r = p(L_\mathrm{P}) \to (q,p), s_{r'} \in \mathbb{F}, r' = p(L) \to q\}$,
- $\eta_p = \eta \cup \{s_r \xrightarrow{\theta} S \mid s_{r'} \xrightarrow{\theta} S \in \eta, r = p(L_\mathrm{P}) \to (q,p), r' = p(L) \to q\}$.

Lemma 1. $\mathcal{L}(\mathcal{A}_p) = \mathcal{L}(\mathcal{A})$.

Proof. We only have to observe that for every t, $t \xrightarrow[\Delta_p]{*} (q, S(t))$ if and only if $t \xrightarrow[\Delta]{*} q$, and that (q,p) is final if and only if q is final. □

4.2 From constraints over P to constraints over $\mathcal{Q}_p$.

In order to faithfully compute the automaton $\mathcal{A}'$, we need to be able to transfer the constraint of $\mathcal{M}$ to the states of $\mathcal{A}'$. Indeed, we need to recognise only valid predecessors of the configurations recognised by $\mathcal{A}$, i.e. those who satisfy the constraints ϕ. Given a regular language $\phi \subseteq \mathrm{P}^*$, we thus define $\langle\phi\rangle = \{(q_1,p_1)\cdots(q_m,p_m) \mid p_1 \cdots p_m \in \phi, q_1, \cdots, q_m \in \mathcal{Q}\}$. It is straightforward to see that this language is also regular.

4.3 Closed set of constraints

During the construction of $\mathcal{A}'$, we add new transitions of the form $p(L) \to (q', p')$, where the constraints L are constructed from those already appearing in $\mathcal{A}_p$ and the constraints ϕ used in $\mathcal{M}$, using intersection and right-quotient operations. Intersection of the form $L \cap \langle\phi\rangle$ allows us to check that the guarding constraint of an $\mathcal{M}$-rule is satisfied at the considered position in the configuration. The right-quotient

$$L(q,p)^{-1} = \{(q_1,p_1)\cdots(q_m,p_m) \mid (q_1,p_1)\cdots(q_m,p_m)(q,p) \in L\}$$

allows us to get immediate predecessors by an operation of the form C2. We define Λ to be the smallest family of languages over $\mathcal{Q}_p$ such that:

- If $r = p(L) \to (q,p) \in \Delta_p$, $L \in \Lambda$,
- If $L \in \Lambda$ and $\tau = \phi : p \xrightarrow{\theta} p_a \in \delta$, or $\tau = \phi : p \to p_a \rhd p_b \in \delta$, then $L \cap \langle\phi\rangle \in \Lambda$,
- If $L \in \Lambda$ and $(q,p) \in \mathcal{Q}_p$, then $L(q,p)^{-1}$.

The following lemma was shown by Bouajjani *et al.* [2].

Lemma 2. *[2, Lemma 3]* Λ *is finite.*

To prove this lemma, we first remark that as all the L and ϕ are regular, there is an automaton recognising it. Moreover there is a finite number of such constraint. Thus, we can define the product automaton of all these automata to get a finite automaton, and associate each constraint with a set of final states. Thus, the constraints produced by the construction of Λ are obtained by modifying the set of final states (as taking the right-product is equivalent to moving backward by one transition, and as we already have a product automaton, we don't have to introduce new states for the intersection). Thus, only a finite number of automata can be generated this way.

4.4 Constructing $\mathcal{A}'$.

We now actually describe our saturation algorithm constructing $\mathcal{A}'$. To do so we start from $\mathcal{A}_p$ and only add new transitions: we will never add new states, so this process terminates. The main idea is, for every $\mathcal{M}$-rule $r = \phi : p \xrightarrow{\theta} p_a$ and every transition starting with p_a, to add a transition starting with p, ending in the same automaton state. Moreover, we as well ensure the sons of the node we apply the rule to satisfy ϕ by using the constraint $L \cap \langle\phi\rangle$. We also ensure that the elements recognised by the alternating transition systems from the state associated with the new rule are predecessors by θ of those recognised from the one associated with the old rule. For the spawning rule $\phi : p \to p_a \rhd p_b$, we moreover ensure that there is exactly one son less and the label was also accepted by the last son. We need that the label was also accepted by the last son since the spawn operation creates a copy of the parent process's label. Hence, the label of the parent must also be the label of the last son.

We construct $\mathcal{A}' = (\mathcal{Q} \times \mathrm{P}, \mathcal{F} \times \mathrm{P}, \Delta', \mathcal{N}')$, with $\mathcal{N}' = (\mathbb{S}_p, \mathbb{F}_p, \eta')$. We give the formal definition of the construction first, and then informally explain the two rules C1 and C2.

We define Δ' and η' inductively as the fixed point of the following sequence. We begin with $\Delta'_0 = \Delta_p$ and $\eta'_0 = \eta_p$. Now, suppose Δ'_{i-1} and η'_{i-1} are defined. We construct Δ'_i and η'_i to be at least Δ'_{i-1} and η'_{i-1} plus transitions added with one of the following rules:

R1. if we have:

- $\tau = \phi : p \xrightarrow{\theta} p_a \in \delta$,
- $r = p_a(L) \to (q', p') \in \Delta'_{i-1}$,

we add

- $r' = p(L \cap \langle\phi\rangle) \to (q', p')$ to Δ'_i,
- $\nu' = s_{r'} \xrightarrow{\theta} \{s_r\}$ to η'_i.

R2. if we have:

- $\tau = \phi : p \to p_a \rhd p_b \in \delta$,
- $r_1 = p_a(L_1) \to (q', p') \in \Delta'_{i-1}$,
- $r_2 = p_b(L_2) \to (q'', p'') \in \Delta'_{i-1}$, with $\varepsilon \in L_2$,

we add

- $r' = p(L_1(q'', p'')^{-1} \cap \langle\phi\rangle) \to (q', p')$ to Δ'_i,
- $\nu' = s_{r'} \xrightarrow{\mathrm{Id}} \{s_{r_1}, s_{r_2}\}$ to η'_i.

This process terminates when $\Delta'_{i-1} = \Delta'_i$ and $\eta'_{i-1} = \eta'_i$. As the set of states is fixed, there is a finite number of possible rules, thus this process terminates and $\mathcal{A}'$ exists.

Intuitively, C1 works as follows. We want to extend the automaton to recognise the result of a reverse application of $\phi : p \xrightarrow{\theta} p_a$. That is, whenever a configuration t' with the root node having internal state p_a is accepted, we should now accept a configuration t with root internal state p. Hence, we look for a transition (r) that will read and accept the root node of t' and introduce a new transition (r') that will read and accept the root of t. In addition, we need to take care of the children of the root. In particular, to be able to apply τ the children must satisfy ϕ. This is why we intersect with $\langle\phi\rangle$. Furthermore, to simulate the (reverse) update to γ, we add the transition $s_{r'} \xrightarrow{\theta} \{s_r\}$ to assert that the label accepted by r' would be accepted by r after an application of θ.

The rule C2 works similarly to C1, except we need to deal with the addition of a new child in the transition from t to t'. This is a removal when applied in reverse, hence the introduced transition performs a right-quotient on the language of children. In addition, we have to ensure that the spawned child has the same label as the parent. To do this, we look at the transition r_2 used to accept the final child. Note, the right quotient removes the target (q'', p'') of this transition. When applying this transition is reverse, the label γ of the root of t must be the same as the label of the root of t' and its final child. This explains the transition $s_{r'} \xrightarrow{\mathrm{Id}} \{s_{r_1}, s_{r_2}\}$ which ensures γ is accepted at both the root and its final child.

5 Correctness

This section is devoted to showing that $\mathcal{A}'$ accepts $\mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$. It is sufficient to prove the following property:

Proposition 1. *Given a state $(q, p) \in \mathcal{Q}_p$, we have $\mathcal{L}_{q,p}(\mathcal{A}') = \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q,p}(\mathcal{A}_p))$.*

The next two subsection are devoted to prove each direction of this property.

5.1 Soundness

Proposition 2. *Given a state $(q,p) \in \mathcal{Q}_p$, we have $\mathcal{L}_{q,p}(\mathcal{A}') \subseteq \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q,p}(\mathcal{A}_p))$.*

Proof. To prove this proposition, we associate to each state of an automaton (and the inner alternating transition system as well) a *meaning* that is intimately connected to the backwards reachability set we want to construct. We consider a transition $r = p(L) \to (q,p')$ to be sound under the following condition: if we take elements satisfying the meaning of each state appearing at the left of the transition, then the configuration including all these elements satisfies the meaning of the right state of the transition. Intuitively this says that, assuming all actions taken by other transitions in the automaton are correct, the current transition does nothing wrong. We inductively show that every transition appearing in $\mathcal{A}'$ is sound. Finally, we show that if an automaton is sound and contains $\mathcal{A}_p$, it satisfies the proposition, showing that it is the case for $\mathcal{A}'$.

Definition 5 (Meaning of a state)**.** *Given a state $(q,p) \in \mathcal{Q}_p$ and a $\mathcal{M}$-configuration t, we define $t \models (q,p) \Leftrightarrow t \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q,p}(\mathcal{A}_p))$.*

Given a state $s \in \mathbb{S}_p$ which is not a s_r, and $\gamma \in \Gamma$, we define $\gamma \models s \Leftrightarrow \gamma \vdash s$.

Given a state $s_r \in \mathbb{S}_p$, with $r = p(L) \to (q,p')$, and $\gamma \in \Gamma$, we define $\gamma \models s_r$ iff for all $t_1 \models (q_1,p_1), \cdots, t_m \models (q_m,p_m)$, with $(q_1,p_1)\cdots(q_m,p_m) \in L$, we have $(p,\gamma)(t_1,\cdots,t_m) \models (q,p')$.

Given a set of states $S \subseteq \mathbb{S}_p$, and $\gamma \in \Gamma$, we define $\gamma \models S \Leftrightarrow \forall s \in S, \gamma \models s$.

Once we have defined the meaning of the states of the automaton, we can define soundness of a transition.

Definition 6 (Soundness of a transition)**.** *A transition $r = p(L) \to (q,p')$ is sound if for every $t_1 \models (q_1,p_1), \cdots, t_m \models (q_m,p_m)$ with $(q_1,p_1)\cdots(q_m,p_m) \in L$, and $\gamma \models s_r$, we have $(p,\gamma)(t_1,\cdots,t_m) \models (q,p')$.*

A transition $\nu = s \xrightarrow{\theta} S$ is sound if for every $\gamma \in \mathbb{S}$ such that $\theta(\gamma) \models S$, we have $\gamma \models s$.

Intuitively an automaton is sound if all of its transitions are sound. We will later show that a sound automaton does not accept configurations that are not in the backwards reachability set.

Definition 7 (Soundness of an automaton)**.** *An automaton $\mathcal{A}$ is sound if:*

- *$\mathcal{A}$ is constructed from $\mathcal{A}_p$ by adding rules of the form defined earlier.*
- *Every transition of Δ and η is sound.*

We show $\mathcal{A}'$ is sound inductively, hence we must start with the base case and show that $\mathcal{A}_p$ is sound.

Lemma 3. *$\mathcal{A}_p$ is sound.*

Proof. $\mathcal{A}_p$ is constructed from itself by adding nothing.

Given $r = p(L) \to (q,p)$, we consider $(q_1,p_1)\cdots(q_m,p_m) \in L$, $t_1,\cdots,t_m$, with for every i, $t_i \models (q_i,p_i)$, and $\gamma \models s_r$. By definition of $\gamma \models s_r$, we have $(p,\gamma)(t_1,\cdots,t_m) \models (q,p)$. Thus, r is sound.

Given $\nu = s \xrightarrow{\theta} S$ with s not a s_r, we consider γ such that $\theta(\gamma) \models S$. Thus, for every $s' \in S, \theta(\gamma) \vdash s'$, and thus, by definition, we have $\gamma \vdash s$. This is the definition of $\gamma \models s$, and thus ν is sound.

Given $\nu = s_r \xrightarrow{\theta} S$ with $r = p(L) \to (q,p)$, we consider γ such that $\theta(\gamma) \models S$. As we suppose that no state $s_{r'}$ has an incoming transition in η_p, we get as previously $\gamma \vdash s$. We consider

$(q_1,p_1)\cdots(q_m,p_m) \in L$, $t_1 \models (q_1,p_1),\cdots,t_m \models (q_m,p_m)$. By definition, for every i, we have $t_i \in \mathrm{pre}^*_{\mathcal{M}}(t'_i)$, such that $t'_i \xrightarrow[\Delta_p]{*} (q_i,p'_i)$. Thus, the following run is valid:

$$(p,\gamma)(t'_1,\cdots,t'_m) \xrightarrow[\Delta_p]{*} (p,\gamma)((q_1,p_1),\cdots,(q_m,p_m)) \xrightarrow{r} (q,p).$$

Thus, $(p,\gamma)(t'_1,\cdots,t'_m) \in \mathcal{L}_{q,p}(\mathcal{A}_p)$, and thus $(p,\gamma)(t_1,\cdots,t_m) \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q,p}(\mathcal{A}_p))$. By definition, we have $(p,\gamma)(t_1,\cdots,t_m) \models (q,p)$, and we can deduce that $\gamma \models s_r$, which concludes the proof that ν is sound. □

We can now show the inductive case, to show that $\mathcal{A}'$ is sound.

Lemma 4. *$\mathcal{A}'$ is sound.*

Proof. We prove this property by induction on $\mathcal{A}'_i$. $\mathcal{A}'_0 = \mathcal{A}_p$ is sound by the previous lemma. Suppose $\mathcal{A}_{i-1}$ is sound. We distinguished the possible cases for adding new rules.

- For rules added with R1., first, we show r' is sound. Consider $t_1 \models (q_1,p_1),\cdots,t_m \models (q_m,p_m)$, with $(q_1,p_1)\cdots(q_m,p_m) \in L \cap \langle\phi\rangle$, and $\gamma \models s_{r'}$. By definition of $\gamma \models s_{r'}$, we get that $(p,\gamma)(t_1,\cdots,t_m) \models (q',p')$, which allows to conclude that r' is sound.

  Now, let us show that ν' is sound. Consider γ such that $\theta(\gamma) \models s_\nu$. We want to show that $\gamma \models s_{\nu'}$. Consider $t_1 \models (q_1,s_1),\cdots,t_m \models (q_m,s_m)$. Thus, for every i, we have $t'_i \in \mathcal{L}_{q_i,p_i}(\mathcal{A}_p)$ such that $t_i \in \mathrm{pre}^*_{\mathcal{M}}(t'_i)$. As $\mathcal{A}_{i-1}$ is sound, r is sound. Thus we have $(p_a,\theta(\gamma))(t'_1,\cdots,t'_m) \models (q',p')$, which means $(p_a,\theta(\gamma))(t'_1,\cdots,t'_m) \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q',p'}(\mathcal{A}_p))$. For all i, $S(t'_i) = p_i$, as $t'_i \in \mathcal{L}_{q_i,p_i}(\mathcal{A}_p)$. Thus, we can apply τ to $(p,\gamma)(t'_1,\cdots,t'_m)$ and get $(p_a,\theta(\gamma))(t'_1,\cdots,t'_m)$. Thus, $(p,\gamma)(t'_1,\cdots,t'_m) \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q',p'}(\mathcal{A}_p))$, and finally, $(p,\gamma)(t_1,\cdots,t_m) \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q',p'}(\mathcal{A}_p))$. Thus, by definition, $(p,\gamma)(t_1,\cdots,t_m) \models (q',p')$, which proves that $\gamma \models s_{r'}$, and thus ν' is sound.

- For rules added with R2., first, we show r' is sound. Consider $t_1 \models (q_1,p_1),\cdots,t_m \models (q_m,p_m)$, with $(q_1,p_1)\cdots(q_m,p_m) \in L(q'',p'')^{-1} \cap \langle\phi\rangle$, and $\gamma \models s_{r'}$. By definition of $\gamma \models s_{r'}$, we get that $(p,\gamma)(t_1,\cdots,t_m) \models (q',p')$, which allows to conclude that r' is sound.

  Now, let us show that ν' is sound. Consider γ such that $\gamma \models s_{r_1}$ and $\gamma \models s_{r_2}$. We want to show $\gamma \models s_{r'}$. Consider $t_1 \models (q_1,p_1),\cdots,t_m \models (q_m,p_m)$. Thus, for every i, we have $t'_i \in \mathcal{L}_{q_i,p_i}(\mathcal{A}_p)$ such that $t_i \in \mathrm{pre}^*_{\mathcal{M}}(t_i)$. As $\mathcal{A}'_{i-1}$ is sound, r_1 and r_2 are sound. As $\varepsilon \in L_2$, we have $(p_b,\gamma)(\emptyset)) \models (q'',p'')$. Thus, $(p_a,\gamma)(t'_1,\cdots,t'_m,(p_b,\gamma)(\emptyset)) \models (q',p')$, which means $(p_a,\gamma)(t'_1,\cdots,t'_m,(p_b,\gamma)(\emptyset)) \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q',p'}(\mathcal{A}_p))$. For all i, $S(t'_i) = p_i$, as $t'_i \in \mathcal{L}_{q_i,p_i}(\mathcal{A}_p)$. Thus, we can apply τ to $(p,\gamma)(t'_1,\cdots,t'_m)$ and get $(p_a,\gamma)(t'_1,\cdots,t'_m,(p_b,\gamma)(\emptyset))$. Thus, $(p,\gamma)(t'_1,\cdots,t'_m) \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q',p'}(\mathcal{A}_p))$, and finally, $(p,\gamma)(t_1,\cdots,t_m) \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q',p'}(\mathcal{A}_p))$. Thus, by definition, $(p,\gamma)(t_1,\cdots,t_m) \models (q',p')$, which proves that $\gamma \models s_{r'}$, and thus ν' is sound.

Thus, $\mathcal{A}'_i$ is sound, which concludes the proof. □

We are now ready to complete the proof by showing that a sound automaton only accepts the correct configurations.

Lemma 5. *If $\mathcal{A}'$ is sound, for every state (q_0,p_0), $\mathcal{L}_{q_0,p_0}(\mathcal{A}') \subseteq \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q_0,p_0}(\mathcal{A}_p))$.*

Proof. To show this lemma, it is sufficient to show that if $t \in \mathcal{L}_{q_0,p_0}(\mathcal{A}')$, then $t \models (q_0, p_0)$.

First, we show that for every element γ, and every $s \in \mathbb{S}'$, $\gamma \vdash s \Rightarrow \gamma \models s$. We show it by induction on the size of the run. First, suppose s is final, thus, by definition of the soundness we have that $\gamma \models s$ for every γ. Suppose now we have $\gamma \vdash s$, such that the run accepting γ from s starts with a transition $\nu = s \xrightarrow{\theta} S$. By definition, we have for every $s' \in S$, $\theta(\gamma) \vdash s'$. By hypothesis of induction, we have that $\theta(\gamma) \models s'$, and then $\theta(\gamma) \models S$. Thus as ν is sound, we obtain that $\gamma \models s$.

We now prove that if $t \in \mathcal{L}_{q_0,p_0}(\mathcal{A}')$, then $t \models (q_0, p_0)$. We show it by induction on the size of the run. First suppose $t = (p, \gamma)(\emptyset)$. We thus have $t \xrightarrow{r} (q_0, p_0)$, with $r = p(L) \to (q_0, p_0)$, $\varepsilon \in L$ and $\gamma \vdash s_r$. We thus have $\gamma \models s_r$, from what precedes. Thus, as the transition r is sound by hypothesis, we have $t \models (q_0, p_0)$.

Suppose now $t = (p, \gamma)(t_1, \cdots, t_m)$, and $t \xrightarrow[\Delta]{*} (p, \gamma)((q_1, p_1), \cdots, (q_m, p_m)) \xrightarrow{r} (q_0, p_0)$, with $r = p(L) \to (q_0, p_0)$, $(q_1, p_1) \cdots (q_m, p_m) \in L$ and $\gamma \vdash s_r$. We thus have $\gamma \models s_r$, from what precedes. By hypothesis of induction, we have for every i, $t_i \models (q_i, p_i)$. Thus, as the transition r is sound by hypothesis, we have $t \models (q_0, p_0)$. □

Thus, for every state (q, p), we have that $\mathcal{L}_{q,p}(\mathcal{A}') \subseteq \mathcal{L}_{q,p}(\mathrm{pre}^*_{\mathcal{M}}(\mathcal{A}_p))$. □

5.2 Completeness

The proof of completeness of $\mathcal{A}'$ is conceptually simpler than the soundness proof. It proceeds by a straightforward induction over the length of the run showing a configuration is in the backwards reachability set. In the base case we have the configuration is accepted by $\mathcal{A}_p$ and the proof is immediate. In the inductive case, we have t reaches t' by a single transition, and an accepting run of $\mathcal{A}'$ over t. We then inspect the transition from t to t' and show that our construction of $\mathcal{A}'$ ensures that we can modify the accepting run of t' to obtain an accepting run of t.

Proposition 3. *Given a state $(q_0, p_0) \in \mathcal{Q}_p$, we have* $\mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q_0,p_0}(\mathcal{A}_p)) \subseteq \mathcal{L}_{q_0,p_0}(\mathcal{A}')$.

Proof. The proof of this proposition is made inductively: first we recall that $\mathcal{L}_{q_0,p_0}(\mathcal{A}_p)$ is included in $\mathcal{L}_{q_0,p_0}(\mathcal{A}')$. Thus, we show that the predecessor by one $\mathcal{M}$-rule of a configuration t accepted by $\mathcal{A}'$ is accepted by $\mathcal{A}'$. In this last proof, the stability condition plays a crucial role, to show that the transition introduced by the automaton for this rule is actually applicable to accept the predecessor t.

We choose t such that $t \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{(q_0,p_0)}(\mathcal{A}_p))$. We show by induction on the size of the run that $t \in \mathcal{L}_{(q_0,p_0)}(\mathcal{A}')$.

If $t \in \mathcal{L}_{(q_0,p_0)}(\mathcal{A}_p)$, then as $\eta_p \subseteq \eta'$ and $\Delta_p \subseteq \Delta'$, we have the result.

We choose t', such that $t \in \mathrm{pre}_{\mathcal{M}}(t')$. By induction hypothesis, $t' \in \mathcal{L}_{(q_0,p_0)}(\mathcal{A}')$. We have two cases to consider.

- When the transition uses a rule of of the form C1 we have $t = C[(p, \gamma)(t_1, \cdots, t_m)]$, and $t' = C[(p_a, \theta(\gamma))(t_1, \cdots, t_m)]$. Let us decompose the run of $\mathcal{A}'$ accepting t':

$$t' \xrightarrow[\Delta']{*} C[(p_a, \theta(\gamma))((q_1, p_1), \cdots, (q_m, p_m))] \xrightarrow{r} C[q', p'] \xrightarrow[\Delta']{*} (q_0, p_0)$$

  where $r = p_a(L) \to (q', p')$, and $\theta(\gamma) \vdash s_r$.

  Let us show that $(q_1, p_1) \cdots (q_m, p_m) \in \langle \phi \rangle$. For every i, we have $t_i \xrightarrow[\Delta']{*} (q_i, p_i)$. From the proof of the soundness of $\mathcal{A}'$, we get that $t_i \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q_i,p_i}(\mathcal{A}_p))$. Thus, there is

$t'_i \in \mathcal{L}_{q_i,p_i}(\mathcal{A}_p)$, such that $S(t'_i) = p_i$, and $t_i \in \mathrm{pre}^*_{\mathcal{M}}(t'_i)$. Thus, we have $\rho^*_{\Delta'}(S(t_i), S(t'_i))$. As, by hypothesis, we can apply r to t, we have $S(t_1)\cdots S(t_m) \in \phi$. As ϕ is $\rho_{\Delta'}$-stable, we deduce that $p_1, \cdots p_m \in \phi$, and thus $(q_1, p_1)\cdots(q_m, p_m) \in \langle\phi\rangle$.

By construction of $\mathcal{A}'$, we have that $r' = p(L \cap \langle\phi\rangle) \to (q', p')$ in Δ' and $\nu' = s_{r'} \xrightarrow{\theta} s_r$ in η'. As $\theta(\gamma) \vdash s_r$, we get that $\gamma \vdash s_{r'}$. As we have moreover $(q_1, p_1)\cdots(q_m, p_m) \in L \cap \langle\phi\rangle$ the following run is valid:

$$t \xrightarrow[\Delta']{*} C[(q_1, p_1), \cdots, (q_m, p_m)] \xrightarrow{r'} C[q', p'] \xrightarrow[\Delta']{*} (q_0, p_0)$$

Thus $t \in \mathcal{L}_{q_0,p_0}(\mathcal{A}')$.

- When the transition uses a rule of the form C2 we have $t = C[(p, \gamma)(t_1, \cdots, t_m)]$, and $t' = C[(p_a, \gamma)(t_1, \cdots, t_m, (p_b, \gamma)(\emptyset))]$. Let us decompose the run of $\mathcal{A}'$ accepting t':

$$\begin{aligned} t' \quad & \xrightarrow[\Delta']{*} C[(p_a, \theta(\gamma))((q_1, p_1), \cdots, (q_m, p_m), (p_b, \gamma)(\emptyset))] \\ & \xrightarrow{r_2} C[(p_a, \theta(\gamma))((q_1, p_1), \cdots, (q_m, p_m), (q'', p''))] \\ & \xrightarrow{r_1} C[q', p'] \\ & \xrightarrow[\Delta']{*} (q_0, p_0) \end{aligned}$$

where $r_1 = p_a(L_1) \to (q', p')$, $r_2 = p_b(L_2) \to (q'', p'')$ with $\varepsilon \in L_2$, $\gamma \vdash s_{r_1}$, and $\gamma \vdash s_{r_2}$.

Let us show that $(q_1, p_1)\cdots(q_m, p_m) \in \langle\phi\rangle$. For every i, we have $t_i \xrightarrow[\Delta']{*} (q_i, p_i)$. From the proof of the soundness of $\mathcal{A}'$, we get that $t_i \in \mathrm{pre}^*_{\mathcal{M}}(\mathcal{L}_{q_i,p_i}(\mathcal{A}_p))$. Thus, there is $t'_i \in \mathcal{L}_{q_i,p_i}(\mathcal{A}_p)$, such that $S(t'_i) = p_i$, and $t_i \in \mathrm{pre}^*_{\mathcal{M}}(t'_i)$. Thus, we have $\rho^*_{\Delta'}(S(t_i), S(t'_i))$. As, by hypothesis, we can apply r to t, we have $S(t_1)\cdots S(t_m) \in \phi$. As ϕ is $\rho_{\Delta'}$-stable, we deduce that $p_1, \cdots p_m \in \phi$, and thus $(q_1, p_1)\cdots(q_m, p_m) \in \langle\phi\rangle$.

By construction of $\mathcal{A}'$, we have that $r' = p(L_1(q'', p'')^{-1} \cap \langle\phi\rangle) \to (q', p')$ in Δ' and $\nu' = s_{r'} \xrightarrow{\theta} \{s_{r_1}, s_{r_2}\}$ in η'. As $\gamma \vdash s_{r_1}$ and $\gamma \vdash s_{r_2}$, we get that $\gamma \vdash s_{r'}$. As we have moreover $(q_1, p_1)\cdots(q_m, p_m) \in L_1(q'', p'')^{-1} \cap \langle\phi\rangle$ the following run is valid:

$$t \xrightarrow[\Delta']{*} C[(q_1, p_1), \cdots, (q_m, p_m)] \xrightarrow{r'} C[q', p'] \xrightarrow[\Delta']{*} (q_0, p_0)$$

Thus $t \in \mathcal{L}_{q_0,p_0}(\mathcal{A}')$.

Hence the automaton is complete. □

6 Conclusion

We have shown that the saturation algorithm for constrained dynamic pushdown networks introduced by Bouajjani *et al.* [2] can be generalised to not only pushdown networks, but networks of any system for which the alternating reachability problem is decidable. In particular, this includes collapsible pushdown systems, or higher-order recursion schemes, which thus allows the analysis of a kind of concurrent higher-order programs.

We showed that, given a regular target set of configurations, the backwards reachability set computable and is also regular. We make some remarks on our notion of regularity. In order to accept a configuration, an automaton must perform several alternating reachability checks.

This is not regular in the conventional sense. However, for alternating pushdown systems, and indeed alternating collapsible pushdown systems, the backwards reachability set of a regular set of stacks is known to have a regular representation [1, 4]. Thus, we can replace the alternating reachability tests with regular automata which run over the stack contents labelling each node. Thus we obtain a truly regular representation of the backwards reachability sets of CDTNs over these systems.

The natural avenue of future work is to attempt to generalise our model further, to permit more intricate communication between processes. One option is to allow the child nodes to inspect the internal state of their parent processes. In general this leads to an undecidable model. It is an open problem to discover a form of interesting upwards communication that is decidable. Similarly, we may seek to relax the stability constraint. One such option is to use the stability constraint defined by Touili and Atig [37] where internal states are grouped into mutually reachable equivalence classes. Thus, any run moves through a bounded number of equivalence classes. We can then insist that constraints are over the equivalence classes rather than individual states. This is reminiscent of *context-bounded* analysis [32]. We can adapt our construction to allow downwards and upwards communication of this form, but it is not clear whether Λ remains finite.

References

[1] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.

[2] A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, pages 473–487, 2005. URL: `http://dx.doi.org/10.1007/11539452_36`, `doi:10.1007/11539452_36`.

[3] W. S. Brainerd. Tree generating regular systems. *Information and Control*, 14(2):217–231, 1969.

[4] C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 165–176, 2012. URL: `http://dx.doi.org/10.1007/978-3-642-31585-5_18`, `doi:10.1007/978-3-642-31585-5_18`.

[5] C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-shore: a collapsible approach to higher-order verification. In *ICFP*, pages 13–24, 2013.

[6] C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, pages 129–148, 2013. URL: `http://dx.doi.org/10.4230/LIPIcs.CSL.2013.129`, `doi:10.4230/LIPIcs.CSL.2013.129`.

[7] R. Chadha and M. Viswanathan. Decidability results for well-structured transition systems with auxiliary storage. In *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, pages 136–150, 2007. URL: `http://dx.doi.org/10.1007/978-3-540-74407-8_10`, `doi:10.1007/978-3-540-74407-8_10`.

[8] L. Clemente, P. Parys, S. Salvati, and I. Walukiewicz. The diagonal problem for higher-order recursive schemes is decidable. To appear in LICS 2016.

[9] A. Cyriac, P. Gastin, and K. N. Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR*, pages 547–561, 2012.

[10] T. M. Gawlitza, P. Lammich, M. Müller-Olm, H. Seidl, and A. Wenner. Join-lock-sensitive forward reachability analysis for concurrent programs with dynamic process creation. In *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX,*

USA, January 23-25, 2011. Proceedings, pages 199–213, 2011. URL: http://dx.doi.org/10.1007/978-3-642-18275-4_15, doi:10.1007/978-3-642-18275-4_15.

[11] M. Hague. Saturation of concurrent collapsible pushdown systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, pages 313–325, 2013. URL: http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2013.313, doi:10.4230/LIPIcs.FSTTCS.2013.313.

[12] M. Hague, J. Kochems, and C.-H. L. Ong. Unboundedness and downward closures of higher-order pushdown automata. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163, 2016. URL: http://doi.acm.org/10.1145/2837614.2837627, doi:10.1145/2837614.2837627.

[13] M. Hague, A. S. Murawski, C.-H. Luke Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461, 2008.

[14] M. Hague and C.-H. L. Ong. Winning regions of pushdown parity games: A saturation method. In *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, pages 384–398, 2009. URL: http://dx.doi.org/10.1007/978-3-642-04081-8_26, doi:10.1007/978-3-642-04081-8_26.

[15] N. Kobayashi. Model-checking higher-order functions. In *PPDP*, pages 25–36, 2009.

[16] N. Kobayashi. Higher-order model checking: From theory to practice. In *LICS*, pages 219–224, 2011.

[17] N. Kobayashi. A practical l inear time algorithm for trivial automata model checking of higher-order recursion schemes. In *FOSSACS*, pages 260–274, 2011.

[18] N. Kobayashi and A. Igarashi. Model-checking higher-order programs with recursive types. In *ESOP*, pages 431–450, 2013.

[19] Naoki Kobayashi. GTRecS2: A model checker for recursion schemes based on games and types. A tool available at http://www-kb.is.s.u-tokyo.ac.jp/~koba/gtrecs2/, 2012.

[20] S. La Torre, A. Muscholl, and I. Walukiewicz. Safety of parametrized asynchronous shared-memory systems is almost always decidable. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, pages 72–84, 2015. URL: http://dx.doi.org/10.4230/LIPIcs.CONCUR.2015.72, doi:10.4230/LIPIcs.CONCUR.2015.72.

[21] P. Lammich, M. Müller-Olm, and A. Wenner. Predecessor sets of dynamic pushdown networks with tree-regular constraints. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, pages 525–539, 2009. URL: http://dx.doi.org/10.1007/978-3-642-02658-4_39, doi:10.1007/978-3-642-02658-4_39.

[22] Peter Lammich, Markus Müller-Olm, Helmut Seidl, and Alexander Wenner. Contextual locking for dynamic pushdown networks. In *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*, pages 477–498, 2013. URL: http://dx.doi.org/10.1007/978-3-642-38856-9_25, doi:10.1007/978-3-642-38856-9_25.

[23] C. Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, 2003.

[24] D. Lugiez. Forward analysis of dynamic network of pushdown systems is easier without order. *Int. J. Found. Comput. Sci.*, 22(4):843–862, 2011. URL: http://dx.doi.org/10.1142/S0129054111008453, doi:10.1142/S0129054111008453.

[25] D. Lugiez and P. Schnoebelen. The regular viewpoint on pa-processes. In *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, pages 50–66, 1998. URL: http://dx.doi.org/10.1007/BFb0055615, doi:10.1007/BFb0055615.

[26] P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL*, pages 283–294, 2011.

[27] R. P. Neatherway, S. J. Ramsay, and C.-H. L. Ong. A traversal-based algorithm for higher-order model checking. In *ICFP*, pages 353–364, 2012.

[28] Benedikt Nordhoff, Markus Müller-Olm, and Peter Lammich. Iterable forward reachability analysis of monitor-dpns. In *Semantics, Abstract Interpretation, and Reasoning about Programs: Essays Dedicated to David A. Schmidt on the Occasion of his Sixtieth Birthday, Manhattan, Kansas, USA, 19-20th September 2013.*, pages 384–403, 2013. URL: http://dx.doi.org/10.4204/EPTCS.129.24, doi:10.4204/EPTCS.129.24.

[29] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.

[30] V. Penelle. Rewriting higher-order stack trees, 2015. arXiv:1311.4915 [cs.FL], to appear in CSR 15.

[31] S. J. Ramsay, R. P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72, 2014. URL: http://doi.acm.org/10.1145/2535838.2535873, doi:10.1145/2535838.2535873.

[32] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, pages 93–107, 2005.

[33] S. Schwoon. *Model-checking Pushdown Systems*. PhD thesis, Technical University of Munich, 2002.

[34] A. Seth. Games on higher order multi-stack pushdown systems. In *RP*, pages 203–216, 2009.

[35] F. Song and T. Touili. Model checking dynamic pushdown networks. *Formal Asp. Comput.*, 27(2):397–421, 2015. URL: http://dx.doi.org/10.1007/s00165-014-0330-y, doi:10.1007/s00165-014-0330-y.

[36] D. Suwimonteerabuth, F. Berger, S. Schwoon, and J. Esparza. jmoped: A test environment for java programs. In *CAV*, pages 164–167, 2007. doi:10.1007/978-3-540-73368-3_19.

[37] T. Touili and M. Faouzi Atig. Verifying parallel programs with dynamic communication structures. *Theor. Comput. Sci.*, 411(38-39):3460–3468, 2010. URL: http://dx.doi.org/10.1016/j.tcs.2010.05.028, doi:10.1016/j.tcs.2010.05.028.

[38] K. Yasukata, N. Kobayashi, and K. Matsuda. Pairwise reachability analysis for higher order concurrent programs by higher-order model checking. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, pages 312–326, 2014. URL: http://dx.doi.org/10.1007/978-3-662-44584-6_22, doi:10.1007/978-3-662-44584-6_22.

[39] Georg Zetzsche. An approach to computing downward closures. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 440–451, 2015. URL: http://dx.doi.org/10.1007/978-3-662-47666-6_35, doi:10.1007/978-3-662-47666-6_35.