# A Branching Time Variant of CaRet

Jens Oliver Gutsfeld, Markus Müller-Olm, and Benedikt Nordhoff

Institut für Informatik, Westfälische Wilhelms-Universität Münster, Germany

**Abstract.** A shortcoming of traditional logics like LTL and CTL on Pushdown Systems is their inability to express specifications about the call-/return-behavior or the stack content. A natural approach to this problem is the logic CaRet. CaRet adds modalities to LTL that allow specifications to navigate over calls and returns of procedures. In this paper, BranchCaRet, a natural CTL-like variant of CaRet is defined that provides existentially and universally quantified CaRet modalities. We prove that BranchCaRet model checking is decidable and EXPTIME-complete by extending a known CTL model checking algorithm for Pushdown Systems based on Alternating Büchi Pushdown Systems.

## 1 Introduction

In recent years, model checking has been ported from finite state systems to a plethora of other models. One of these model classes are Pushdown Systems (PDSs) which offer a natural abstraction of recursive programs [19]. Unlike finite Kripke structures, they represent a possibly infinite state space and allow us to track procedure calls using a call stack. In the last two decades, the well-known logics LTL and CTL as well as the full modal $\mu$-calculus have been considered for PDSs [20,6,19,22,8]. However, these logics lack the ability to specify properties about the call-/return-behaviour or the stack content. An example property one would like to express is: "Every call has a matching return." The logic CaRet (**Ca**ll and **Ret**urn) [4] provides a solution to this problem. It extends the logic LTL by two types of modalities: abstract modalities and caller modalities. Intuitively, abstract modalities inspect the local behaviour of procedures, while caller modalities inspect the call chain. However, just like LTL, CaRet only defines properties for single paths and cannot specify CTL-like requirements such as "There is a path arising from this configuration satisfying $\phi$" or "For all paths arising from this configuration, $\phi$ holds".

In this paper, we propose the novel logic BranchCaRet, a CTL-style variant of CaRet, that combines the ability of CaRet to specify call/return-related properties with the ability of CTL to specify branching time properties. We show that the BranchCaRet model checking problem can be solved by reduction to the emptiness problem of Alternating Büchi Pushdown Systems and prove that it is EXPTIME-complete, like CTL model checking on PDSs [6].

This paper is organised as follows: In Section 2, we introduce several well-known classes of automata. In particular, we define Pushdown Systems as our

model of systems, Alternating Büchi Pushdown Systems as our main tool of analysis as well as Büchi Pushdown Systems and Multiautomata.

Then, in Section 3, we formally introduce the logics BranchCaRet$^*$, CaRet, and BranchCaRet and explain their potential usefulness as well as their relation to each other. In Section 4, we show how the BranchCaRet model checking problem can be decided using Alternating Büchi Pushdown Systems (ABPDSs) by extending the model checking algorithm of Song and Touili for CTL [20] to BranchCaRet. Finally, in Section 5, we summarise this paper and offer suggestions for future work. Due to lack of space, we only prove some of the theorems in this paper and provide the proofs for the others in the Appendix.

*Related Work.* The logic CaRet was introduced by Alur et al. [4] for Recursive State Machines and ported to PDSs by Nguyen and Touili [17]. Several extensions of CaRet have later been proposed in the literature. These include past-time operators [18,7], a "Within"-modality [1] and a variant for multithreaded programs [14,15]. The practical usefulness of CaRet model checking for malware detection has been demonstrated by Nguyen and Touili [15,16]. A generalisation of the modal $\mu$-calculus called *visibly pushdown $\mu$-calculus* that can express all CaRet properties is given by Alur et al. [3]. We should mention that the translation of CaRet is not direct but works via an encoding of a non-deterministic Büchi Visibly Pushdown Automaton computed from a given CaRet formula. ABPDSs were introduced by Song and Touili [20] and our approach for the construction of the ABPDS from a BranchCaRet formula is based on their approach for CTL, as mentioned already. A general overview of model checking techniques for PDSs, including algorithms for model checking LTL, CTL and CTL$^*$, can be found in Schwoon's Phd thesis [19].

## 2 Preliminaries

In this section, we introduce Pushdown Systems as our system model class and several language acceptor models. Let $AP$ denote a finite, non-empty set of atomic propositions. A *Pushdown System* [20,5] is a tuple $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$ consisting of a finite, non-empty set of control locations $P$, a finite, non-empty stack alphabet $\Gamma$, a transition relation $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$, a labelling function $\lambda : P \times \Gamma \to 2^{AP}$, and a non-empty set of initial configurations $I \subseteq P \times \Gamma$. We write $(p, \gamma) \to (p', \omega)$ to denote $((p, \gamma), (p', \omega)) \in \Delta$. By slight abuse of notation, we also write $(p, \gamma) \to (p', \omega) \in \Delta$. For simplicity, we assume that $\Delta$ is total. We say that a rule $(p, \gamma) \to (p', \omega) \in \Delta$ *pushes* symbols onto the stack if $|\omega| \geq 2$ and *pops* a symbol from the stack if $\omega = \varepsilon$. We also presume the existence of a bottom of stack symbol $\# \in \Gamma$. This symbol may never be pushed or popped and only self-loops (rules of the form $(p, \#) \to (p, \#)$) are allowed from it. We also require the existence of these self-loops since $\Delta$ is total. In the following, we use variants of $\gamma, \beta$ and $\tau$ to denote symbols of $\Gamma$ and variants of $\omega$ to denote elements of $\Gamma^*$. Following a widely used convention, we allow only the following types of rules in $\Delta$: $(p, \gamma) \to (p', \gamma')$ (Internal actions), $(p, \gamma) \to (p', \beta\tau)$ (Calls) and $(p, \gamma) \to (p', \varepsilon)$ (Returns).

A configuration of $\mathcal{P}$ is a tuple $(p, \omega) \in P \times \Gamma^*$ and consists of a control location $p$ and a stack $\omega$. A configuration can be thought of as a snapshot of an execution of $\mathcal{P}$. We say $(p, \gamma)$ is the *configuration head* of a configuration $(p, \omega)$ if $\omega = \gamma\omega'$. We use the same terminology for our other model classes.

The *reachability relation* $\Rightarrow_{\mathcal{P}} \subseteq (P \times \Gamma^+) \times \Delta^* \times (P \times \Gamma^*)$ for a PDS $\mathcal{P}$ is defined as follows: $(p, \omega) \overset{\varepsilon}{\Rightarrow}_{\mathcal{P}} (p, \omega)$, $(p, \gamma\omega) \overset{r_1 w}{\Longrightarrow}_{\mathcal{P}} (p', \omega')$ if $r_1 = (p, \gamma) \to (p'', \omega'') \in \Delta$ and $(p'', \omega''\omega) \overset{w}{\Rightarrow}_{\mathcal{P}} (p', \omega')$. We sometimes omit the rules and write $(p, \omega) \Rightarrow_{\mathcal{P}} (p', \omega')$ if there is a sequence of rules $w$ such that $(p, \omega) \overset{w}{\Rightarrow}_{\mathcal{P}} (p', \omega')$ for $w \in \Delta^*$. Furthermore, we write $(p, \omega) \overset{+}{\Rightarrow}_{\mathcal{P}} (p', \omega')$ for $(p, \omega) \overset{w}{\Rightarrow}_{\mathcal{P}} (p', \omega')$ with $w \in \Delta^+$. From the reachability relation, we naturally obtain a function $Pre^* : 2^{P \times \Gamma^*} \to 2^{P \times \Gamma^*}$ with $Pre^*(C) = \{(p, \omega) \mid \exists (p', \omega') \in C : (p, \omega) \Rightarrow_{\mathcal{P}} (p', \omega')\}$. The function $Pre^*$ returns the set of configurations from which a given set of configurations is reachable.

We model the executions of Pushdown Systems by Kripke structures. The Kripke structure $\mathcal{K}_P = (Q, Q_0, \kappa, \delta)$ for a PDS $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$ consists of the set of states $Q = P \times \Gamma^+$, the set of initial states $Q_0 = \{(p, \gamma\#) \mid (p, \gamma) \in I\}$, the labelling function $\kappa : P \times \Gamma^* \to 2^{AP}$ with $\kappa(p, \gamma\omega) = \lambda(p, \gamma)$, and the transition relation $\delta = \{(q, q') \in Q \times Q \mid \exists r \in \Delta : q \overset{r}{\Rightarrow}_{\mathcal{P}} q'\}$. A *path* of a Kripke structure is an infinite sequence $\pi = \pi_0\pi_1 \ldots$ such that $\pi_0 \in Q_0$ and $(\pi_i, \pi_{i+1}) \in \delta$ for all $i$. We write $\pi(i)$ for the configuration $\pi_i$. For a configuration $\pi(i) = (p, \omega)$ of a path $\pi$ of $K_{\mathcal{P}}$, we write $|\pi(i)|$ to denote the size of the stack $|\omega|$. We denote the set of paths of $\mathcal{K}_{\mathcal{P}}$ by $\Pi_{\mathcal{P}}$. Furthermore, we denote by $Ext(\pi, i)$ the set $\{\pi' \in \Pi_{\mathcal{P}} \mid \forall j \leq i : \pi'(j) = \pi(j)\}$ of *possible extensions* of $\pi$ at index $i$.

In order to analyse executions of PDSs, we need three types of successor functions: global successors, abstract successors and callers. The function $succ_g : \Pi_{\mathcal{P}} \times \mathbb{N} \to \mathbb{N}$ defined by $succ_g(\pi, i) = i + 1$ is the *global successor function*. The global successor is the next index in the execution. $succ_g$ is a total function as all executions of PDSs are infinite and the configuration at the position $i+1$ can thus never be undefined. In order to denote that a function is partial, we use the symbol $\rightsquigarrow$. The partial function $succ_a : \Pi_{\mathcal{P}} \times \mathbb{N} \rightsquigarrow \mathbb{N}$ with

$$succ_a(\pi, i) = \begin{cases} inf\,\{j \mid j > i \wedge |\pi(i)| = |\pi(j)|\}, & \text{if } |\pi(i+1)| \geq |\pi(i)|, \\ undefined, & otherwise \end{cases}$$

is the *abstract successor function*. Unlike the global successor, it behaves differently for different types of configurations. If the configuration at index $i + 1$ is obtained by an internal rule, the abstract successor is equal to the global successor. If the configuration at index $i + 1$ is obtained by a call rule, the abstract successor is the corresponding return (or undefined if that return does not exist). If the global successor of the configuration at index $i$ is a return, the abstract successor is always undefined.

Finally, the partial function $succ_- : \Pi_{\mathcal{P}} \times \mathbb{N} \rightsquigarrow \mathbb{N}$ defined by

$$succ_-(\pi, i) = \begin{cases} sup\,\{j \mid j < i \wedge |\pi(j)| < |\pi(i)|\}, & \text{if } \exists j < i : |\pi(j)| < |\pi(i)| \\ undefined, & otherwise \end{cases}$$

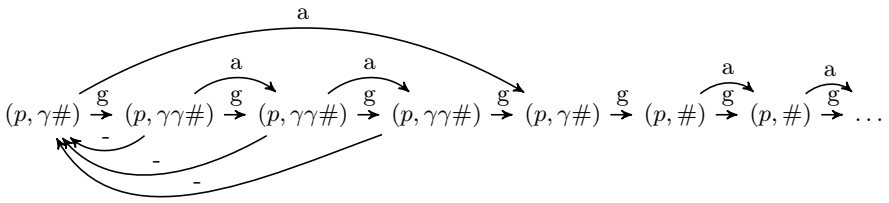


**Fig. 1.** A sample execution of a PDS $\mathcal{P} = (P, \Gamma\Delta, \lambda, I)$ where $P = \{p\}$, $\Gamma = \{\gamma, \#\}$, $\Delta = \{((p,\gamma),(p,\gamma\gamma)), ((p,\gamma),(p,\gamma)), ((p,\gamma),(p,\varepsilon)), ((p,\#),(p,\#))\}$, $\lambda(p,\gamma) = \lambda(p,\#) = \{\}$, and $I = \{(p,\gamma)\}$.

is the *caller function*. This function assigns to an index $i$ the index of the last pending call before it, if there is one. Fig. 1 illustrates the three different kinds of successor functions.

In order to find configurations from which calls without matching returns are possible, we use Büchi Pushdown Systems. A *Büchi Pushdown System* (BPDS) [19] is a tuple $\mathcal{BPS} = (P, \Gamma, \Delta, G)$ consisting of a finite, non-empty set of control locations $P$, a finite, non-empty stack alphabet $\Gamma$, a transition relation $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$, and a set of accepting control locations $G \subseteq P$.

The *reachability relation* and the function $Pre^*$ for BPDSs are defined just as for PDSs, and we write $(p,\omega) \overset{w}{\Rightarrow}_{\mathcal{BPS}} (p',\omega')$ and $(p,\omega) \Rightarrow_{\mathcal{BPS}} (p',\omega')$. To avoid ambiguities, we also differentiate between $Pre^*_{\mathcal{P}}$ and $Pre^*_{\mathcal{BPS}}$.

A *run* $c = (c_0, c_1 \dots)$ of a BPDS $\mathcal{BPS} = (P, \Gamma, \Delta, G)$ is an infinite sequence of configurations with $c_0 = (p, \omega\#)$ and $c_i \overset{r_i}{\Rightarrow}_{\mathcal{BPS}} c_{i+1}$ for transition rules $r_i \in \Delta$. A run is *accepting* if it visits control locations from $G$ infinitely often (Büchi condition). $\mathcal{BPS}$ accepts a configuration $(p, \omega\#)$ if there is an accepting run starting from $(p, \omega\#)$.

For two configurations $c$ and $c'$ of a BPDS $\mathcal{BPS} = (P, \Gamma, \Delta, G)$, we write $c \twoheadrightarrow c'$ to denote $c \Rightarrow_{\mathcal{BPS}} (g,\omega) \overset{+}{\Rightarrow}_{\mathcal{BPS}} c'$ for $g \in G$. Let $(p,\gamma) \in P \times \Gamma$ be a head. We say that $(p,\gamma)$ is a *repeating head* iff $(p,\gamma) \twoheadrightarrow (p,\gamma\omega)$ for some $\omega \in \Gamma^*$. We denote the set of repeating heads by $R$.

Two well-known theorems guide our analysis of BPDSs:

**Theorem 1 ([19]).** *There is an accepting run from $(p,\gamma\omega)$ iff $\omega = \omega''\#$ and $(p,\gamma\omega) \Rightarrow_{\mathcal{BPS}} (p',\gamma'\omega')$ for a repeating head $(p',\gamma')$.*

**Theorem 2 ([11]).** *The repeating heads of a BPDS $\mathcal{BPS} = (P, \Gamma, \Delta, G)$ can be computed in time $\mathcal{O}(|P|^2 \cdot |\Delta|)$ and space $\mathcal{O}(|P| \cdot |\Delta|)$.*

In conjunction with Theorem 5, these theorems allow us to efficiently compute the configurations accepted by a BPDS.

For model checking our logic, we also need a variant of BPDSs with the ability to branch existentially and universally. This variant is given by Alternating Büchi Pushdown Systems. An *Alternating Büchi Pushdown System* (ABPDS) $\mathcal{BP} = (P, \Gamma, \Delta, F)$ consists of a finite, non-empty set of control locations $P$, a finite, non-empty stack alphabet $\Gamma$, a function $\Delta : P \times \Gamma \to \phi_{P \times \Gamma^*}$ that maps elements

of $P \times \Gamma$ to positive boolean formulae over $P \times \Gamma^*$ in disjunctive normal form, and a non-empty set of final states $F \subseteq P$.

For any configuration head $(p, \gamma)$, the associated formula $\phi = \Delta(p, \gamma)$ has the form $\bigvee_{j=1}^{n} \bigwedge_{i=1}^{m_j} (p_i^j, \omega_i^j)$. We can consider each conjunction as a rule $(p, \gamma) \rightarrow \{(p_1^j, \omega_1^j), \ldots (p_{m_j}^j, \omega_{m_j}^j)\}$ and $\phi$ as a whole to be the set consisting of the right hand sides of these rules. This allows us to express logical conditions that we want to be true as transition rules and vice versa. We always implicitly assume that there are two control locations $p, p' \in P$ such that $p \in F$ and $p' \notin F$ and for these states, the only possible transition rules are $(p, \gamma) \rightarrow \{(p, \gamma)\}$ and $(p', \gamma) \rightarrow \{(p', \gamma)\}$ for all $\gamma \in \Gamma$. For some fixed $\gamma \in \Gamma$, we use *true* to denote $(p, \gamma)$ and *false* to denote $(p', \gamma)$. Furthermore, we mostly provide the transition rules for ABPDSs by enumeration. Whenever we do not provide a transition rule for a head $(p, \gamma)$, we implicitly assume $\Delta(p, \gamma) = \mathit{false}$ to ensure $\Delta$ is defined for all heads. A *run* $\rho$ of an ABPDS $\mathcal{BP}$ is an infinite tree of configurations that has a root node $c_0$ and if the configurations $c_{i+1}, \ldots, c_k$ are the children of a node $c_i = (p, \gamma\omega)$, then there is a rule $(p, \gamma) \rightarrow \{(p_{i+1}, \omega_{i+1}), \ldots, (p_k, \omega_k)\} \in \Delta$ such that $c_i = (p_{i+1}, \omega_{i+1}\omega), \ldots, c_k = (p_k, \omega_k\omega)$ holds.

A *path* of a run $\rho$ is a sequence $r_0 r_1 \ldots$ of configurations such that $r_0 = c_0$ and $r_{i+1}$ is a child node of $r_i$. Such a run $\rho$ of $\mathcal{BP}$ is *accepting* if every path of $\rho$ visits control locations in $F$ infinitely often. We say that $\mathcal{BP}$ accepts a configuration $(p, \omega)$ iff there is an accepting run from $(p, \omega)$. By construction, all runs arising from a configuration with configuration head $true$ are always accepting and all runs arising from a configuration with configuration head $false$ are not accepting.

The *reachability relation* for an ABPDS $\mathcal{BP} \Rightarrow_{BP} \subseteq (P \times \Gamma) \times 2^{P \times \Gamma^*}$ is given by the following rules: $(p, \omega) \Rightarrow_{BP} \{(p, \omega)\}$ and $(p, \gamma\omega) \Rightarrow_{BP} \{(p_1, \omega_1\omega), \ldots, (p_n, \omega_n\omega)\}$ if $(p, \gamma) \rightarrow \{(p_1, \omega_1), \ldots, (p_n, \omega_n)\} \in \Delta$, $(p, \gamma\omega) \Rightarrow_{BP} \bigcup C_i$ if $(p, \gamma\omega) \Rightarrow_{BP} \{(p_1, \omega_1), \ldots, (p_n, \omega_n)\}$ and $(p_i, \omega_i) \Rightarrow_{BP} C_i$.

For our model checking algorithm, we need to compute the configurations accepted by an ABPDS. This can be done, for instance, by exploiting corresponding constructions for Parity Pushdown Games (PPG). For a PPG, an Alternating Multiautomaton (AMA) can be computed that recognises whether a configuration belongs to the winning region of a given player in time exponential in the number of control locations and the maximum priority of the PPG [13]. Since an ABPDS corresponds to a PPG with maximum priority two and since there is an algorithm that checks whether an AMA accepts a configuration in time polynomial in the size of the AMA [10], we obtain the following:

**Theorem 3.** *For a configuration $c$ of an ABPDS $\mathcal{BP}$, we can determine whether $\mathcal{BP}$ accepts $c$ in time exponential in $|P|$ and polynomial in all other parameters.*

Finally, we need a class of automata that assists in reachability analysis and the definition of regular valuations. This class is given by Multiautomata. Let $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$ be a PDS. A *Multiautomaton (MA) [20,5] for* $\mathcal{P}$ is a tuple $\mathcal{A} = (Q, \Gamma, \delta, Q_f)$ such that $Q$ is a finite, non-empty set of states with $P \subseteq Q$, $\Gamma$, the input alphabet of $\mathcal{A}$, is the finite, non-empty stack alphabet of

$\mathcal{P}$, $\delta \subseteq Q \times \Gamma \times Q$ is a transition relation and $Q_f \subseteq Q$ is a set of final states. We write $q \xrightarrow{\gamma} q'$ to denote $(q, \gamma, q') \in \delta$.

The *reachability relation* $\rightarrow_\delta \subseteq Q \times \Gamma^* \times Q$ for $\mathcal{A}$ is given by the following rules: $q \xrightarrow{\varepsilon}_\delta q$ and $q \xrightarrow{\gamma\omega}_\delta q'$ if $q \xrightarrow{\gamma}_\delta q''$ and $q'' \xrightarrow{\omega}_\delta q'$ for some $q'' \in Q$. We say that an MA $\mathcal{A}$ accepts $(p, \omega)$ if $p \xrightarrow{\omega} q'$ for some $q' \in Q_f$. We define the language of $\mathcal{A}$ as

$$L(\mathcal{A}) = \{(p, \omega) \in P \times \Gamma^* \mid \mathcal{A} \text{ accepts } (p, \omega)\}.$$

A set of configurations is called *regular* if there is an MA recognising it. It is well-known that multiautomata are closed under union, intersection and complementation [5].

**Theorem 4 ([9]).** *For an MA $\mathcal{A}$ and a configuration $(p, \omega)$, it can be determined in time $\mathcal{O}(|\delta| \cdot |\omega|)$ whether $(p, \omega) \in L(\mathcal{A})$ holds.*

We can now state the following theorem which guides our analysis of possible abstract successors in PDSs:

**Theorem 5 ([19]).** *Let $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$ be a PDS and $\mathcal{A} = (Q, \Gamma, \delta, Q_f)$ be an MA for $\mathcal{P}$. An MA $\mathcal{A}_{Pre^*} = (Q, \Gamma, \delta', Q_f)$ with $L(\mathcal{A}_{Pre^*}) = Pre^*(L(\mathcal{A}))$ can be computed in time $\mathcal{O}(|Q|^2 \cdot |\Delta|)$ and space $\mathcal{O}(|Q| \cdot |\Delta| + |\delta|)$.*

## 3 CaRet and BranchCaRet

In this section, we briefly present the well-known logic CaRet and introduce the new logic BranchCaRet. In order to consider both logics in a common framework, we first introduce the novel logic BranchCaRet$^*$ that includes both logics as subsets in order to allow for a more concise definition. Intuitively, BranchCaRet$^*$ is analogous to CTL$^*$ while CaRet and BranchCaRet are analogous to LTL and CTL respectively.

A *BranchCaRet$^*$ formula* $\phi$ is a formula that can be built from the following context-free grammar:

$$\phi \rightarrow ap \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid E\phi \mid A\phi \mid \bigcirc^f \phi \mid \bigcirc^l_W \phi \mid \phi \,\mathcal{U}^f\, \phi \mid \phi \,\mathcal{U}^a_W\, \phi \mid \phi \,\mathcal{R}^f\, \phi \mid \phi \,\mathcal{R}^a_W\, \phi$$

for $f \in \{g, a, -\}$ , $l \in \{a, -\}$ and $ap \in AP$. We define *true*, *false* and $\Rightarrow$ in the usual way. The semantics of BranchCaRet$^*$ is defined as follows. Let $\mathcal{P}$ be a PDS. $\mathcal{P}$ fulfills a formula $\phi$ (written $P \vDash \phi$) if and only if $\pi \vDash \phi$ for all $\pi \in \Pi_\mathcal{P}$ where $\pi \vDash \phi$ is inductively defined by the following rules:

- $\pi \vDash \phi$ iff $(\pi, 0) \vDash \phi$,
- $(\pi, i) \vDash ap$ iff $ap \in \kappa(\pi(i))$, $ap \in AP$,
- $(\pi, i) \vDash \neg\phi$ iff $(\pi, i) \nvDash \phi$,
- $(\pi, i) \vDash \phi_1 \wedge \phi_2$ iff $(\pi, i) \vDash \phi_1 \wedge (\pi, i) \vDash \phi_2$,
- $(\pi, i) \vDash \phi_1 \vee \phi_2$ iff $(\pi, i) \vDash \phi_1 \vee (\pi, i) \vDash \phi_2$,
- $(\pi, i) \vDash \bigcirc^f \phi_1$ iff $\exists k : succ_f(\pi, i) = k \wedge (\pi, k) \vDash \phi_1, f \in \{g, a, -\}$,

- $(\pi, i) \vDash \bigcirc_W^f \phi_1$ iff $\forall k : succ_f(\pi, i) = k \Rightarrow (\pi, k) \vDash \phi_1, f \in \{a, -\}$,
- $(\pi, i) \vDash \phi_1 \, \mathcal{U}^f \, \phi_2$ iff $\exists k : (\pi, succ_f^k(\pi, i)) \vDash \phi_2 \wedge$
$\forall 0 \leq l < k : (\pi, succ_f^l(\pi, i)) \vDash \phi_1, f \in \{g, a, -\}$,
- $(\pi, i) \vDash \phi_1 \, \mathcal{U}_W^a \, \phi_2$ iff $((\pi, i) \vDash \phi_1 \, \mathcal{U}^a \, \phi_2) \vee$
$(\exists k : succ_a^k(\pi, i)$ is undefined $\wedge$
$\forall 0 \leq l < k : (\pi, succ_a^l(\pi, i)) \vDash \phi_1)$,
- $(\pi, i) \vDash \phi_1 \, \mathcal{R}^f \, \phi_2$ iff $\forall k : (succ_f^k(\pi, i)$ is undefined $\vee$
$(\pi, succ_f^k(\pi, i)) \vDash (\neg\phi_2)) \Rightarrow$
$(\exists n < k : (\pi, succ_f^n(\pi, i)) \vDash \phi_1)$,
- $(\pi, i) \vDash \phi_1 \, \mathcal{R}_W^a \, \phi_2$ iff $((\pi, i) \vDash \phi_1 \, \mathcal{R}^a \, \phi_2) \vee$
$(\exists k : succ_a^k(\pi, i)$ is undefined $\wedge$
$\forall 0 \leq l < k : (\pi, succ_a^l(\pi, i)) \vDash \phi_2)$,
- $(\pi, i) \vDash E\phi_1$ iff $\exists \pi' \in Ext(\pi, i) : (\pi', i) \vDash \phi_1$,
- $(\pi, i) \vDash A\phi_1$ iff $\forall \pi' \in Ext(\pi, i) : (\pi', i) \vDash \phi_1$.

The BranchCaRet$^*$ model checking problem is to check whether $\mathcal{P} \vDash \phi$ for a given PDS $\mathcal{P}$ and a BranchCaRet$^*$ formula $\phi$. We sometimes need to consider the equivalence of formulae and write $\phi \equiv \phi'$ for two BranchCaRet$^*$ formulae iff for any $\mathcal{P}$ and any $\pi \in \Pi_\mathcal{P}$, $\pi \vDash \phi \iff \pi \vDash \phi'$.

The modalities $\bigcirc^g$, $\mathcal{U}^g$ and $\mathcal{R}^g$ are just the usual LTL modalities, i.e. they navigate over sequences of global successors. On the other hand, the modalities $\bigcirc^a$, $\mathcal{U}^a$ and $\mathcal{R}^a$ navigate over abstract successors and the modalities $\bigcirc^-$, $\mathcal{U}^-$ and $\mathcal{R}^-$ navigate over callers.

The modalities $\bigcirc_W^f$, $\mathcal{R}_W^f$ and $\mathcal{U}_W^f$ for $f \in \{g, a, -\}$ are called *weak modalities*. For a formula $\bigcirc_W^f \phi$ with $f \in \{g, a, -\}$ involving the weak successor modality, it is required that either the respective successor fulfill $\phi$ or be undefined. Formulae with a weak Until-Operator, i.e. $\phi_1 \, \mathcal{U}_W^f \, \phi_2$, are fulfilled if either $\phi_2$ holds at some point in the respective sequence or the respective successor is undefined at some point, and in both cases, all configurations before that point in the sequence fulfill $\phi_1$. The weak Release-operator is defined analogously. Notice that the weak modalities can be derived from the normal modalities and we only introduce them for the purpose of constructing formulae in Negation Normal Form (NNF). The term *weak* is used only for modalities which also allow undefined successors in this paper and not for constructs such as the well-known weak Until from the literature that allows $\phi_1$ to hold forever.

For caller modalities, the existential and universal variant have the same semantics because there is only one caller path anyway. We therefore consider only existential caller modalities in the following.

From the basic modalities, we can derive other well-known modalities such as *Future* by $F^f\phi = true \, U^f \, \phi$ and *Generally* by $G^f\phi = \neg(true \, U^f \, \neg\phi)$ for $f \in \{g, a, -\}$. Furthermore, we can define $\phi_1 \, \mathcal{U}_W^- \, \phi_2 = \phi_1 \, \mathcal{U}^- \, (\phi_2 \vee (\phi_1 \wedge \bigcirc_W^- false))$ and $\phi_1 \, \mathcal{R}_W^- \, \phi_2 = (\phi_1 \vee \bigcirc_W^- false) \, \mathcal{R}^- \, \phi_2$. Therefore, we did not include the modalities $\mathcal{U}_W^-$ and $\mathcal{R}_W^-$ in the syntax or semantics of BranchCaRet$^*$.

The logic *CaRet* is the set of BranchCaRet$^*$ formulae which do not contain any quantifiers. For instance, formula $\bigcirc^a(\phi_1\ U^-\ \phi_2)$ is a CaRet formula, while $\bigcirc^a E \bigcirc^g \psi$ is not because CaRet does not allow for quantifiers. CaRet was originally introduced in [4] for the model of Recursive State Machines (RSMs). However, it is well-known that the dynamic behaviour of RSMs can be represented by PDSs [2] and, indeed, CaRet has recently been ported to PDSs explicitly [17]. We therefore restrict our analysis to PDSs.

CaRet can be considered an extension of LTL with modalities for navigating over sequences of abstract successors and callers instead of only global successors (as in LTL).

### 3.1 BranchCaRet

The logic *BranchCaRet* is the set of BranchCaRet$^*$ formulae in which quantifiers are only present in front of modalities and all modalities are quantified. As an example, the formula $E(\phi_1 U^g A \bigcirc^a \phi_2)$ is a BranchCaRet formula, while $\bigcirc^a A \bigcirc^g \phi_1$ is not because the first modality is not quantified. The difference between CaRet and BranchCaRet is analogous to the difference between LTL and CTL: LTL and CaRet only allow modalities without quantifiers, CTL and BranchCaRet only allow quantified modalities. Therefore, CaRet is a logic that talks about linear properties of paths, while BranchCaRet is a branching-time logic.

Naturally, BranchCaRet can express branching time variants of CaRet properties such as the ones presented in [4]. Furthermore, it can express classical CTL properties in a local manner, i.e. on abstract paths and caller paths. It is well-known, for instance, that pushdown systems can be used for interprocedural dataflow analysis e.g. to identify live variables, very busy expressions or available expressions [21,12,19,3]. In BranchCaRet, we can now express specifications for these problems not just for global variables, but also for local variables using abstract modalities. In order to illustrate this, we assume that the variables of a program are divided into global ($G$) and local ($L$) variables with $G \cap L = \emptyset$. We call a variable $x$ *live* at a control location $u$ if there exists a path that, after visiting $u$, encounters a use of $x$ without encountering a definition of $x$ in between. As in CTL, to check whether a global variable $x$ is live at a designated control location labeled $at_u$, we can use the specification $E\mathcal{F}^g(at_u \wedge E((\neg def_x)\ \mathcal{U}^g\ use_x))$, where the atomic proposition $def_x$ is valid just for the control locations where the variable $x$ is defined and $use_x$ for those where it is used. Liveness of a local variable $x$ can now be specified in BranchCaRet very similarly by the formula $E\mathcal{F}^g(at_u \wedge E((\neg def_x)\ \mathcal{U}^a\ use_x))$. Note that the abstract until modality $\mathcal{U}^a$ ensures that the uses and definitions of $x$ are not confused with uses and definitions of other variables with the same name $x$, e.g. in recursively called instances of the same procedure. Similarly, recall that an expression $e$ is *very busy* at a control location $u$ if it is evaluated on all paths emerging from $u$ before any of its free variables are redefined. To check whether an expression $e$ that may contain local as well as global variables is very busy at $u$, we use the formula $A\mathcal{G}^g(at_u \Rightarrow A[(\bigwedge_{x \in FV(e) \cap G} \neg def_x)\ \mathcal{U}^g\ eval_e] \wedge A[(\bigwedge_{x \in FV(e) \cap L} \neg def_x)\ \mathcal{U}^a\ eval_e])$,

where $FV(e)$ denotes the free variables of $e$ and locations evaluating $e$ are labeled with $eval_e$. Furthermore, we can express stack-related liveness properties in BranchCaRet. An example is the property that the stack can always fully be emptied which can be expressed by the following specification: $A\mathcal{G}^g E\mathcal{F}^g E \bigcirc_W^- \mathit{false}$. Another example is the property that for every call, it is possible to return later. Assuming that control locations from which calls are possible are labelled *call* and no other actions are possible from these control locations, we can describe this property by the formula $A\mathcal{G}^g(\mathit{call} \Rightarrow E\bigcirc^a \mathit{true})$. In this setting, the same property can be expressed by $A\mathcal{G}^g\, E\bigcirc_W^- E\bigcirc^a \mathit{true}$ without the assumption that calls are labelled.

For the relation between CaRet and BranchCaRet, we have the following theorem since we can consider both logics on Kripke structures (treated as PDSs without stack action) only and they then reduce to LTL and CTL respectively:

**Theorem 6.** *The expressive power of BranchCaRet and CaRet is incomparable.*

## 4 Model Checking

### 4.1 Computing Abstract Successors and Loops

In addition to the well-known CTL modalities, BranchCaRet allows us to reference callers and abstract successors. In order to cope with the latter in a model checking procedure, two problems need to be considered. First of all, we have to compute all possible abstract successors for a given configuration. Secondly, we need to find the heads from which a call without a matching return is possible. This is necessary for handling abstract modalities correctly.

The first problem is solved by precomputing the heads of possible abstract successors. It is easy to see that the head of a possible abstract successor depends only on the head of the current configuration. Hence we can reduce the analysis of abstract successors of configurations to the analysis of abstract successors of heads. For this purpose, we introduce the function $PSReturns_H : P\times\Gamma \to 2^{P\times\Gamma}$ where $PSReturns_H(p,\gamma) = \{(p',\tau) \mid (p,\gamma) \to (p'',\beta\tau) \wedge (p'',\beta\#) \Rightarrow_{\mathcal{P}} (p',\#)\}$. This function associates every head with heads of possible returns.

**Theorem 7.** *Let $\pi \in \Pi_{\mathcal{P}}$ and $\pi(i) = (p,\gamma\omega)$. There exists $\pi' \in Ext(\pi,i)$ such that $(\pi',i)$ is a call and $succ_a(\pi',i) = k$ and $\pi'(k) = (p',\tau\omega)$ iff $(p',\tau) \in PSReturns_H(p,\gamma)$.*

Theorem 7 implies we solely need $PSReturns_H$ to find the possible heads of returns for a call. From the definition of $PSReturns_H$, we can directly infer a procedure to compute it. This procedure is given by Algorithm 1. Using Theorems 4 and 5 for the complexity of the computation of $Pre^*$ and the membership check in the MA generated by the $Pre^*$ algorithm, we obtain the following complexity:

**Theorem 8.** *Algorithm 1 computes $PSReturns_H$ in time $\mathcal{O}(|\Delta|^2 \cdot |P|)$.*

**Algorithm 1** An algorithm for computing $\text{PSReturns}_H$.

Input: A PDS $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$
Output: The function $PSReturns_H$ as a set
$Calls = \{(p,\gamma) \to (p'', \beta\tau) \in \Delta\}$
**for all** $(p,\gamma) \in P \times \Gamma$ **do**
    $PSReturns_H(p,\gamma) = \emptyset$
**while** $Calls \neq \emptyset$ **do**
    Choose and remove a rule $(p,\gamma) \to (p'', \beta\tau)$ from $Calls$
    **for all** $p' \in P$ **do**
        **if** $(p'', \beta\#) \in Pre^*(\{(p', \#)\})$ **then**
            $PSReturns_H(p,\gamma) = PSReturns_H(p,\gamma) \cup \{(p', \tau)\}$
return $PSReturns_H$

We now turn to the analysis of possibly undefined abstract successors. For a PDS $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$, let $PSLoops_H \subseteq P \times \Gamma$ be the set such that $(p,\gamma) \in PSLoops_H$ iff there is $(p,\gamma) \to (p', \beta\tau) \in \Delta$ and an infinite sequence $c_0 c_1 \ldots$ with $c_0 = (p', \beta\#)$. For each $c_i$ we require a rule $r_i \in \Delta$ such that $c_i \overset{r_i}{\Rightarrow}_{\mathcal{P}} c_{i+1}$ and $|c_i| \geq |c_0|$ must hold for all $i$. The set $PSLoops_H$ thus characterises the heads from which a call with an undefined abstract successor is possible. The name $PSLoops_H$ for configurations with calls without matching returns is chosen in the spirit of infinite loops in computer programs.

**Theorem 9.** *Let $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$ be a PDS, $\pi \in \Pi_{\mathcal{P}}$ and $\pi(i) = (p, \gamma\omega)$ for $i \in \mathbb{N}$. There is $\pi' \in Ext(\pi, i)$ such that $(\pi', i)$ is a call and $succ_a(\pi', i)$ is undefined iff $(p,\gamma) \in PSLoops_H$.*

The *associated BPDS* for a PDS $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$ is $\mathcal{BPS}_{\mathcal{P}} = (P', \Gamma, \Delta', G)$ with control locations $P' = P \uplus \{p_{loop}\}$, accepting control locations $G = P$ and rules $\Delta' = (\Delta \setminus \{(p, \#) \to (p, \#) \mid p \in P\}) \cup \{(p, \#) \to (p_{loop}, \#) \mid p \in P'\}$.

**Theorem 10.** *For any head $(p,\gamma)$, $(p,\gamma) \in PSLoops_H$ iff there is $(p,\gamma) \to (p', \beta\tau) \in \Delta$ and there is an accepting run from $(p', \beta\#)$ in $\mathcal{BPS}_{\mathcal{P}}$.*

Algorithm 2 uses the observation from Theorem 10 that $PSLoops_H$ can be computed by analysing accepting runs from heads in $BPS_{\mathcal{P}}$. Using Theorems 2, 4 and 5, we obtain the following complexity result:

**Theorem 11.** *Algorithm 2 computes the set $PSLoops_H$ in time $\mathcal{O}(|\Delta|^2 \cdot |P|^2)$.*

### 4.2 Configurations with Call Histories

In order to check formulae involving caller modalities, we need to know the last call without a matching return that was made before a configuration. However, there is in general no way to infer this call just from the configuration itself because different calls can lead to the same configuration. In order to solve this problem, we store the caller's head in the stack. For this, we use special stack symbols of the form $(\gamma, (p', \gamma'))$. More concretely, for a call rule $(p,\gamma) \to (p', \beta\tau)$,

**Algorithm 2** An algorithm for computing $PSLoops_H$.

Input: A PDS $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$
Output: The set $PSLoops_H$
$Calls = \{(p,\gamma) \to (p',\beta\tau) \in \Delta\}$
$PSLoops_H = \emptyset$
Construct $\mathcal{BPS}_{\mathcal{P}}$
Compute the repeating heads $R$ of $\mathcal{BPS}_{\mathcal{P}}$
**while** $Calls \neq \emptyset$ **do**
    Remove a rule $(p,\gamma) \to (p',\beta\tau)$ from $Calls$
    **if** $(p',\beta\#) \in Pre^*_{\mathcal{BPS}}(\{(p'',\gamma''\omega) \mid (p'',\gamma'') \in R\})$ **then**
        $PSLoops_H = PSLoops_H \cup \{(p,\gamma)\}$
Return $PSLoops_H$

we switch to $(p', \beta(\tau, (p,\gamma)))$ instead of $(p', \beta\tau)$. For this purpose, we introduce *configurations with call histories*. Let $\Sigma = (P \times \Gamma)$ be the set of possible caller heads and $\Xi = \Gamma \cup (\Gamma \times \Sigma)$ be the set of stack symbols and stack symbols equipped with caller heads. We call the elements of $P \times \Xi^+$ *configurations with call histories*. In the following, we use variants of $\xi$ for elements of $\Xi$, variants of $\sigma$ to denote elements of $\Sigma$ and variants of $\Omega$ to denote elements of $\Xi^*$. In each configuration with call history, the symbol from $\Gamma$ is the usual stack symbol as in a normal configuration and the symbol from $\Xi$ denotes the head of the caller.

In order to employ configurations with call histories for model checking, we need a method to construct a configuration with call history from a configuration in a path. The function $Conf : \Pi_{\mathcal{P}} \times \mathbb{N} \to P \times \Xi^+$ serves this purpose. For $\pi \in \Pi_{\mathcal{P}}$ and $i \in \mathbb{N}$ with $\pi(i) = (p, \gamma_1 \dots \gamma_n\#)$ and $\pi(succ^m_-(\pi, i)) = (p'_m, \gamma'_m\omega_m)$ for $1 \leq m < n$, we define

$$Conf(\pi, i) = \begin{cases} (p, \gamma_1(\gamma_2, (p'_1, \gamma'_1)) \dots (\gamma_n, (p'_{n-1}, \gamma'_{n-1}))\#), & \text{if } n > 1, \\ (p, \gamma_1\#), & \text{if } n = 1, \\ (p, \#), & \text{if } n = 0. \end{cases}$$

Configurations with call histories now enable us to handle caller modalities very easily. Whenever we need to reason about the caller of $(\pi, i)$ and $Conf(\pi, i) = (p, \gamma(\gamma', (p', \gamma''))\Omega)$, we can just pop the symbol $\gamma$ from the stack and afterwards switch to $(p', \gamma''\Omega)$ to restore the caller.

For the analysis of configurations with call histories, we introduce two types of successor relations. These relations lift the global and the abstract successor to configurations with call histories. The relation $\hookrightarrow_g \subseteq (P \times \Gamma) \times (P \times \Xi^*)$ is the smallest relation for which the following conditions hold:

- If $(p,\gamma) \to (p',\gamma') \in \Delta$, then $(p,\gamma) \hookrightarrow_g (p',\gamma')$.
- If $(p,\gamma) \to (p',\varepsilon) \in \Delta$, then $(p,\gamma) \hookrightarrow_g (p',\varepsilon)$.
- If $(p,\gamma) \to (p',\beta\tau) \in \Delta$, then $(p,\gamma) \hookrightarrow_g (p',\beta(\tau,(p,\gamma)))$.

The relation $\hookrightarrow_g$ lifts the global successor to configurations with call histories. For internal moves, the transition is the same as in $\Delta$. For calls, the control

location and top of stack symbol are updated, but additionally, the head of the caller is saved in the second stack symbol. This enables us to reconstruct the caller at any point of time. The relation $\hookrightarrow_a \subseteq (P \times \Gamma) \times (P \times \Xi^*)$ is the smallest relation for which the following conditions hold:

- If $(p, \gamma) \to (p', \gamma') \in \Delta$, then $(p, \gamma) \hookrightarrow_a (p', \gamma')$.
- If $(p', \tau) \in PSReturns_H(p, \gamma)$, then $(p, \gamma) \hookrightarrow_a (p', \tau)$.

The relation $\hookrightarrow_a$ lifts the abstract successor to configurations with call histories. If a possible abstract successor is given by an internal move, it works exactly as $\hookrightarrow_g$. On the other hand, if a call from the current configuration is possible and a matching return exists, $\hookrightarrow_a$ can jump directly to that successor.

We now introduce $ADF$ (**A**lways **D**e**f**ined) sets that indicate whether successors of different types are always defined for configurations.

- $ADF_a = \{(p, \gamma\Omega) \mid (p, \gamma) \to (p', \varepsilon) \notin \Delta \wedge (p, \gamma) \notin PSLoops_H\}$ is the set of configurations with call histories such that $succ_a(\pi, i)$ is always defined iff $Conf(\pi, i) \in ADF_a$.
- $ADF_g = P \times \Xi^+$ is the set of configurations with call histories such that $succ_g(\pi, i)$ is always defined iff $Conf(\pi, i) \in ADF_g$.

Notice that since $succ_g(\pi, i)$ is always defined, we introduce the set $ADF_g$ for mere notational convenience to simplify rules in our ABPDS.

### 4.3 Negation Normal Form

In order to check whether a formula holds in a given PDS, we need to transform the formula to a special form for the construction of the ABPDS. A BranchCaRet formula is in *Negation Normal Form* (NNF) iff the operator $\neg$ occurs only in front of atomic propositions.

Since the existential quantifier is dual to the universal quantifier and $\bigcirc_W$, $\mathcal{U}_W$ and $\mathcal{R}_W$ are dual to $\bigcirc$, $\mathcal{R}$ and $\mathcal{U}$ respectively, we can drive the negations inwards to obtain a BranchCaRet formula $\phi'$ in NNF for any BranchCaRet formula $\phi$. Hence, we have:

**Theorem 12.** *For every BranchCaRet formula $\phi$, there is an equivalent BranchCaRet formula $\phi'$ in NNF.*

### 4.4 An ABPDS for BranchCaRet Model Checking

In this section, we define the ABPDS for a PDS and a formula and show how it can be used for model checking. Our construction extends the construction by Song and Touili for CTL [20]. The construction we use for the global modalities is largely the same as theirs, but we also need to take abstract successors and callers into account. We thus have to add appropriate transitions to possible returns and save the call history on the stack to handle caller modalities. Our construction relies on the closure of a formula. This closure will be used to define the states of our ABPDS: For a BranchCaRet formula $\phi$ in NNF, the closure $cl(\phi)$ is the smallest set such that

- $\phi \in cl(\phi)$,
- If $\phi_1 \wedge \phi_2 \in cl(\phi)$ or $\phi_1 \vee \phi_2 \in cl(\phi)$, then $\phi_1 \in cl(\phi)$ and $\phi_2 \in cl(\phi)$,
- If $\phi' \equiv Q\ Op\ \psi \in cl(\phi)$ for $Op \in \{\bigcirc^f, \bigcirc^l_W\}$, then $\psi \in cl(\phi)$,
- If $Q\ \phi_1\ Op\ \phi_2 \in cl(\phi)$ for $Op \in \{\mathcal{U}^f, \mathcal{R}^f, \mathcal{U}^l_W, \mathcal{R}^l_W\}$, then $\phi_1 \in cl(\phi)$ and $\phi_2 \in cl(\phi)$,
- If $\phi' \equiv E(\phi_1\ Op\ \phi_2) \in cl(\phi)$ for $Op \in \{\mathcal{R}^-, \mathcal{U}^-\}$, then $\phi_1 \in cl(\phi)$, $\phi_2 \in cl(\phi)$ and $E\bigcirc^-\ \phi' \in cl(\phi)$,
- If $\phi' \equiv E(\phi_1\ Op\ \phi_2) \in cl(\phi)$ for $Op \in \{\mathcal{R}^-_W, \mathcal{U}^-_W\}$, then $\phi_1 \in cl(\phi)$, $\phi_2 \in cl(\phi)$ and $E\bigcirc^-_W\ \phi' \in cl(\phi)$,

for $f \in \{g, a, -\}$, $l \in \{g, a\}$ and $Q \in \{E, A\}$. For a formula $\phi$ without caller modalities, the closure is simply the set of subformulae of $\phi$. On the other hand, for caller modalities, we also require that the formula itself as well as an appropriate formula involving a caller modality be in the closure. This is necessary because caller modalities are backwards looking. For example, when we encounter a Caller-Until formula $\phi \equiv\ \phi_1\ \mathcal{U}^-\ \phi_2$ and we see that $\phi_2$ does not hold at $(\pi, i)$, we need to check whether $\phi_1$ holds at $(\pi, i)$ and $\phi$ holds at the caller of $(\pi, i)$. We can do this by checking the formula $\phi' \equiv \phi_1 \wedge E\bigcirc^-\ \phi$, but for this purpose, we need $E\bigcirc^-\ \phi$ in the closure of $\phi$.

Let $\mathcal{P} = (P, \Gamma, \Delta, \lambda, I)$ be a PDS and $\phi$ be a BranchCaRet formula in NNF. Let further $P' = (P \cup \{p_c\}) \times cl(\phi)$ for a fresh control location $p_c$. We denote the elements of $P'$ as $[p, \phi]$ and $[p_c, \phi]$ respectively. We define an ABPDS $\mathcal{BP} = (P', \Xi, \Delta', F)$. Let $F \subseteq P'$ be the smallest set such that

- $[p, ap] \in F$, $ap \in AP$,
- $[p, \neg ap] \in F$, $ap \in AP$,
- $[p, E(\phi_1\ \mathcal{R}^f\ \phi_2)] \in F$ , $f \in \{g, a\}$,
- $[p, A(\phi_1\ \mathcal{R}^f\ \phi_2)] \in F$ , $f \in \{g, a\}$,
- $[p, E(\phi_1\ \mathcal{R}^a_W\ \phi_2)] \in F$,
- $[p, A(\phi_1\ \mathcal{R}^a_W\ \phi_2)] \in F$

iff the respective control locations are members of $P'$. In order to define the transition rules of $\Delta$, we often use conjunctions of the form $\bigwedge_{i \in I} \phi_i$ and disjunctions of the form $\bigvee_{i \in I} \phi_i$ for an index set $I$ and certain formulae $\phi_i$. We adopt the convention that if $I$ is empty, $\bigwedge_{i \in I} \phi_i$ evaluates to *true* and $\bigvee_{i \in I} \phi_i$ evaluates to *false*. $\Delta'$ has the following transition rules iff the respective control locations of the form $[p, \phi]$ and $[p_c, \phi]$ are members of $P'$:

1. $([p, \phi], (\gamma, (p, \gamma'))) \rightarrow ([p, \phi], \gamma) \in \Delta'$,
2. $([p, ap], \gamma) \rightarrow ([p, ap], \gamma) \in \Delta', ap \in AP \wedge ap \in \lambda(p, \gamma)$,
3. $([p, \neg ap], \gamma) \rightarrow ([p, \neg ap], \gamma) \in \Delta', ap \in AP \wedge ap \notin \lambda(p, \gamma)$,
4. $([p, \phi_1 \wedge \phi_2], \gamma) \rightarrow ([p, \phi_1], \gamma) \wedge ([p, \phi_2], \gamma) \in \Delta'$,
5. $([p, \phi_1 \vee \phi_2], \gamma) \rightarrow ([p, \phi_1], \gamma) \vee ([p, \phi_2], \gamma) \in \Delta'$,
6. $([p, E\bigcirc^f\ \phi_1], \gamma) \rightarrow \bigvee_{(p,\gamma) \hookrightarrow_f (p', \Omega)} ([p', \phi_1], \Omega) \in \Delta', f \in \{g, a\}$,
7. $([p, A\bigcirc^f\ \phi_1], \gamma) \rightarrow \bigwedge_{(p,\gamma) \hookrightarrow_f (p', \Omega)} ([p', \phi_1], \Omega) \in \Delta'$ , $f \in \{g, a\} \wedge (p, \gamma) \in ADF_f$ ,
8. $([p, E(\phi_1\ \mathcal{U}^f\ \phi_2)], \gamma) \rightarrow ([p, \phi_2], \gamma) \in \Delta'$, $f \in \{g, a\}$,

9. $([p, E(\phi_1\ \mathcal{U}^f\ \phi_2)], \gamma) \to \bigvee_{(p,\gamma)\hookrightarrow_f(p',\Omega)}(([p,\phi_1],\gamma) \wedge ([p', E(\phi_1\ \mathcal{U}^f \phi_2)], \Omega)) \in \Delta'$, $f \in \{g, a\}$,
10. $([p, A(\phi_1\ \mathcal{U}^f\ \phi_2)], \gamma) \to ([p,\phi_2],\gamma) \in \Delta'$, $f \in \{g, a\}$,
11. $([p, A(\phi_1\ \mathcal{U}^f\ \phi_2)], \gamma) \to \bigwedge_{(p,\gamma)\hookrightarrow_f(p',\Omega)}(([p,\phi_1],\gamma) \wedge ([p', A(\phi_1\ \mathcal{U}^f\ \phi_2)], \Omega)) \in \Delta', (p,\gamma) \in ADF_f$,
12. $([p, E(\phi_1\ \mathcal{R}^f\ \phi_2)], \gamma) \to ([p,\phi_2],\gamma) \wedge ([p,\phi_1],\gamma) \in \Delta'$,
13. $([p, E(\phi_1\ \mathcal{R}^f\ \phi_2)], \gamma) \to \bigvee_{(p,\gamma)\hookrightarrow_f(p',\Omega)}(([p,\phi_2],\gamma) \wedge ([p', E(\phi_1\ \mathcal{R}^f\ \phi_2)], \Omega)) \in \Delta', f \in \{g, a\}$,
14. $([p, A(\phi_1\ \mathcal{R}^f\ \phi_2)], \gamma) \to ([p,\phi_2],\gamma) \wedge ([p,\phi_1],\gamma) \in \Delta'$, $f \in \{g, a\}$,
15. $([p, A(\phi_1\ \mathcal{R}^f\ \phi_2)], \gamma) \to \bigwedge_{(p,\gamma)\hookrightarrow_f(p',\Omega)}(([p,\phi_2],\gamma) \wedge ([p', A(\phi_1\ \mathcal{R}^f\ \phi_2)], \Omega)) \in \Delta'$, $f \in \{g, a\} \wedge (p,\gamma) \in ADF_f$,
16. $([p, E\bigcirc^a_W\ \phi_1], \gamma) \to \mathit{true} \in \Delta'$, $(p,\gamma) \notin ADF_a$,
17. $([p, E\bigcirc^a_W\ \phi_1], \gamma) \to \bigvee_{(p,\gamma)\hookrightarrow_a(p',\Omega)}([p',\phi_1],\Omega) \in \Delta'$,
18. $([p, A\bigcirc^a_W\ \phi_1], \gamma) \to \bigwedge_{(p,\gamma)\hookrightarrow_a(p',\Omega)}([p',\phi_1],\Omega) \in \Delta'$,
19. $([p, E(\phi_1\ \mathcal{U}^a_W\ \phi_2)], \gamma) \to ([p,\phi_1],\gamma) \in \Delta'$, $(p,\gamma) \notin ADF_a$,
20. $([p, E(\phi_1\ \mathcal{U}^a_W\ \phi_2)], \gamma) \to ([p,\phi_2],\ \gamma) \vee \bigvee_{(p,\gamma)\hookrightarrow_a(p',\Omega)}\ (([p,\phi_1],\ \gamma) \wedge ([p',\ E(\phi_1\ \mathcal{U}^f_W\ \phi_2)],\ \Omega)) \in \Delta'$,
21. $([p, A(\phi_1\ \mathcal{U}^a_W\ \phi_2)], \gamma) \to ([p,\phi_2],\gamma) \in \Delta'$,
22. $([p, A(\phi_1\ \mathcal{U}^a_W\ \phi_2)], \gamma) \to (([p,\phi_1],\gamma) \wedge \bigwedge_{(p,\gamma)\hookrightarrow_a(p',\Omega)}([p', A(\phi_1\ U^a_W\ \phi_2)], \Omega)) \in \Delta'$,
23. $([p, E(\phi_1\ \mathcal{R}^a_W\ \phi_2)], \gamma) \to ([p,\phi_2],\gamma) \in \Delta'$, $(p,\gamma) \notin ADF_a$,
24. $([p, E(\phi_1\ \mathcal{R}^a_W\ \phi_2)], \gamma) \to (([p,\phi_2],\gamma) \wedge ([p,\phi_1],\gamma)) \vee \bigvee_{(p,\gamma)\hookrightarrow_a(p',\Omega)}(([p,\phi_2],\gamma) \wedge ([p', E(\phi_1\ \mathcal{R}^a_W\ \phi_2)], \Omega)) \in \Delta'$,
25. $([p, A(\phi_1\ \mathcal{R}^a_W\ \phi_2)], \gamma) \to (([p,\phi_2],\gamma) \wedge ([p,\phi_1],\gamma)) \in \Delta'$,
26. $([p, A(\phi_1\ \mathcal{R}^a_W\ \phi_2)], \gamma) \to (([p,\phi_2],\gamma) \wedge \bigwedge_{(p,\gamma)\hookrightarrow_a(p',\Omega)}([p', A(\phi_1\ \mathcal{R}^a_W\ \phi_2)], \Omega)) \in \Delta'$,
27. $([p, EM\phi_1], \gamma) \to ([p_c, EM\phi_1], \varepsilon) \in \Delta', M \in \{\bigcirc^-, \bigcirc^-_W\} \wedge \gamma \neq \#$,
28. $([p', E\bigcirc^-_W\ \phi_1], \#) \to \mathit{true} \in \Delta', p' \in P \cup \{p_c\}$
29. $([p_c, EM\phi_1], (\gamma, (p',\gamma')) \to ([p',\phi_1],\gamma') \in \Delta', M \in \{\bigcirc^-, \bigcirc^-_W\}$,
30. $([p, E(\phi_1\ \mathcal{U}^-\ \phi_2)], \gamma) \to ([p,\phi_2],\gamma) \vee (([p,\phi_1],\ \gamma) \wedge ([p,\ E\bigcirc^-\ E(\phi_1\ \mathcal{U}^-\ \phi_2)], \gamma)) \in \Delta'$,
31. $([p, E(\phi_1\ \mathcal{R}^-\ \phi_2)], \gamma) \to (([p,\phi_2],\gamma) \wedge ([p,\phi_1],\gamma)) \vee (([p,\phi_2],\gamma) \wedge ([p, E\bigcirc^-\ E(\phi_1\ \mathcal{R}^-\ \phi_2)], \gamma)) \in \Delta'$.

Finally, we need a function that associates executions of our PDS and formulae with configurations of our ABPDS: Let $\mathit{Conf}(\pi, i) = (p, \Omega)$ and $\phi$ be a BranchCaRet formula in NNF. Then $\mathit{Assoc} : (\Pi_{\mathcal{P}} \times \mathbb{N}) \times cl(\phi) \to P' \times \Xi^+$, $\mathit{Assoc}(\pi, i, \phi) = ([p,\phi], \Omega)$ is the function that associates every $(\pi, i)$ and BranchCaRet formula $\phi$ with a configuration of $\mathcal{BP}$. After this formal introduction, we now explain the intuition behind the construction of $\mathcal{BP}$. Our objective is to show that $(\pi, i) \vDash \phi$ iff there is an accepting run from $\mathit{Assoc}(\pi, i, \phi)$.

The control locations $P'$ of $\mathcal{BP}$ are divided into two sets: control locations equipped with formulae $[p, \phi]$ and intermediate states $[p_c, \phi]$. Starting from $\mathit{Assoc}(\pi, i, \phi)$, the configurations of $\mathcal{BP}$ reachable with a single transition rule are of the form $\mathit{Assoc}(\pi', k, \phi')$ for $\pi' \in \mathit{Ext}(\pi, i)$, $\phi' \in cl(\phi)$ and some index $k$.

The only exception are configurations with control locations of the form $[p_c, \phi]$ in their head. These control locations solely are used to restore the caller in case we encounter a caller modality. The transition rules in $\Delta'$ are straightforward applications of the semantics of BranchCaRet. We explain these rules and the set $F$ in turn to establish an intuitive basis for the formal correctness proof of our approach. Our rules consist of three general blocks. The first block (rules 1-15) handles (negated) atomic propositions as well as global and abstract modalities. An atomic proposition holds iff the current configuration is labelled with it, so we construct an accepting run via an infinite loop in this case and include no transitions at all otherwise. For successor modalities, we can just check if the formula $\phi_1$ holds in the respective successor, while also requiring that the successor be always defined in the universal case. For Until formulae, we always check whether $\phi_2$ holds directly or $\phi_1$ holds and the Until formula holds again in the respective successor. The Release modality is handled analogously. The only outlier in this block is rule 1. This rule is only applicable after a symbol has been popped from the stack using a rule that does not refer to caller modalities. In this case, when we encounter a symbol that combines a stack symbol and the head of a caller, we can just ignore that caller and place just the stack symbol on the stack. The second block (rules 16-26) handles weak abstract modalities. The rules for these modalities are largely similar to the corresponding rules in the first block, but we also allow the abstract successor to be undefined in the relevant cases. For example, for the existential abstract successor modality, we require that either the abstract successor be undefined (as indicated by the $ADF_a$ set) or fulfill $\phi_2$. In both blocks, the control locations for the Release modality are in $F$ because we also need to construct accepting runs in case these are visited infinitely often. In contrast, all other modalities only lead to finitely many steps before either a Release modality or a (negated) atomic proposition is encountered. Therefore, the control locations for these modalities do not need to be included in the set $F$. The third block (rules 27-31) deals with caller modalities. The caller is handled by popping the current symbol from the stack and switching into the intermediate state $p_c$. If the current configuration with call history has a caller, this intermediate state will contain a stack symbol of the form $(\gamma, (p', \gamma'))$ and $(p, \gamma')$ will be the configuration head of the caller. Thus, we can restore the configuration with call history that was present when the call was made then check whether the formula holds in that configuration. If the stack symbol is not of that form, we know there is no caller and can use this information to construct an accepting run for weak caller formulae. The other caller modalities are handled by reduction to the simple caller modality. For caller modalities, we do not need to include the control locations for Release in $F$ because only finitely many steps are possible anyway. We can now state our main theorem for the connection between BranchCaRet model checking and the ABPDS $\mathcal{BP}$.

**Theorem 13.** *Let $\phi$ be a BranchCaRet formula in NNF and $\pi \in \Pi_{\mathcal{P}}$. Then $(\pi, i) \vDash \phi$ iff there is an accepting run from $Assoc(\pi, i, \phi)$ in $BP$.*

Using algorithms 1 and 2 to compute $PSReturns_H$ and the $ADF$ sets, we can construct $\mathcal{BP}$ in polynomial time. Since there are only at most $|P| \cdot |\Gamma|$ initial configurations and thus only that many variants of $\pi(0)$, we only need to check for $|P| \cdot |\Gamma|$ configurations whether there is an accepting run from the respective configuration to solve the BranchCaRet model checking problem. There are $\mathcal{O}(|P| \cdot |\phi|)$ control locations in $\mathcal{BP}$ and we can therefore use Theorem 3 to obtain:

**Theorem 14.** *The BranchCaRet model checking problem can be solved in time exponential in $|P|$ and $|\phi|$ and polynomial in $|\Gamma|$ and $|\Delta|$.*

In conjunction with the fact that CTL model checking is already EXPTIME-hard [6] (even for a fixed size formula), we obtain the following result:

**Theorem 15.** *BranchCaRet model checking is EXPTIME-complete (even for a fixed size formula).*

This is the same model checking complexity as for CTL. Theorem 15 also shows why it is prudent to consider BranchCaRet separately from the whole BranchCaRet$^*$. For the latter logic, we obtain a lower bound of 2EXPTIME-hardness because this bound already holds for the subset CTL$^*$ [6]. Thus, model checking BranchCaRet is generally more efficient. We note that our approach can easily be extended to accommodate regular stack properties as atomic propositions with similar transition rules as in [20].

## 5 Conclusion

In this paper, we introduced the logic BranchCaRet as a branching time variant of CaRet. We showed how the BranchCaRet model checking problem can be solved via the construction of ABPDSs and checking them for emptiness. We further proved BranchCaRet model checking to be EXPTIME-complete and therefore to have the same asymptotic model checking complexity as CTL.

*Future Work.* We would like to implement our model checking algorithm in a model checker to analyse its feasibility in case studies. Furthermore, a natural question would be whether and how our model checking approach for BranchCaRet can be extended to the full logic BranchCaRet$^*$ which was only considered as a general framework in this paper. As mentioned in Section 1, the visibly pushdown $\mu$-calculus (VP-$\mu$) [3] can express all properties of the logic CaRet. It would be interesting to investigate the relation of VP-$\mu$ to BranchCaRet. Finally, we would like to analyse whether our approach can be extended to a branching time version of the logic NWTL and the "Within"-modality introduced by Alur et al. in [1].

## References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., and Libkin, L. First-order and temporal logics for nested words. *Logical Methods in Computer Science 4*, 4 (2008).

2. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T. W., and Yannakakis, M. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst. 27*, 4 (2005), 786–818.
3. Alur, R., Chaudhuri, S., and Madhusudan, P. A fixpoint calculus for local and global program flows. In *POPL 2006s* (2006), pp. 153–165.
4. Alur, R., Etessami, K., and Madhusudan, P. A temporal logic of nested calls and returns. In *TACAS 2004* (2004), pp. 467–481.
5. Bouajjani, A., Esparza, J., and Maler, O. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR '97* (1997), pp. 135–150.
6. Bozzelli, L. Complexity results on branching-time pushdown model checking. *Theor. Comput. Sci. 379*, 1-2 (2007), 286–297.
7. Bozzelli, L. CaRet with forgettable past. *Electr. Notes Theor. Comput. Sci. 231* (2009), 343–361.
8. Burkart, O., and Steffen, B. Model checking the full modal mu-calculus for infinite sequential processes. *Theor. Comput. Sci. 221*, 1-2 (1999), 251–270.
9. Cachat, T. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP 2002* (2002), pp. 704–715.
10. Cachat, T. *Games on pushdown graphs and extensions.* PhD thesis, RWTH Aachen University, Germany, 2003.
11. Esparza, J., Hansel, D., Rossmanith, P., and Schwoon, S. Efficient algorithms for model checking pushdown systems. In *CAV 2000* (2000), pp. 232–247.
12. Esparza, J., and Knoop, J. An automata-theoretic approach to interprocedural data-flow analysis. In *Foundations of Software Science and Computation Structure, Second International Conference, FoSSaCS'99.* (1999), pp. 14–30.
13. Hague, M., and Ong, C. L. Winning regions of pushdown parity games: A saturation method. In *CONCUR 2009* (2009), pp. 384–398.
14. La Torre, S., and Napoli, M. A temporal logic for multi-threaded programs. In *Theoretical Computer Science - 7th IFIP TC 1* (2012), pp. 225–239.
15. Nguyen, H., and Touili, T. CARET analysis of multithreaded programs. *CoRR abs/1709.09006* (2017).
16. Nguyen, H., and Touili, T. CARET model checking for malware detection. In *SPIN Symposium 2017* (2017), pp. 152–161.
17. Nguyen, H., and Touili, T. CARET model checking for pushdown systems. In *SAC 2017* (2017), pp. 1393–1400.
18. Rosu, G., Chen, F., and Ball, T. Synthesizing monitors for safety properties: This time with calls and returns. In *RV 2008* (2008), pp. 51–68.
19. Schwoon, S. *Model checking pushdown systems.* PhD thesis, Technical University Munich, Germany, 2002.
20. Song, F., and Touili, T. Efficient CTL model-checking for pushdown systems. *Theor. Comput. Sci. 549* (2014), 127–145.
21. Steffen, B. Data flow analysis as model checking. In *TACS '91* (1991), pp. 346–365.
22. Walukiewicz, I. Pushdown processes: Games and model-checking. *Inf. Comput. 164*, 2 (2001), 234–263.

## Appendix

*Proof (of Theorem 7).* We prove the following stronger statement from which our claim trivially follows:

$$\begin{aligned}&\exists \pi' \in Ext(\pi, i) : \pi'(i+1) = (p'', \beta\tau\omega) \wedge \pi'(succ_a(\pi', i)) = (p', \tau\omega)\\ &\iff (p, \gamma) \to (p'', \beta\tau) \in \Delta \wedge (p'', \beta\#) \in Pre^*(\{(p', \#)\})\end{aligned}$$

1. "$\Rightarrow$"
   Let $\pi' \in Ext(\pi, i)$ with $\pi'(i+1) = (p'', \beta\tau\omega)$ and $\pi'(succ_a(\pi', i)) = (p', \tau\omega)$. Naturally, $(p, \gamma) \to (p'', \beta\tau)$ then holds. Let $succ_a(\pi', i) = k$. By definition, $succ_a = inf\{j | j > i \wedge |\pi'(j)| = |\pi'(i)|\}$. This implies $|\pi'(j)| \geq |\pi'(i+1)|$ $\forall (i+1) \leq j \leq (k-1)$ and $|\pi'(k)| = |\pi'(i+1)| - 1$. Let $\pi'(k-1) = (q, \gamma'\tau\omega)$. We therefore obtain $(p'', \beta\#) \Rightarrow_{\mathcal{P}} (q, \gamma'\#)$. Since $\pi'(k) = (p', \tau\omega)$, there must be a transition $(q, \gamma') \to (p', \varepsilon) \in \Delta$ and thus $(q, \gamma'\#) \Rightarrow_{\mathcal{P}} (p', \#)$. In combination, $(p'', \beta\#) \Rightarrow_{\mathcal{P}} (p', \#)$ and therefore $(p'', \beta\#) \in Pre^*(\{p', \#\})$.
2. "$\Leftarrow$"
   Let $(p, \gamma) \to (p'', \beta\tau) \in \Delta$ and $(p'', \beta\#) \in Pre^*(\{(p', \#)\})$. Let $\pi(i) = (p, \gamma\omega)$. By assumption, there is a sequence $c_0, \ldots, c_k$ with $c_0 = (p'', \beta\#)$, $c_k = (p', \#)$ and $c_j \overset{r_j}{\Rightarrow}_{\mathcal{P}} c_{j+1}$ for all $0 \leq j < k$ with $r_j \in \Delta$. Without loss of generality, we can assume $|c_j| > |c_k|$ because there are only self-loop rules for heads with # as the top stack symbol. Let $c_{k-1} = (q, \gamma'\#)$. We obtain an extension $\pi' \in Ext(\pi, i)$ such that $\pi'(i + (k-1)) = (q, \gamma'\tau\omega)$ and $|\pi'(j)| \geq |\pi'(i+1)| \forall (i+1) \leq j < k$. There must be a rule $(q, \gamma') \to (p', \varepsilon) \in \Delta$ and therefore we can construct $\pi'$ with $\pi'(i+k) = (p', \tau\omega)$. By construction, $succ_a(\pi', i) = i + k$ and this concludes the proof. □

*Proof (of Theorem 8).* The initialisation in line 5 is executed $|P| * |\Gamma|$ times. The main loop in line 6 is executed once for every call rule since $Calls$ is initialised to all possible call rules and no rules are ever added to $Calls$ afterwards. A natural upper bound for $|Calls|$ is given by $|\Delta|$ because there cannot be more call rules than total transitions. Therefore, we can assume that the main loop is executed $|\Delta|$ times in the worst case. In every iteration of the main loop, the loop in line 8 is executed $|P|$ times and we can implement the check in line 8 using Theorem 1. Thus, Algorithm 1 always terminates. For correctness, notice that only heads $(p, \gamma)$ for which call rules exist can have any returns. Therefore, if no call rules exist for a head $(p, \gamma)$, the returns for that head are initialised to $\emptyset$ in line 5. They are also never changed afterwards because no entries for the head exist in $Calls$ and the main loop considers only entries from $Calls$. This implies that Algorithm 1 works correctly for heads without associated call rules. For all other heads, there is an entry for every call rule in $Calls$. The loop in line 8 checks for every control location whether it is part of a head of a possible return and the correctness follows from Theorem 7. For the complexity analysis, only the execution time of line 9 remains to be estimated. Since an AMA for $\{(p, \#)\}$ for any $p \in \mathcal{P}$ can be constructed in constant time and with constant size, it takes time $\mathcal{O}(|\Delta|)$ to construct $\mathcal{A}_{Pre^*}$ and $\mathcal{O}(|\Delta|)$ time to decide the membership of

$(p'', \beta\#)$ in $Pre^*(\{p', \#\})$ because the size of heads is constant and the size of $\mathcal{A}_{Pre^*}$ is in $\mathcal{O}(|\Delta|)$. In total, we obtain a bound of $\mathcal{O}(|P| * |\Gamma| + |\Delta|^2 * |P|)$ and this can be simplified to $\mathcal{O}(|\Delta|^2 * |P|)$ since there is at least one transition rule for every head $(p, \gamma)$ and thus $|\Delta| \geq |P| * |\Gamma|$ holds.

*Proof (of Theorem10).*

- "$\Rightarrow$"
  Let $(p, \gamma) \in PSLoops_H$. By assumption, there is a rule $(p, \gamma) \to (p', \beta\tau) \in \Delta$ and an infinite sequence $c_0 c_1 \dots$ such that $c_0 = (p', \beta\#)$, $c_i \overset{r_i}{\Rightarrow}_{\mathcal{P}} c_{i+1}$ for $r_i \in \Delta$ and $|c_i| \geq |c_0| \forall i \geq 1$. The same sequence is a valid run in $\mathcal{BPDS}_{\mathcal{P}}$ because the symbol $\#$ is never reached and this implies all transition rules $r_i$ are also available in $\Delta'$. Also, $P = G$ holds and therefore accepting control locations are visited infinitely often in $c$. This means $c$ is accepting.
- "$\Leftarrow$"
  Let $c = (c_0, c_1, \dots)$ be an accepting run from $(p', \beta\#)$ in $\mathcal{BPDS}_{\mathcal{P}}$. This means no configuration $(p, \#)$ can ever be visited because otherwise, the run would have to visit $(p_{loop}, \#)$, but this configuration is not accepting and only has a self-loop as a possible transition. Therefore, all rules and control locations used to construct $c$ are also available in $\mathcal{P}$. Furthermore, $|c_i| \geq |c_0| \forall i$ holds. The rule $(p, \gamma) \to (p', \beta\tau)$ is available in $\Delta$ by assumption and we obtain $(p, \gamma) \in PSLoops_H$. □

*Proof (of Theorem 11).* The construction in line 5 removes every transition of the form $(p, \#) \to (p, \#) \in \Delta$ and instead adds a transition to $(p, \#) \to (p', \#)$ to $\Delta'$. Since one such transition exists for every $p \in P$, this can be done in $|P|$ steps. The computation of the repeating heads in line 6 can be performed in time $\mathcal{O}(|P|^2 * |\Delta|)$ by Theorem 2. The loop in line 7 cannot be executed more than $|\Delta|$ times in the worst case because we can again use $|\Delta|$ as an upper bound for $|Calls|$. Furthermore, there can be between 0 and $|P \times \Gamma|$ repeating heads and we can build an AMA $\mathcal{A} = (Q, \Gamma, \delta, Q_f)$ representing the set $\{(p'', \gamma''\omega) | (p'', \gamma'') \in R\}$ as follows:

- $Q = P \cup \{q_f\}$,
- $\delta = \{(p, \gamma, q_f) | (p, \gamma) \in R\} \cup \{(q_f, \gamma, q_f) | \gamma \in \Gamma\}$,
- $Q_f = \{q_f\}$.

Obviously, $\mathcal{A}$ can be constructed in time $\mathcal{O}(|P \times \Gamma|)$, $|Q| = |P + 1|$ and $|\delta| = \mathcal{O}(|P \times \Gamma|)$. Accordingly, $\mathcal{A}_{Pre^*}$ for this set can be computed in time $\mathcal{O}(|Q|^2 * |\Delta|) = \mathcal{O}(|P|^2 * |\Delta|)$ by Theorem 5. The membership of $(p', \beta\#)$ in $L(\mathcal{A}_{Pre^*})$ can be checked in time $\mathcal{O}(|\delta| + |P^2| * |\Delta|)$ by Theorem 4. The bounds for the construction of $\mathcal{BPDS}_{\mathcal{P}}$ and the construction of $\mathcal{A}$ are asymptotically irrelevant.

Adding the remaining bounds, we obtain a total bound of

$$\begin{aligned}
&\mathcal{O}(|P|^2 * \Delta|) + \mathcal{O}(|P|^2 * |\Delta|) + \mathcal{O}(|\Delta| * (|\delta| + |P|^2 * |\Delta|)) \\
&= \mathcal{O}(|\Delta| * |\delta| + |\Delta|^2 * |P|^2) \\
&= \mathcal{O}(|\Delta| * |P| * |\Gamma| + |\Delta|^2 * |P|^2) \\
&= \mathcal{O}(|\Delta|^2 * |P|^2 + |\Delta|^2 * |P|^2) && (|\Delta| \geq |P| * |\Gamma|) \\
&= \mathcal{O}(|\Delta|^2 * |P|^2)
\end{aligned}$$

The correctness of Algorithm 2 follows from Theorem 10 and Theorem 1. □

We frequently make use of the following proposition:

**Proposition 1.** *Let $\pi \in \Pi$ and $\pi' \in Ext(\pi, i)$. For any BranchCaRet formula $\phi$, $(\pi, i) \vDash \phi$ iff $(\pi', i) \vDash \phi$.*

*Proof.* The proof is by induction on the complexity of $\phi$.

**Basis:** $\phi = ap$, $ap \in AP$
By definition, $\pi(i) = \pi'(i)$ and therefore $ap \in \kappa(\pi(i)) \iff ap \in \kappa(\pi'(i))$.
**Induction Hypothesis:** For any proper subformula $\phi'$ of $\phi$, $(\pi, i) \vDash \phi'$ iff $(\pi', i) \vDash \phi'$.
**Step:**
- $\phi = \phi_1 \wedge \phi_2$, $\phi = \phi_1 \vee \phi_2$, $\phi = \neg\phi_1$
  The claim follows directly from the induction hypothesis.
- $\phi = E\psi$ or $\phi = A\psi$
  The claim follows from the fact that $Ext(\pi, i) = Ext(\pi', i)$. □

We will also use the following fact:

**Proposition 2.** *Let $(p, \xi) \in P \times \Xi$ and $(p, \xi\Omega) \in P \times \Xi^+$. Then $(p, \xi) \in ADF_a$ iff $(p, \xi\Omega) \in ADF_a$.*

We can now prove our main theorem. For this purpose, we now introduce two new relations. The relation $\rightharpoonup_g \subseteq (P \times \Xi^+) \times (P \times \Xi+)$ is the smallest relation such that

- If $(p, \gamma) \to (p', \gamma') \in \Delta$, then $(p, \gamma\Omega) \rightharpoonup_g (p', \gamma'\Omega)$.
- If $(p, \gamma) \to (p', \gamma'\tau) \in \Delta$, then $(p, \gamma\Omega) \rightharpoonup_g (p', \gamma'(\tau, (p, \gamma))\Omega$.
- If $(p, \gamma) \to (p', \varepsilon) \in \Delta$, then $(p, \gamma(\gamma', (p'', \tau))\Omega) \rightharpoonup_g (p', \gamma'\Omega)$.
- If $(p, \gamma) \to (p', \varepsilon) \in \Delta$, then $(p, \gamma\#) \rightharpoonup_g (p', \#)$.

The relation $\rightharpoonup_a \subseteq (P \times \Xi^+) \times (P \times \Xi+)$ is given by $(p, \gamma\Omega) \rightharpoonup_a (p', \gamma'\Omega)$ if $(p, \gamma) \hookrightarrow_a (p', \gamma')$. These relations lift the relations $\hookrightarrow_g$ and $\hookrightarrow_a$ from heads to configurations with call histories. Both relations behave analogously to their $\hookrightarrow$-variants. The only new rule in $\rightharpoonup_g$ is the penultimate one which combines popping a symbol from the stack with the use of rule 1 in just one step. We use these relations throughout the proof to simplify the description of the steps in our ABPDS and do not explicitly mention the use of rule 1, but it will always be clear from the context how the proof could be rewritten using just the regular relations $\hookrightarrow_g$ and $\hookrightarrow_a$.

*Proof (of Theorem 13).*
"⇒"
We use induction on the complexity of the formula $\phi$. Throughout this proof, we sometimes need other types of inductions. To distinguish between the induction hypotheses for both inductions, we call the induction hypothesis for the induction on the formula complexity the *global induction hypothesis* and the other induction hypothesis the *local induction hypothesis*.

**Basis:**

1. $\phi = ap, ap \in AP$
   Since $(\pi, i) \vDash \phi$, $ap \in \kappa(\pi(i))$ by definition and therefore, the configuration head of $Assoc(\pi, i, ap)$ has the form $([p, ap], \gamma)$ and $ap \in \lambda(p, \gamma)$. Using rule 2, we can loop infinitely in $Assoc(\pi, i, ap)$. Since $[p, ap] \in F$, the run thus obtained is accepting.
2. $\phi = \neg ap, ap \in AP$
   Similar to the previous case.

**Global Induction Hypothesis:**

For any $\pi' \in \Pi_{\mathcal{P}}$, any proper subformula $\phi'$ of $\phi$ and any index $j$, $(\pi', j) \vDash \phi'$ implies that there is an accepting run from $Assoc(\pi', j, \phi')$.

**Step:**

3. $\phi = \phi_1 \wedge \phi_2$
   Since $(\pi, i) \vDash \phi_1$ and $(\pi, i) \vDash \phi_2$, the induction hypothesis implies that there are accepting runs $\rho_1$ and $\rho_2$ from $Assoc(\pi, i, \phi_1)$ and $Assoc(\pi, i, \phi_2)$. Using rule 4, we can build an accepting run $\rho$ with root node $Assoc(\pi, i, \phi)$ and $\rho_1$ and $\rho_2$ as left and right subtrees.
4. $\phi = \phi_1 \vee \phi_2$
   The proof is similar to the previous case, using rule 5 instead of rule 4.
5. $\phi = E \bigcirc^g \phi_1$
   Since $(\pi, i) \vDash \phi$ holds, there must be $\pi' \in Ext(\pi, i)$ such that $(\pi', i+1) \vDash \phi_1$. Naturally, there is $r \in \Delta$ with $\pi'(i) \overset{r}{\Rightarrow}_{\mathcal{P}} \pi'(i+1)$. As a consequence, $Conf(\pi, i) \rightharpoonup_g Conf(\pi', i+1)$. Furthermore, the induction hypothesis yields an accepting run $\rho$ from $Assoc(\pi', i+1, \phi_1)$. Using rule 6, we can construct an accepting run $\rho'$ for $Assoc(\pi, i, \phi)$ by using $Assoc(\pi, i, \phi)$ as the root node and connecting it with the accepting run $\rho$.
6. $\phi = A \bigcirc^g \phi_1$
   Similar to the previous case.
7. $\phi = E\ \phi_1\ \mathcal{U}^g\ \phi_2$
   Since $(\pi, i) \vDash \phi$ holds, there is $\pi' \in Ext(\pi, i)$ and $j \geq 0$ such that $(\pi', i+j) \vDash \phi_2$ and $(\pi', i+k) \vDash \phi_1 \forall 0 \leq k < j$. To prove our claim, we use another induction on $j$.
   **Basis:**
   Let $j = 0$. Then $(\pi', i) \vDash \phi_2$ and the global induction hypothesis yields an accepting run from $Assoc(\pi', i, \phi_2)$. Since $Assoc(\pi', i, \phi_2) = Assoc(\pi, i, \phi_2)$, there is also an accepting run from $Assoc(\pi, i, \phi_2)$. Using rule 8, we directly obtain an accepting run from $Assoc(\pi, i, \phi)$ as well.

**Local Induction Hypothesis:**

For any path $\pi' \in \Pi_{\mathcal{P}}$ and any index $m$, if $(\pi', m+j') \vDash \phi_2$ for $j' < j$ and $(\pi', k) \vDash \phi_1 \forall m \leq k < j'$, then there is an accepting run from $Assoc(\pi', m, \phi)$.

**Step:**

Let $j > 0$. By assumption, $(\pi', i) \vDash \phi_1$ and $(\pi', i+1) \vDash E\ \phi_1\ \mathcal{U}^g\ \phi_2$ holds. In particular, for $(\pi', i+1)$, the first index $j'$ for which $(\pi', i+1+j') \vDash \phi_2$ holds is $j' = j-1$. The local induction hypothesis thus implies that there is an accepting run from $Assoc(\pi', i+1, \phi)$ and the global induction hypothesis implies that there is an accepting run from $Assoc(\pi, i, \phi_1)$. Furthermore, $Conf(\pi, i) \rightharpoonup_g Conf(\pi', i+1)$ holds. Rule 9 then shows $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{Assoc(\pi, i, \phi_1), Assoc(\pi', i+1, \phi)\}$. We can therefore directly use the accepting runs from both these configurations to obtain an accepting run from $Assoc(\pi, i, \phi)$.

8. $\phi = A\ \phi_1\ \mathcal{U}^g\ \phi_2$
   Similar to the previous case.
9. $\phi = E\ \phi_1\ \mathcal{R}^g\ \phi_2$
   Let $\pi' \in Ext(\pi, i)$ such that $(\pi', i) \vDash \phi_1\ \mathcal{R}^g\ \phi_2$. We need to consider two cases. In the first case, there is an index $j$ such that $(\pi', i+j) \vDash \phi_1$ and $(\pi', i+k) \vDash \phi_2\ \forall 0 \leq k \leq j$. In this case, the proof is analogous to the proof of case 7. In the second case, $\phi_1$ is never fulfilled and $(\pi', i+k) \vDash \phi_2\ \forall k$. First of all, notice that $Conf(\pi', m) \rightharpoonup_g Conf(\pi', m+1) \forall m \geq i$. From the global induction hypothesis, we know that there are accepting runs $\rho_m$ from $Assoc(\pi', m, \phi_2)$. Let us construct an accepting run $\rho$' from $Assoc(\pi', i, \phi)$. The tree $\rho'$ can be represented by an infinite linear sequence of nodes $n_l$ with the following properties:
   (a) The root $n_0$ is $Assoc(\pi', i, \phi)$.
   (b) $n_l = Assoc(\pi', i+l, \phi)$ and $n_l$ has $Assoc(\pi', i+l, \phi_2)$ and $n_{l+1}$ as its children.
   (c) $Assoc(\pi', i+l)$ is the root node of an accepting run $\rho_l$ which is integrated into $\rho'$.

   $\rho'$ is a valid run because every run $\rho_l$ is valid by the global induction hypothesis and the transitions from $n_l$ to $n_{l+1}$ and the root of $\rho_l$ are valid due to rule 13. Every path that reaches one of the subtrees $\rho_l$ is accepting since these runs are accepting. The only path that does not reach any of these trees is the path $n_0 n_1 \ldots$. But since every control location $[p, \phi]$ is accepting, this path is also accepting. Thus, all paths of $\rho'$ are accepting and $\rho'$ is an accepting run. The fact that $Assoc(\pi', i, \phi) = Assoc(\pi, i, \phi)$ holds then implies our claim.
10. $\phi = A\ \phi_1\ \mathcal{R}^g\ \phi_2$
    The proof is similar to the previous case.
11. $\phi = E \bigcirc^a \phi_1$
    Let $\pi' \in Ext(\pi, i)$ such that $(\pi', i) \vDash \bigcirc^a \phi_1$. This means $succ_a(\pi', i) = k$ and $(\pi', k) \vDash \phi_1$. Accordingly, $Conf(\pi, i) \rightharpoonup_a Conf(\pi', k)$ and therefore $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{Assoc(\pi', k, \phi_1)\}$ by rule 6. The global induction

hypothesis yields an accepting run from $Assoc(\pi', k, \phi_1)$ and thus also from $Assoc(\pi, i, \phi)$.

12. $\phi = A \bigcirc^a \phi_1$
Let $(\pi, i) \vDash \phi$. Then $succ_a(\pi' i)$ is defined for all possible $\pi' \in Ext(\pi, i)$. This shows $Conf(\pi, i) \in ADF_a$ and thus rule 7 is available. In all other respects, the proof is similar to the previous case.
13. $\phi = E\ \phi_1\ \mathcal{U}^a\ \phi_2$
Let $(\pi, i) \vDash \phi$. Then there is $\pi' \in Ext(\pi, i)$ such that $(\pi', i) \vDash \phi_1\ \mathcal{U}^a\ \phi_2$. This means there is $k \geq 0$ such that $(\pi', succ_a^k(\pi', i)) \vDash \phi_2$ and $(\pi', succ_a^m(\pi', i)) \vDash \phi_1\ \forall 0 \leq m < k$. To show our claim, we use another local induction on $k$.
**Basis:**
Let $k = 0$. We have $succ_a^0(\pi', i) = i$ and therefore $(\pi', i) \vDash \phi_2$. By the global induction hypothesis, there is an accepting run from $Assoc(\pi', i, \phi_2)$. Using rule 8, we see $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{Assoc(\pi, i, \phi_2)\}$ and therefore directly obtain an accepting run from $Assoc(\pi, i, \phi)$ because $Assoc(\pi', i, \phi_2) = Assoc(\pi, i, \phi_2)$ holds.
**Local Induction Hypothesis**
For every $\pi' \in \Pi_{\mathcal{P}}$ and every index $m$, if there is $k' < k$ such that $(\pi', succ_a^{k'}(\pi', m)) \vDash \phi_2$ and $(\pi', succ_a^l(\pi', m)) \vDash \phi_1 \forall 0 \leq l < k'$, then there is an accepting run from $Assoc(\pi', m, \phi)$.
**Step:**
Let $k > 0$. Then, $(\pi', i) \vDash \phi_1$ and $(\pi', succ_a(\pi', i)) \vDash \phi$. This implies there is $\pi'' \in Ext(\pi', succ_a(\pi', i))$ with $(\pi'', succ_a^{k'}(\pi'', succ_a(\pi', i))) \vDash \phi_2$ for $k' = k - 1$ and $(\pi'', succ_a^l(\pi'', i)) \vDash \phi_1 \forall 0 \leq l < k'$. Accordingly, there is an accepting run from $Assoc(\pi', succ_a(\pi', i), \phi)$ by the local induction hypothesis. Furthermore, $Conf(\pi, i) \rightharpoonup_a Conf(\pi', succ_a(\pi', i))$ and there is an accepting run from $Assoc(\pi, i, \phi_1)$ by the global induction hypothesis. Using rule 8, we see that $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{Assoc(\pi, i, \phi_1), Assoc(\pi', succ_a(\pi', i), \phi)\}$ and the accepting runs from these configurations therefore yield an accepting run from $Assoc(\pi, i, \phi)$.
14. $\phi = A\ \phi_1\ \mathcal{U}^a\ \phi_2$
The proof is similar to the previous case.
15. $\phi = E\ \phi_1 \mathcal{R}^a\ \phi_2$
Let $(\pi, i) \vDash \phi$. Then there is $\pi' \in Ext(\pi, i)$ such that $(\pi', i) \vDash \phi_1 \mathcal{R}^a\ \phi_2$. There are two cases to consider. In the first case, there is $k$ such that $succ_a^k(\pi', i) = j$, $(\pi', j) \vDash \phi_1$ and $(\pi', succ^l(\pi', i)) \vDash \phi_2 \forall 0 \leq l \leq k$. This case is similar to case 13. In the second case, $\phi_1$ is never fulfilled and $(\pi', succ_a^l(\pi', i)) \vDash \phi_2\ \forall l \geq 0$. By the induction hypothesis, there are accepting runs $\rho_l$ from $Assoc(\pi', succ_l(\pi', i), \phi_2)$. Furthermore, $Conf(\pi', succ_a^l(\pi', i)) \rightharpoonup_a Conf(\pi', succ_a^{l+1}(\pi', i))$ holds by definition. We now construct an accepting run $\rho'$ from $Assoc(\pi, i, \phi)$. This run can be represented by an infinite linear sequence of nodes $n_l$ with the following properties:
(a) The root $n_0$ is $Assoc(\pi', i, \phi)$.

(b) $n_l = Assoc(\pi', succ_a^l(\pi', i), \phi)$ and $n_l$ has $n_{l+1}$ and $Assoc(\pi', succ_a^l(\pi', i), \phi_2)$ as children.

(c) $Assoc(\pi', succ_a^l(\pi', i), \phi_2)$ is the root node of the accepting run $\rho_l$ which is integrated into $\rho'$.

$\rho'$ is a valid run because all $\rho_l$ are valid runs by the induction hypothesis and the transition from $n_l$ to $n_{l+1}$ and the root node of $\rho_l$ is possible by rule 13. Paths that reach nodes in any $\rho_l$ are accepting by definition. The only other possible path is $n_0, n_1, \ldots$ and control locations in this path are accepting since $[p, \phi] \in F$ for any $p \in P$. Accordingly, $\rho'$ is an accepting run. The fact that $Assoc(\pi', i, \phi) = Assoc(\pi, i, \phi)$ holds implies our claim.

16. $\phi = A\ \phi_1\ \mathcal{R}^a\ \phi_2$
The proof is similar to the previous case, using rule 15 instead of rule 13 in the second case.

17. $\phi = E \bigcirc_W^a \phi_1$
If the abstract successor exists and fulfills $\phi_1$, the proof is similar to case 11. Otherwise, the abstract successor must be undefined in at least one case and this is the case iff $Conf(\pi, i) \notin ADF_a$. Rule 16 then implies $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{true\}$ and there is thus an accepting run from $Assoc(\pi, i, \phi)$.

18. $\phi = A \bigcirc_W^a \phi_1$
Let $(\pi, i) \vDash \phi$. Then for any $\pi' \in Ext(\pi, i)$, $(\pi', i) \vDash \bigcirc_W^a \phi_1$ holds. This means $succ_a(\pi', i)$ is either undefined or $(\pi', succ_a(\pi', i)) \vDash \phi_1$. If $succ_a(\pi', i)$ is undefined on all extensions, rule 18 yields $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{true\}$ and there is an accepting run. Let there be at least one extension on which $succ_a$ is defined and let $\pi_0, \ldots, \pi_k \in Ext(\pi, i)$ such that the $\pi_j$ cover exactly the heads abstract successors of $(\pi, i)$ can have. Then $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{Assoc(\pi_0, succ_a(\pi_0, i), \phi_1), \ldots, Assoc(\pi_k, succ_a(\pi_k, i), \phi_1)\}$ by rule 18. By the global induction hypothesis, there are accepting runs $\rho_j$ from $Assoc(\pi_j, succ_a(\pi_j, i), \phi_1)$ and therefore there is an accepting run from $Assoc(\pi, i, \phi)$.

19. $\phi = E\ \phi_1\ \mathcal{U}_W^a\ \phi_2$
We consider two cases. In the first case, $(\pi, i) \vDash E\phi_1 U^a \phi_2$ holds and the proof is similar to case 13. Otherwise, there is $\pi' \in Ext(\pi, i)$ and an index $j > 0$ such that $succ_a^j(\pi', i)$ is undefined and $succ_a^k(\pi', i) \vDash \phi_1 \forall k < j$. We use a local induction on $j$.

**Basis:**

Let $j = 1$. Then $(\pi', i) \vDash \phi_1$ and $succ_a(\pi', i)$ is undefined. As a consequence, $Conf(\pi', i) \notin ADF_a$ and therefore rule 19 is available. This implies $Assoc(\pi, i, \phi) \rightarrow_{\mathcal{BP}} \{Assoc(\pi, i, \phi_1)\}$. By the global induction hypothesis, there is an accepting run from $Assoc(\pi, i, \phi_1)$ and therefore also from $Assoc(\pi, i, \phi)$.

**Local Induction Hypothesis:**

For any $\pi' \in \Pi_{\mathcal{P}}$ and any index $m$, if $succ_a^{j'}(\pi', m)$ is undefined for $j' < j$ and $(\pi', succ_a^k(\pi', m)) \vDash \phi_1 \forall k < j'$, then there is an accepting run from $Assoc(\pi', m, \phi)$.

**Step:**

Let $j > 1$. By assumption, $(\pi', i) \vDash \phi_1$ and there is $k$ such that $succ_a(\pi', i) = k$ and $(\pi', k) \vDash \phi$. Furthermore, we have $Conf(\pi, i) \rightharpoonup_a Conf(\pi', k)$. Using rule 20, we obtain $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{Assoc(\pi, i, \phi_1), Assoc(\pi', k, \phi)\}$. From the global induction hypothesis, we get an accepting run for $Assoc(\pi, i, \phi_1)$ and from the local induction hypothesis, we get an accepting run for $Assoc(\pi', k, \phi)$. Thus, there is an accepting run for $Assoc(\pi, i, \phi)$.

20. $\phi = A\ \phi_1\ \mathcal{U}^a_W\ \phi_2$
Similar to the previous case.
21. $\phi = E\ \phi_1\ \mathcal{R}^a_W\ \phi_2$
There are two possible cases. In the first case, $(\pi, i) \vDash E\phi_1 R^a \phi_2$ holds and the proof is thus similar to case 15. In the second case, there is $\pi' \in Ext(\pi, i)$ and an index $j$ such that $succ^j_a(\pi', i)$ is undefined and $(\pi', succ^k_a(\pi', i)) \vDash \phi_2 \forall k < j$. But then, the proof is the same as in case 19.
22. $\phi = A\ \phi_1\ \mathcal{R}^a_W\ \phi_2$
Similar to the previous case.
23. $\phi = E \bigcirc^-\ \phi_1$
Let $(\pi, i) \vDash \phi$. Then $(\pi, succ_-(\pi, i)) \vDash \phi_1$. Using rules 27 and 29, we obtain $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{Assoc(\pi, succ_-(\pi, i), \phi_1)\}$. By the global induction hypothesis, there is an accepting run from $Assoc(\pi, succ_-(\pi, i), \phi_1)$. This implies that there is an accepting run from $Assoc(\pi, i, \phi)$ as well.
24. $\phi = E \bigcirc^-_W\ \phi_1$
If $succ_-(\pi, i)$ exists, the proof is similar to the previous case. If it does not exist, $Assoc(\pi, i, \phi)$ has the form $([p, \phi], \gamma\#)$ or $([p, \phi], \#)$. In the first case, Rule 27 can be used to obtain $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} ([p_c, \phi], \#)$ and then $([p_c, \phi], \#) \Rightarrow_{\mathcal{BP}} true$ by rule 28. In the second case, we directly have $([p, \phi], \#) \Rightarrow_{\mathcal{BP}} true$ by rule 28. Both results directly imply an accepting run.
25. $\phi = E\ \phi_1\ \mathcal{U}^-\ \phi_2$
Let $(\pi, i) \vDash \phi$. Then there must be an index $j$ such that $succ^j_-(\pi, i) = k$, $(\pi, k) \vDash \phi_2$ and $(\pi, succ^m_-(\pi, i)) \vDash \phi_1 \forall 0 \leq m < k$. We use a local induction on $j$.

**Basis:**

Let $j = 0$. Then, using rule 30, we obtain $Assoc(\pi, i, \phi) \Rightarrow_{\mathcal{BP}} \{Assoc(\pi, i, \phi_2)\}$ and there is an accepting run from $Assoc(\pi, i, \phi_2)$ due to the global induction hypothesis.

**Local Induction Hypothesis:**

If $(\pi, succ^{j'}_-(\pi, n)) \vDash \phi_2$ for any index $n$, $j' < j$ and $(\pi, succ^m_-(\pi, n)) \vDash \phi_1 \forall m < j'$, then there is an accepting run from $Assoc(\pi, i, n)$.

**Step:**

Let $j \geq 1$. Then $(\pi, i) \vDash \phi_1$ and $(\pi, succ_-(\pi, i)) \vDash \phi$. Using rule 30, we need to show that there are accepting runs from $Assoc(\pi, i, \phi_1)$ and from $Assoc(\pi, i, E \bigcirc^- \phi)$. The first claim follows directly from

the global induction hypothesis. For the second claim, we obtain $Assoc(\pi, i, E \bigcirc^- \phi) \Rightarrow_{\mathcal{BP}} \{Assoc(\pi, succ_-(\pi, i), \phi)\}$ using rules 27 and 29. Naturally, for $j' = j - 1$, $(\pi, succ_-^{j'}(\pi, i)) \vDash \phi_2$ and $(\pi, succ_-^{m}(\pi, i)) \vDash \phi_1 \forall 0 \leq m < j'$. The local induction hypothesis then implies that there is an accepting run from $Assoc(\pi, succ_-(\pi, i), \phi)$ and thus from $Assoc(\pi, i, E \bigcirc^- \phi)$ as desired.

26. $\phi = E\ \phi_1\ \mathcal{R}^-\ \phi_2$
    The proof is similar to the previous case. Notice in particular that we have only finitely many possible iterations until the bottom of the stack is reached. Therefore, unlike in the other release cases, it is not possible that $\phi$ is fulfilled and $\phi_1$ is never reached.

" $\Leftarrow$"
Again, we use induction on the complexity of the formula $\phi$.

**Basis:**

1. $\phi = ap, ap \in AP$
   Let $\rho$ be an accepting run from $Assoc(\pi, i, ap)$. The only rule that can be used to construct such a run is rule 2. This implies $Assoc(\pi, i, ap)$ has the form $([p, ap], \gamma\Omega)$ and $ap \in \lambda(p, \gamma)$. But then, $\pi(i)$ must have the form $(p, \gamma\omega)$ and $\kappa(\pi(i)) = \lambda(p, \gamma)$. Accordingly, $ap \in \kappa(\pi(i))$ and therefore $(\pi, i) \vDash ap$.
2. $\phi = \neg ap, ap \in AP$
   Similar to the previous case.

**Global Induction Hypothesis:**

For any $\pi' \in \Pi_{\mathcal{P}}$, any index $m$ and any proper subformula $\phi'$ of $\phi$, an accepting run from $Assoc(\pi', m, \phi')$ in $\mathcal{BP}$ implies $(\pi', m) \vDash \phi'$.

**Step:**

3. $\phi = \phi_1 \wedge \phi_2$
   Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. The only available rule for the root node of $\rho$ is rule 4 and this means there must be accepting runs from $Assoc(\pi, i, \phi_1)$ and $Assoc(\pi, i, \phi_2)$. The global induction hypothesis implies that $(\pi, i) \vDash \phi_1$ and $(\pi, i) \vDash \phi_2$. But this is equivalent to $(\pi, i) \vDash \phi_1 \wedge \phi_2$.
4. $\phi = \phi_1 \vee \phi_2$
   Similar to the previous case.
5. $\phi = E \bigcirc^g \phi_1$
   For an accepting run $\rho$ from $Assoc(\pi, i, \phi)$, the only possible transition from the root node is based on rule 6. This means there is one direct child node $c = ([p, \phi_1], \Omega)$ and $Conf(\pi, i) \rightharpoonup_g (p, \Omega)$. This implies there is $\pi' \in Ext(\pi, i)$ such that $Conf(\pi', i + 1) = (p, \Omega)$. The accepting run from $c$ implies $(\pi', i + 1) \vDash \phi_1$ by the global induction hypothesis and thus $(\pi, i) \vDash \phi$.
6. $\phi = A \bigcirc^g \phi_1$
   Similar to the previous case.

7. $\phi = E\ \phi_1\ \mathcal{U}^g\ \phi_2$
   Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. There are two possible cases. In the first case, $\rho$ has $Assoc(\pi, i, \phi_2)$ as a direct child node and there is an accepting run from $Assoc(\pi, i, \phi_2)$. The global induction hypothesis then implies $(\pi, i) \vDash \phi_2$ and therefore also $(\pi, i) \vDash \phi$. Otherwise, $\rho$ can be represented by a finite sequence of nodes $n_0 \dots n_m$ for $m \geq 1$ with the following properties:
   (a) The root node $n_0$ is $Assoc(\pi, i, \phi)$.
   (b) For any $l > 0$, there is $\pi_l$ such that $n_l = Assoc(\pi_l, i, \phi)$ and $\pi_l \in Ext(\pi_{l-1}, i + (l-1))$ for $n_{l-1} = Assoc(\pi_{l-1}, i + (l-1), \phi)$.
   (c) For any $n_l$ with $l < m$, the children of $n_l$ are $Assoc(\pi_l.i + l, \phi_1)$ and $n_{l+1}$.
   (d) For $n_m$, the only child node is $Assoc(\pi_m, i + m, \phi_2)$.
   (e) $Assoc(\pi_l, i + l, \phi_1)$ and $Assoc(\pi_m, i + m, \phi_2)$ are the root nodes of accepting runs integrated in $\rho$.

   Property b) follows from the fact that the connections between the nodes must be based on rule 9. There are accepting runs from $Assoc(\pi_l, i + l, \phi_1)$ $\forall l < m$ and from $Assoc(\pi_m,\ i + m, \phi_2)$ and the global induction hypothesis thus implies $(\pi_l, i + l) \vDash \phi_1$ $\forall 0 \leq l < m$ and $(\pi_m, i + m) \vDash \phi_2$. Furthermore, notice that $\pi_m \in Ext(\pi_l, i + l)$ $\forall 0 \leq l < m$ and therefore $(\pi_m, i + l) \vDash \phi_1$ $\forall 0 \leq l < m$ by Proposition 1. This shows $(\pi_m, i) \vDash \phi_1\ \mathcal{U}^g\ \phi_2$ and since $\pi_m \in Ext(\pi, i)$, we obtain $(\pi, i) \vDash \phi$.
8. $\phi = A\ \phi_1\ \mathcal{U}^g\ \phi_2$
   Similar to the previous case.
9. $\phi = E\ \phi_1\ \mathcal{R}^g\ \phi_2$
   Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. There are two cases. In the first case, $\rho$ eventually reaches $([p'', \phi_1 \wedge \phi_2], \Omega'')$ as a child of $Assoc(\pi'', i + k, \phi)$ for some $\pi'' \in Ext(\pi, i)$ and some $k$. Then, the proof is similar to case 7. In the second case, $\rho$ can be represented by an infinite linear sequence of nodes $n_l$ with the following properties:
   (a) The root node $n_0$ is $Assoc(\pi, i, \phi)$.
   (b) For any $l > 0$, there is $\pi_l$ such that $n_l = Assoc(\pi_l, i, \phi)$ and $\pi_l \in Ext(\pi_{l-1}, i + (l-1))$ for $n_{l-1} = Assoc(\pi_{l-1}, i + (l-1), \phi)$.
   (c) For $n_l = Assoc(\pi_l, i + l, \phi)$, the children are $Assoc(\pi_l, i + l, \phi_2)$ and $n_{l+1}$.
   (d) The nodes $Assoc(\pi_l, i + l, \phi_2)$ are root nodes of accepting runs integrated in $\rho$.

   Since there are accepting runs from $Assoc(\pi_l, i + l, \phi_2)$, the global induction hypothesis shows that $(\pi_l, i + l) \vDash \phi_2$ $\forall l$. We use this fact to construct an appropriate $\pi' \in Ext(\pi, i)$. For this purpose, we let $\pi'(i + l) = \pi_l(i + l)$ $\forall l$. By construction, $(\pi', i + l) \in Ext(\pi_l, i + l)$, and therefore Proposition 1 shows that $(\pi', i + l) \vDash \phi_2$ $\forall l$. Thus, $(\pi', i) \vDash \phi_1\ \mathcal{R}^g\ \phi_2$ and this directly implies $(\pi, i) \vDash \phi$.
10. $\phi = A\ \phi_1\ \mathcal{R}^g\ \phi_2$
    The proof is similar to the previous case.

11. $\phi = E \bigcirc^a \phi_1$
    Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. Since only rule 6 is available for this configuration, the direct child node must be of the form $([p, \phi_1], \Omega)$ and $Conf(\pi, i) \rightharpoonup_a (p, \Omega)$. This implies that there is $\pi' \in Ext(\pi, i)$ such that $Conf(\pi', succ_a(\pi', i)) = (p, \Omega)$. Therefore, $([p, \phi_1], \Omega) = Assoc(\pi', i+1, \phi_1)$ and since there is an accepting run from this configuration, $(\pi', i+1) \vDash \phi_1$ holds by the global induction hypothesis. Accordingly, $(\pi, i) \vDash \phi$ also holds.
12. $\phi = A \bigcirc^a \phi_1$
    Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. Since rule 7 is the only applicable rule, this shows $Conf(\pi, i) \in ADF_a$ and therefore $succ_a(\pi', i)$ is defined for all $\pi' \in Ext(\pi, i)$. The rest of the proof is similar to the previous case.
13. $\phi = E\ \phi_1\ \mathcal{U}^a\ \phi_2$
    Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. There are two possible cases. In the first case, $\rho$ has $Assoc(\pi, i, \phi_2)$ as a direct child node and there is an accepting run from $Assoc(\pi, i, \phi_2)$. The global induction hypothesis then implies $(\pi, i) \vDash \phi_2$ and therefore also $(\pi, i) \vDash \phi$. Otherwise, $\rho$ can be represented by a finite sequence of nodes $n_0 \dots n_m$ for $m \geq 1$ with the following properties:
    (a) The root node $n_0$ is $Assoc(\pi, i, \phi)$.
    (b) For any $l > 0$, there is $\pi_l$ such that $n_l = Assoc(\pi_l, succ_a^l(\pi_l, i), \phi)$ and $\pi_l \in Ext(\pi_{l-1}, succ_a^{l-1}(\pi_{l-1}, i))$ for $n_{l-1} = Assoc(\pi_{l-1}, succ_a^{l-1}(\pi_{l-1}, i), \phi)$.
    (c) For any $n_l$ with $l < m$, the children of $n_l$ are $Assoc(\pi_l, succ_a^l(\pi_l, i), \phi_1)$ and $n_{l+1}$.
    (d) For $n_m$, the only child node is $Assoc(\pi_m, succ_a^m(\pi_m, i), \phi_2)$.
    (e) $Assoc(\pi_l, succ_a^l(\pi_l, i), \phi_2)$ and $Assoc(\pi_m, succ_a^m(\pi_m, i), \phi_2)$ are the root nodes of accepting runs integrated in $\rho$.

    Let $j_l = succ_a^l(\pi_l, i)$. Since there are accepting runs from $Assoc(\pi_l, j_l, \phi_2)$ $\forall 0 \leq l < m$ and from $Assoc(\pi_m, j_m, \phi_2)$, the global induction hypothesis shows that $(\pi_l, j_l) \vDash \phi_1$ $\forall 0 \leq l < m$ and $(\pi_m, j_m) \vDash \phi_2$. Furthermore, $\pi_m \in Ext(\pi_l, j_l)$ $\forall 0 \leq l < m$. Proposition 1 then implies $(\pi_m, j_l) \vDash \phi_1$ $\forall 0 \leq l < m$ and therefore $(\pi_m, i) \vDash \phi_1\ \mathcal{U}^a\ \phi_2$. Since $\pi_m \in Ext(\pi, i)$, this proves $(\pi, i) \vDash \phi$.
14. $\phi = A\ \phi_1\ \mathcal{U}^a\ \phi_2$
    The proof is similar to the previous case.
15. $\phi = E\ \phi_1\ \mathcal{R}^a\ \phi_2$
    Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. We distinguish two cases. In the first case, a node $([p, \phi_1 \wedge \phi_2], \Omega'')$ is eventually reached in $\rho$ as a child of $Assoc(\pi'', i+k, \phi)$ for some $\pi'' \in Ext(\pi, i)$ and some $k$. The proof is then similar to case 13. In the second case, $\phi_1$ is never reached and $\rho$ can be represented by an infinite linear sequence of nodes $n_l$ with the following properties:
    (a) The root node $n_0$ is $Assoc(\pi, i, \phi)$.

(b) For any $l > 0$, there is $\pi_l$ such that $n_l = Assoc(\pi_l, succ_a^l(\pi_l, i), \phi)$ and $\pi_l \in Ext(\pi_{l-1}, succ_a^{l-1}(\pi_{l-1}, i))$ for $n_{l-1} = Assoc(\pi_{l-1}, succ_a^{l-1}(\pi_{l-1}, i), \phi)$.
(c) For $n_l = Assoc(\pi_l, succ_a^l(\pi_l, i), \phi)$, the children are $Assoc(\pi_l, succ_a^l(\pi_l, i), \phi_2)$ and $n_{l+1}$.
(d) The nodes $Assoc(\pi_l, succ_a^l(\pi_l, i), \phi_2)$ are root nodes of accepting runs integrated in $\rho$.

Let $j_l = succ_a^l(\pi_l, i)$. Using the global induction hypothesis, we obtain $(\pi_l, j_l) \vDash \phi_2 \ \forall l \geq 0$. Treating each $\pi_l$ as a function $\pi_l : \mathbb{N} \to P \times \Gamma^+$ represented by its function graph, we can define $\pi_l[0,k] = \{(0, \pi_l(0)), \ldots, (k, \pi_l(k))\}$ to be the prefix of $\pi_l$ up to index $k$. We then set $\pi' = \bigcup \pi_l[0, j_l]$. $\pi'$ thus is equal to $\pi_l$ up to index $j_l$ for every $l$ and this means $\pi' \in Ext(\pi_l, j_l) \ \forall l$. Proposition 1 then shows $(\pi', j_l) \vDash \phi_2 \ \forall l$ and this is equivalent to $(\pi', succ_a^l(\pi', i)) \vDash \phi_2 \ \forall l$. Therefore, $(\pi', i) \vDash \phi_1 \ \mathcal{R}^a \ \phi_2$ and thus $(\pi, i) \vDash \phi$.

16. $\phi = A \ \phi_1 \ \mathcal{R}^a \ \phi_2$
The proof is similar to the previous case.
17. $\phi = E \bigcirc_W^a \phi_1$
Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. There are two cases to consider. In the first case, the only child node of $\rho_0$ has the form $([p, \phi_1], \Omega)$ and then the proof is similar to case 11. In the second case, the only child node is *true* and therefore rule 16 must have been used to derive it. Then $Conf(\pi, i) \notin ADF_a$ and therefore there is $\pi' \in Ext(\pi, i)$ such that $succ_a(\pi', i)$ is undefined. This shows $(\pi, i) \vDash \phi$.
18. $\phi = A \bigcirc_W^a \phi_1$
Similar to the previous case.
19. $\phi = E \ \phi_1 \ \mathcal{U}_W^a \ \phi_2$
Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. There are two possible cases. In the first case, a node $([p, \phi_2], \Omega'')$ is eventually reached in $\rho$ as a child node of $Assoc(\pi'', i+k, \phi)$ for $\pi'' \in Ext(\pi, i)$ and some $k$. The proof is then similar to case 13. Otherwise, $\rho$ can be represented by a finite sequence of nodes $n_0 \ldots n_m$ for $m \geq 1$ with the following properties:
(a) The root node $n_0$ is $Assoc(\pi, i, \phi)$.
(b) For any $l > 0$, there is $\pi_l$ such that $n_l = Assoc(\pi_l, succ_a^l(\pi_l, i), \phi)$ and $\pi_l \in Ext(\pi_{l-1}, succ_a^{l-1}(\pi_{l-1}, i))$ for $n_{l-1} = Assoc(\pi_{l-1}, succ_a^{l-1}(\pi_{l-1}, i), \phi)$.
(c) For any $n_l$ with $l < m$, the children of $n_l$ are $Assoc(\pi_l, succ_a^l(\pi_l, i), \phi_1)$ and $n_{l+1}$.
(d) For $n_m$, the children are *true* and $Assoc(\pi_m, succ_a^m(\pi_m, i), \phi_1)$.
(e) $Assoc(\pi_l, succ_a^l(\pi_l, i), \phi_1)$ is the root node of an accepting run integrated into $\rho$.

Let $j_l = succ_a^l(\pi_l, i)$. Since there are accepting runs from $Assoc(\pi_l, j_l, \phi_1)$ $\forall 0 \leq l \leq m$, the induction hypothesis implies $(\pi_l, j_l) \vDash \phi_1 \ \forall 0 \leq l \leq m$. Furthermore, since *true* is a child of $n_l$ and the only possible rule for this connection is rule 19, we obtain $Conf(\pi_m, j_m) \notin ADF_a$. Accordingly, there is $\pi_{m+1} \in Ext(\pi_m, j_m)$ such that $succ_a(\pi_{m+1}, j_m)$ is undefined.

Naturally, $\pi_{m+1} \in Ext(\pi_l, j_l)\ \forall l \leq m$ and therefore $(\pi_{m+1}, j_l) \vDash \phi_1\ \forall l \leq m$ by Proposition 1. This shows $(\pi_{m+1}, i) \vDash \phi_1\ \mathcal{U}^a_W\ \phi_2$. Furthermore, $\pi_{m+1} \in Ext(\pi, i)$ and therefore $(\pi, i) \vDash \phi$.

20. $\phi = A\ \phi_1\ \mathcal{U}^a_W\ \phi_2$
    Similar to the previous case.
21. $\phi = E\ \phi_1\ \mathcal{R}^a_W\ \phi_2$
    Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. There are two possible cases. In the first case, the accepting run either reaches a node with control location $[p', \phi_1 \wedge \phi_2]$ as a child of a node with control location $[p, \phi]$ or control locations $[p, \phi]$ are reached infinitely often. The proof is then similar to case 15. In the second case, a node $([p, \phi_2], \Omega'')$ is reached as a child of $Assoc(\pi'', i + k, \phi)$ for some $\pi'' \in Ext(\pi, i)$ and the node was derived using rule 23. But then, the proof is similar to case 19.
22. $\phi = A\ \phi_1\ \mathcal{R}^a_W\ \phi_2$
    Similar to the previous case.
23. $\phi = E\ \bigcirc^-\ \phi_1$
    Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi)$. $\rho$ has the following structure:
    (a) The root node $n_0$ is $Assoc(\pi, i, \phi) = ([p, \phi], \gamma(\gamma', (p', \tau))\Omega)$.
    (b) The only child node of $n_0$ is $n_1 = ([p_c, \phi], (\gamma', (p', \tau))\Omega)$.
    (c) The only child node of $n_1$ is $n_2 = ([p', \phi_1], \tau\Omega)$ and further $([p', \phi_1], \tau\Omega) = Assoc(\pi, succ_-(\pi, i), \phi_1)$ holds.

    This structure follows from the fact that only rule 27 is available for $Assoc(\pi, i, \phi)$ and only rule 29 is available for $n_1$. Since there is an accepting run from $n_2$, the induction hypothesis implies $(\pi, succ_-(\pi, i)) \vDash \phi_1$ and therefore $(\pi, i) \vDash \phi$.
24. $\phi = E\ \bigcirc^-_W\ \phi_1$
    Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi) = ([p, \phi], \gamma\Omega)$. If $\rho$ contains *true* as an immediate child of $Assoc(\pi, i, \phi)$, then $\gamma = \#$ and $\Omega = \varepsilon$ which implies $\pi(i) = (p, \#)$ (otherwise rule 28 is not available) and the claim holds. Otherwise, only rule 27 is available and $\rho$ has $([p_c, \phi], \xi\Omega')$ as the only child of $Assoc(\pi, i, \phi)$. In the first case, $\xi = \#$ and $\Omega' = \varepsilon$ and therefore the only child node of $([p_c, \phi], \xi)$ is *true*. This implies the caller is undefined and the claim holds. In the second case, $\xi = (\gamma', (p', \tau))$, the only child of $([p_c, \phi], \xi\Omega')$ is $([p', \phi_1], \tau\Omega')$ and there is an accepting rum from this node. This case is similar to 23.
25. $\phi = E\ \phi_1\ \mathcal{U}^-\ \phi_2$
    Let $\rho$ be an accepting run from $Assoc(\pi, i, \phi) = ([p, \phi], \Omega)$. We use a local induction on $|\Omega|$.

    **Basis:**

    The case $|\Omega| = 0$ is impossible, so consider $|\Omega| = 1$. Then $\Omega = \#$. Rule 30 is the only applicable rule, implying there is only one child node $n_1 = Assoc(\pi, i, \phi_2)$ of $n_0$ and there is an accepting run from $n_1$. The global induction hypothesis implies $(\pi, i) \vDash \phi_2$ and therefore $(\pi, i) \vDash \phi$.

**Local Induction Hypothesis:**

For any $j$, if there is an accepting run from $Assoc(\pi, j, \phi) = ([p, \phi], \Omega')$ and $|\Omega'| < |\Omega|$, then $(\pi, j) \vDash \phi$.

**Step:**

Let $|\Omega| > 1$. If $Assoc(\pi, i, \phi)$ has a child node $([p, \phi_2], \Omega)$, the claim again follows directly from the global induction hypothesis. Let us therefore assume that the children are $([p, \phi_1], \Omega)$ and $([p,\ E \bigcirc^- (\phi_1\ \mathcal{U}^-\ \phi_2)],\ \Omega)$, which is the only other possibility due to rule 30. From both children, there are accepting runs $\rho_1$ and $\rho_2$ and the global induction hypothesis directly implies $(\pi, i) \vDash \phi_1$. Consider now the accepting run $\rho_2$. Just as in case 23, only rules 27 and 29 can have been applied consecutively to obtain $\rho_2$. Therefore, $\rho_2$ has the following structure:

(a) The root node $n_0$ is $Assoc(\pi, i, E \bigcirc^- \phi) = ([p,\ E \bigcirc^- \phi],\ \gamma(\gamma',\ (p',\ \gamma''))\Omega')$.

(b) The only child node of $n_0$ is $n_1 = ([p_c, E \bigcirc^- \phi], (\gamma', (p', \gamma''))\Omega')$.

(c) The only child node of $n_1$ is $n_2 = ([p', \phi], \gamma''\Omega')$ and further $([p', \phi], \gamma''\Omega') = Assoc(\pi, succ_-(\pi, i), \phi)$ holds.

Let $\Omega'' = \gamma''\Omega'$. Naturally, $|\Omega''| < |\Omega|$ and therefore the local induction hypothesis shows $(\pi, succ_-(\pi, i)) \vDash \phi$. Combining both results, we obtain $(\pi, i) \vDash \phi$.

26. $\phi = E\ \phi_1\ \mathcal{R}^-\ \phi_2$

Similar to the previous case. In particular, notice again that the accepting run $\rho$ cannot be constructed by visiting control locations $[p, \phi]$ infinitely often because they are not accepting. □