Resource Reachability Games on Pushdown Graphs

Martin Lang*

RWTH Aachen University, Lehrstuhl für Informatik 7, D-52056 Aachen, Germany
lang@automata.rwth-aachen.de

Abstract. We consider two-player reachability games with additional resource counters on arenas that are induced by the configuration graphs of pushdown systems. For a play, we define the resource cost to be the highest occurring counter value. In this way, we quantify resources and memory that player 0 needs to win. We introduce the bounded winning problem: Is there a uniform bound k such that player 0 can win the game from a set of initial configurations with this bound k? We provide an effective, saturation-based method to solve this problem for regular sets of initial and goal configurations.

1 Introduction

Pushdown automata have become an important tool in the formal analysis and verification of recursive programs. Since their introduction by A.G. Oettinger in 1961 and M.-P. Schützenberger in 1963, they have been intensively studied and are relatively well-understood today. Pushdown automata without input alphabet, which only operate with their control states on the stack, are usually called *pushdown systems*. The configuration graphs of such systems are called *pushdown graphs*. They can be used as a formal model for recursive programs because they combine good expressive power with an (efficiently) decidable point-to-point reachability problem. An example of their application in the area of formal verification is the model checker jMoped [14], which uses symbolic pushdown systems to verify Java bytecode.

However, mere reachability on transition systems lacks the possibility to model an environment system or possible user input. This can be achieved by two-player games on graphs. Such games were studied in the course of the controller synthesis problem proposed by A. Church in [8], and many positive algorithmic results are known today for games on finite graphs. Moreover, two player games on pushdown graphs with ω-regular winning conditions were solved in [15] by I. Walukiewicz. Later, T. Cachat showed in [4] that the well-known saturation approach for pushdown system reachability can be extended to reachability and Büchi games on pushdown graphs.

* Supported by DFG research grant *Automatentheoretische Verifikationsprobleme mit Ressourcenschranken*

A. Muscholl (Ed.): FOSSACS 2014, LNCS 8412, pp. 195–209, 2014.

Recently, several models of games with additional resource constraints were introduced to provide a model for systems with resource consumption. In this context, the resources are usually modeled by integer counters that can be modified by the players but not read during the game. In addition to usual winning conditions such as Büchi or Parity, the winning conditions of these games restrict the values of the resource counters throughout the game. Typically, it is required that these values are bounded by a global limit. Examples for such games are energy games (cf. [5]), energy parity games (cf. [6]), or consumption games (cf. [3]). However, all these previous games are defined over finite graphs. Only very recently, and independently from the author, games with such a structure were considered on graphs induced by pushdown systems (cf. [7]). Although the game model considered there is essentially the same, the questions we solve and the methods we use are quite different from those in [7].

In this work, we consider *resource pushdown systems*. These are pushdown systems that are extended with a finite set of non-negative integer counters. They can be used to model recursive programs with resource consumption. We examine two-player games on the configuration graphs of these systems. Every resource counter can be modified by the pushdown rules either by incrementing it (for short `i`), or resetting it to zero (for short `r`). Moreover, it is possible to leave a counter unchanged (no operation or `n`). The counters cannot be read during the game. This reflects a step-by-step consumption and all-at-once replenishment model of resources. The form of the counters is the same as in the ωB-games considered in [7] and very similar to the model used in consumption games (cf. [3]) on finite graphs. It is also used in the model of B-automata (cf. [9]).

We introduce reachability games with an additional bound on the resource counters – called *resource reachability games*. We fix some *resource bound* $k \in \mathbb{N}$. In a play on the configuration graph of a resource pushdown system, the counters are updated according to the operations associated with the used pushdown rules. In order to win the game w.r.t. the bound k, player 0 does not only have to reach a certain set of goal configurations F but also needs to ensure that all counter values throughout the play stay below k. We examine this kind of game and present a method to compute the winning region and winning strategies for player 0 in the case of regular sets of goal configurations. Furthermore, we investigate the *bounded winning problem*. Given a set A of initial configurations. Is there a *uniform* resource bound $k \in \mathbb{N}$ such that player 0 wins the resource reachability game from all configurations in A w.r.t. this bound k? In order to solve this problem, we thoroughly analyze the propagation of counter operations in the saturation approach. Thereby, we can extend the saturation idea introduced by T. Cachat to effectively translate the question of winning cost into a membership query for alternating B-automata. In total, we reduce the bounded winning problem to a boundedness problem for B-automata. In contrast to [7], we consider only finite plays and are interested in finding *uniform* bounds that can be respected by all plays starting from sets of initial configurations. In [7], infinite games are considered with an interest in checking for a given initial configuration whether for each play there is an individual bound.

This work is grouped into three parts. First, we fix the notation and present the preliminaries. Second, we formally introduce resource pushdown systems and resource reachability games. Furthermore, we investigate some general properties of the games and calculate the winning region of player 0 for a given resource bound. In the third part, we consider the bounded winning problem and present our solution approach.

2 Preliminaries

For a set Σ, we denote the set of finite sequences (or words) over Σ by Σ^*. For a word $w \in \Sigma^*$, we write $w(i)$ for the i-th letter in the word (zero indexed). For sets A, B, we write B^A for the set of all functions from A to B and $B_{\mathfrak{p}}^A$ for the set of all partial functions from A to B. We write $\perp$ to indicate that a partial function is undefined on some value.

The fundamental model that we consider are two-player games on graphs. A game graph $\mathcal{A} = (V, E)$ is directed and its vertices V are partitioned into two sets V_0, V_1 indicating to which player the vertex belongs. Such a game graph is often called *arena*. The two players are called 0 (or Eve) and 1 (or Adam). A *play* of a game on $\mathcal{A}$ is a (possibly infinite) sequence of moves in which the two players move a game pebble across the graph. A play starts in some vertex $v \in V$. In each step of the game, the player to which the current vertex belongs can move the pebble to one of the successor vertices. Formally, we say a play is the sequence τ of edges along which the pebble is moved.

The winner of a fixed play in a game is determined by the so-called *winning-condition*. In a reachability game, we fix a *goal set* $F \subseteq V$ of vertices in the game graph. Player 0 wins the reachability game if the play visits a vertex from F after a finite number of moves. Otherwise, player 1 wins the game. In general, we are interested in knowing whether one of the players can force to win (independent of how the other player moves) by following a certain *strategy*. A strategy for player i is a mapping σ that maps all past moves ($\in E^*$) of the play to the next edge to take whenever the current position is a vertex of V_i. A strategy is called *winning* for player i if all plays in which player i moves according to the strategy are winning for player i. A strategy is called *finite memory* if it can be implemented by a finite state Mealy machine that reads all the moves of the opponent and outputs the next move of the respective player. It is called *memoryless* or *positional* if the next move only depends on the current vertex. We call the vertices from which player i has a winning strategy the *winning region* of player i. A game is called *determined* if for every starting vertex either player 0 or player 1 has a winning strategy. Reachability games are know to be determined and admit positional winning strategies for both players on their respective winning regions. A comprehensive introduction to two-player games and proofs for the claims above can be found, e.g., in [11].

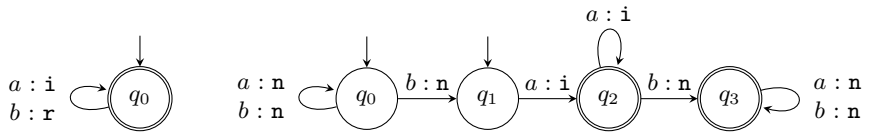

Fig. 1. Example B-automaton: left: count maximal length of uninterrupted a-block / right: count minimal length of uninterrupted a-block

2.1 Counters as Resource Model

We model resources by a finite set of non-negative integer counters. Each counter supports two kinds of operations. First, the counter can be incremented (for short `i`). This represents the usage of a single resource. Second, a counter can be reset to zero (for short `r`). This models the full replenishment of the resource. Additionally, we use `n` as a shorthand notation for no operation (the counter is left unchanged). The counters operate independently from each other. Thus, we can use multiple counters in order to model different types of resources. We associate the *resource usage* or *consumption* with the highest occurring counter value. This scheme of step-by-step consumption and all at once replenishment is motivated by scenarios such as battery driven systems or the usage of paper in a printer.

Finite state automata with similar counters have been studied in [1] (R-automata) and [9] (B-automata). In the context of this work, we use the model of B-automata and known results for this formalism as a tool. B-automata were introduced by T. Colcombet in [9] and extend finite state automata with a finite set of counters (denoted by Γ) as described above[1]. The counters can be manipulated by the transitions but not read by the automaton. Throughout a run, the counters are updated according to the used transitions. The value of a run is the maximal counter value (over all counters) that occurs in the run. B-automata naturally define a function from words to $\mathbb{N} \cup \{\infty\}$. For a B-automaton $\mathfrak{A}$ and a word w, we assign w to the infimum of the values of all accepting runs of $\mathfrak{A}$ on w and denote this value by $[\![\mathfrak{A}]\!](w)$. We also call it the (resource) cost of w. Note that $[\![\mathfrak{A}]\!](w) = \infty$ if there is no accepting run for w.

Figure 1 shows two examples of B-automata. Their semantics are to count the maximal (left) / minimal (right) number of subsequent letters a without interruption. The left automaton just increments for each letter a and resets the counter to zero when it reads a b. Consequently, the unique run of a word has the value of the longest uninterrupted block of as. The right automaton calculates the minimal length of a blocks by nondeterministically guessing the position of the minimal block in the word. It changes to q_1 when the block starts (or starts in q_1 if this block is at the beginning) and counts its length.

In the context of systems with resources modeled by counters as described above, we are especially interested in the question of *boundedness*: Is there a bound on the resource consumption for a given set of runs? A solution to this

[1] In difference to the original publication we do not use the counter operation *check* because it is not necessary for our work and simplifies the overall presentation.

question is part of the realizability problem since real world systems can only have limited resources. With formal verification in mind, we are especially interested in decidable variants of this question. The *boundedness problem* for B-automata is decidable, i.e., one can algorithmically check if there is a global bound $k \in \mathbb{N}$ for a given B-automaton $\mathfrak{A}$ such that for all words w, we have $[\![\mathfrak{A}]\!](w) \leq k$. In the case of multiple counters, this was first shown by D. Kirsten in [13] for a slightly more restrictive counter model (hierarchical counters). He also proved that this problem is PSPACE-hard. In the case of B-automata, the boundedness problem was solved by T. Colcombet in [9].

2.2 Counter Profiles

In order to provide a well understandable way to reason about sequences of counter operations, we introduce a structured representation in the form of a well-partially ordered monoid. This enables us to present our results more generally for systems that are annotated with such a structure and to emphasize which properties are needed to obtain the results. For sequences of counter operations, we introduce the notion of *counter profiles* and use this model instead of sequences of counter operations in the context of the bounded winning problem. A counter profile is a 3-tuple $(i^+_{\leftarrow}, c_{max}, i^+_{\rightarrow}) \in (\mathbb{N} \cup \{/\})^3$. It represents a sequence u of counter operations (from $\{\mathtt{i}, \mathtt{n}, \mathtt{r}\}^*$) with the following intuition. For the sake of simplicity, we assume that u does not contain any $\mathtt{n}$s since they have no influence on the counter. The component $i^+_{\leftarrow}$ represents the number of increments before the first reset, i.e., the largest $j \in \mathbb{N}$ such that $\mathtt{i}^j$ is a prefix of u. The component c_{max} represents the maximal counter value between two subsequent resets, i.e., the largest $j \in \mathbb{N}$ such that $\mathtt{ri}^j\mathtt{r}$ is an infix of u. Lastly, $i^+_{\rightarrow}$ represents the number of increments after the last reset, i.e., the largest $j \in \mathbb{N}$ such that $\mathtt{ri}^j$ is a suffix of u. If the sequence u contains only one (or even no) reset, the components c_{max} (and $i^+_{\rightarrow}$) are set to $/$ (read n/a). On these profiles, we define the concatenation $\circ$ such that it reflects the concatenation of counter sequences. One can see by checking all cases that all counter profiles together with the concatenation and $(0, /, /)$ as neutral element form a monoid. Each of the three base operations directly corresponds to a profile – $\mathtt{n}$ to $(0, /, /)$, $\mathtt{i}$ to $(1, /, /)$ and $\mathtt{r}$ to $(0, /, 0)$. By translating each operation into its profile and concatenating all the profiles along a run, one obtains a profile that provides the value of this run by its maximal entry as well as all information necessary to interpret the sequence as part of a longer sequence. As a result, we can use counter profiles as an equivalent representation for counter sequences.

In contrast to counter sequences, counter profiles offer a natural way to define a partial order. For two profiles p_1 and p_2 we say p_1 is less than or equal to p_2 (and write $p_1 \leq p_2$) if all components of p_1 are less than or equal to p_2 (component-wise order). In each component we use the canonical order on $\mathbb{N}$ and let the newly introduced $/$ be incomparable to all natural numbers. This order is a well-partial order in every component since it has neither infinitely decreasing chains nor infinite anti-chains. By a result of Higman (cf. [12]), we obtain that the component-wise order we defined on the counter profiles is a

well-partial order, too. We remark that the order is compatible with the monoid operation.

Systems with several counters can be represented by a vector of counter profiles. We extend the concatenation and the order to these vectors by applying the concatenation in each component and taking the component-wise order. Again by the result of Higman and simple checking, we obtain that these vectors of counter profiles still form a monoid and the order is a well-partial order. For a set of such vectors of profiles or a set of profiles A, we denote the set of maximal elements of A by $\max A$. We remark that $\max A$ may contain several elements because the order is not total. However, by definition of maximal, $\max A$ is an anti-chain and thus finite.

3 Resource Reachability Games

We introduce pushdown systems with a finite set of counters as model for recursive programs with resource consumption. These counters follow the previously described ideas and provide a way to model step-by-step usage and all at once replenishment of several resource types during the execution of recursive programs.

Definition 1. *A* resource pushdown system *is a 4-tuple $\mathcal{P} = (P, \Sigma, \Delta, \Gamma)$ where P is a finite set of control states, Σ is a finite stack alphabet, $\Delta \subseteq P \times \Sigma \times \Sigma^* \times P \times \{\mathtt{i}, \mathtt{r}, \mathtt{n}\}^\Gamma$ is a finite transition relation and Γ is a finite set of counters.*

Similar to normal pushdown systems, a configuration of a resource pushdown system is a pair of a state from P and a finite word from Σ^*. The successor relation on configurations is defined similar to normal pushdown systems. We additionally associate this step of the system with the counter operation f of the transition used. Formally, for two configurations $pu, qv \in P \times \Sigma^*$, we say qv is an f-successor of pu and write $pu \vdash_f qv$ if there is a common suffix w and a transition $(p, a', v', q, f) \in \Delta$ such that $u = a'w$ and $v = v'w$. We denote the configuration graph of $\mathcal{P}$ by $\mathcal{C}_\mathcal{P} = (P \times \Sigma^*, \vdash)$. In our examples, we use systems with only one counter and write, e.g., $pa \xrightarrow{\mathtt{i}} qv$ as a shorthand notation for a transition (p, a, v, q, f) where f maps the unique counter to $\mathtt{i}$. Analogously, we write $pu \vdash_\mathtt{i} qv$ if $pu \vdash_f qv$ with $f(c) = \mathtt{i}$ for the unique counter c.

We obtain a game arena from the configuration graph of a resource pushdown system by providing an additional partition of the state space $P = P_0 \uplus P_1$. Configurations with a state in P_i belong to player i. A game on this arena is played as in the classical case but each move additionally provides counter operations according to the corresponding pushdown rule. As for B-automata, we simulate the counters along the play and associate the *resource consumption* at every point in the play with the highest counter value that occurred so far. On such arenas, we consider a combined reachability and resource limit objective. Let F be a set of goal configurations and $k \in \mathbb{N}$ be a resource limit. Player 0 wins the *resource reachability game* with respect to F and k if the play reaches a configuration in F and the resource consumption at this point is at most k.

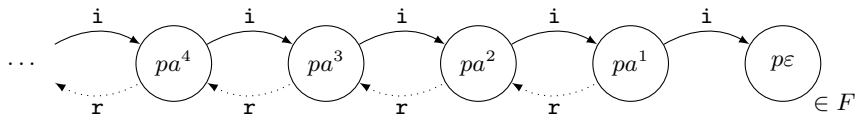

Fig. 2. Example for the configuration graph of a simple resource pushdown system

For these games, we consider different kinds of winning regions. First, we consider the resource independent winning region of player 0 denoted by $W_0(F)$. A configuration pw is in $W_0(F)$ if player 0 can reach F from pw with arbitrarily high resource consumption. Second, we consider the winning region with resource limit k denoted by $W_0^{(k)}(F)$. A configuration pw is in $W_0^{(k)}(F)$ if player 0 wins the resource reachability game with the respective limit k on the resource consumption. This second, new type of winning region immediately yields two algorithmic questions:

1. We fix F and $k \in \mathbb{N}$ and ask what is $W_0^{(k)}(F)$?
2. We fix a set A and ask whether there is a uniform resource bound k such that player 0 wins the resource reachability game with bound k from A, i.e., whether $A \subseteq W_0^{(k)}(F)$. We call this problem the *bounded winning problem*.

We illustrate the newly introduced concepts with the following example. Consider a resource pushdown system $\mathcal{P} = (P, \Sigma, \Delta, \Gamma)$ with only one state $p \in P$, the stack alphabet $\Sigma = \{a\}$ and only one pushdown rule $pa \xrightarrow{\mathtt{i}} p\varepsilon \in \Delta$. Figure 2 (without the dotted transitions) shows a part of the configuration graph of $\mathcal{P}$. On this configuration graph, we compare the different winning regions for the resource reachability game in which all configurations belong to player 0 and the goal set is $F = \{p\varepsilon\}$. First, the resource independent winning region is $W_0(F) = \{pa^n \mid n \in \mathbb{N}\}$ because one can remove all letters a from the stack by successively applying the rule $pa \xrightarrow{\mathtt{i}} p\varepsilon$. However, each such step costs one increment of the resource counter. Hence, the winning region with resource bound k is $W_0^{(k)}(F) = \{pa^n \mid n \leq k\}$. Consequently, there is no uniform bound k such that player 0 wins on complete $W_0(F)$ with this bound.

Now, we add the pushdown rule $pa \xrightarrow{\mathtt{r}} paa$ to Δ of $\mathcal{P}$. Then, the configuration graph includes the dotted transitions in Figure 2. This does not change $W_0(F)$ in the considered resource reachability game but reduces the resources needed to reach F from an arbitrary configuration to 2. For instance, let us start at configuration pa^3. The sequence $pa^3 \vdash_{\mathtt{i}} pa^2 \vdash_{\mathtt{i}} pa \vdash_{\mathtt{r}} pa^2 \vdash_{\mathtt{i}} pa \vdash_{\mathtt{i}} p\varepsilon$ shows that F is reachable with a resource bound of 2. This idea of incrementing two times and then resetting one time can easily be extended to all configurations. Thus, we obtain that 2 is a uniform bound such that player 0 wins the resource reachability game, i.e., $W_0^{(2)}(F) = W_0(F)$. This example already shows that we generally cannot expect to obtain memoryless winning strategies for player 0 in resource reachability games. Moreover, it illustrates that memoryless strategies cannot obtain minimal resource bounds even on finite graphs.

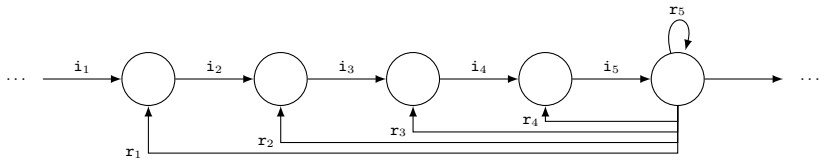

Fig. 3. Exponential memory in the number of counters is unavoidable to achieve the best resource-limit possible

In the following, we solve the two algorithmic questions for the case that F and A are regular. First, we consider the problem of calculating $W_0^{(k)}(F)$ for a fixed F and k. This problem can be reduced to solving (normal) reachability games on pushdown graphs. The reduction is based on the idea of simulating the counters up to the (finite) value k in the state space of the pushdown system. With standard techniques (see e.g. [11]), we can obtain the winning region and a winning strategy for the original game. We formalize this idea by

Proposition 2. *Let $\mathcal{P} = (P_0 \uplus P_1, \Sigma, \Delta, \Gamma)$ be a resource pushdown system. Let F be a regular goal set and $k \in \mathbb{N}$ a resource bound for the bounded reachability game on the configuration graph of $\mathcal{P}$. One can effectively compute the winning region $W_0^{(k)}(F)$ and a corresponding finite memory winning strategy.*

In a similar way as previously described, we obtain winning strategies for player 1 for all configurations in $P \times \Sigma^* \setminus W_0^{(k)}(F)$. As a direct consequence, we obtain that resource reachability games are determined. We remark that this idea can be easily extended to all ω-regular winning conditions.

The strategy obtained from the above reduction uses a memory structure that is exponential in the number of counters. The example in Figure 3 shows that this is generally unavoidable if the strategy should achieve the lowest possible resource bound. We use 5 counters in the example and denote the increment/reset of counter j by $\mathtt{i}_j/\mathtt{r}_j$. While it is possible to get through the shown gadget with resource limit 1 and all counters reset to zero before leaving, the strategy of player 0 has to store the state of all counters in order to achieve this ($2^5 = 32$). Nevertheless, if we allow a resource limit of 5, all counters can be reset to zero before leaving the gadget with a memory structure of size 6.

4 The Bounded Winning Problem

In this section, we first show that the bounded winning problem is at least as complex as solving the boundedness problem for B-automata. As already mentioned, this is known to be PSPACE-hard (cf. [13]) even for a slightly simpler version of automata (with hierarchical counters). Furthermore, there is a 2-EXPSPACE algorithm (cf. [1]) to solve it. Subsequently, we introduce an alternating variant of automata with monoid annotations, such as counter profiles. This model enables us to extend a saturation-based solution approach for normal reachability games on pushdown graphs to resource reachability games.

Proposition 3. *The bounded winning problem is at least as complex as the boundedness problem for B-automata.*

Proof. Let $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, \mathrm{Fin}, \Gamma)$ be a B-automaton. We define a resource pushdown system with the following idea in mind: The pushdown system simulates the automaton by letter-wise consuming the stack contents and simulating the operation of $\mathfrak{A}$ in its state space and with its counters. Formally, we define the resource pushdown system by $\mathcal{P} = (Q, \Sigma, \Delta', \Gamma)$ where

$$\Delta' := \{(p, a, \varepsilon, q, f) \mid (p, a, q, f) \in \Delta\}$$

With this definition, we obtain for all words $w \in \Sigma^*$ the equivalence that $[\![\mathfrak{A}]\!](w) \leq k$ iff there is is a $q_f \in \mathrm{Fin}$ and a sequence of pushdown operations to reach $q_f\varepsilon$ from $q_0 w$ with resource limit k. Consequently, there is a global bound k such that for all words $w \in \Sigma^* : [\![\mathfrak{A}]\!](w) \leq k$ iff there is a bound k such that one can reach $\mathrm{Fin} \times \{\varepsilon\}$ from all configurations in $q_0\Sigma^*$ with bound k. That is exactly the bounded winning problem on $\mathcal{P}$ for $P_0 = Q, P_1 = \emptyset$, $F = \mathrm{Fin} \times \{\varepsilon\}$ and $A = \{q_0 w \mid w \in \Sigma^*\}$. □

The main tool in constructing our saturation method for resource reachability games is the model of alternating automata with B-automaton like counters. We introduce and argue on the base of a slightly more general model with annotations from well-partially ordered monoids. This shows the properties we use more clearly, and simplifies the presentation. In the analysis of resource reachability games, we instantiate the model with (vectors of) counter profiles as a formalism equivalent to B-automaton counters. We model the alternation by nondeterministic choice among transitions with possibly several target states. A run of the alternating automaton has the form of a tree. For the transition chosen, the run has to be continued from all target states of the respective transition. This is an explicit presentation of the otherwise often used positive boolean formula notation for transitions of alternating automata. It has the advantage, for our purpose, that we can associate the different paths in the automaton with different annotations more easily. This is needed to reflect the multiple choices in the game. Formally, we have

Definition 4. *An* annotated alternating automaton *is a tuple* $\mathfrak{A} = (Q, \Sigma, \mathrm{In}, \Delta, F, \mathcal{M})$. *The components* Q, Σ, In *and* F *are defined as usually for automata.* $\mathcal{M} = (M, \circ, e_{\mathcal{M}}, \leq)$ *is a well-partially ordered monoid. The transition relation* Δ *is a finite set* $\Delta \subseteq Q \times \Sigma \times (\mathsf{AntiChain}(M))_{\mathfrak{p}}^{Q}$ *where* $\mathsf{AntiChain}(M)$ *is the set of all anti-chains with elements from* M. *For a transition* $t = (p, a, f)$, *we define the successor states of the transition by* $\mathsf{Succ}(t) := \mathrm{dom}(f)$. *The automaton is called normalized, if states in* In *have no ingoing transitions.*

In order to define a run and the (annotation) values associated with the run, we need the notion of a tree.

Definition 5. *A tree* $\mathfrak{T}$ *consists of a set of nodes* T, *a root node* $t_0 \in T$ *and a child (or successor) function* $s_{\mathfrak{T}} : T \to \mathcal{P}ow(T)$ *such that:*

1. *for every node* $v \in T \setminus \{t_0\}$ *there is a unique parent node* $p \in T$ *such that* $v \in s_{\mathfrak{T}}(p)$.
2. *the child function has no loops, i.e., there is no sequence* $v_0, v_1, \ldots, v_n$ *with* $v_{i+1} \in s_{\mathfrak{T}}(v_i)$ *for all* $i = 1, \ldots, n-1$ *such that* $v_0 = v_n$.
3. *every node is reachable from the root, i.e., for all nodes* $v \in T$ *there is a sequence* $t_0 = v_0, \ldots, v_n = v$ *such that* $v_{i+1} \in s_{\mathfrak{T}}(v_i)$.

We use the following common operations on trees. The parent function $\pi_{\mathfrak{T}} : T \setminus \{t_0\} \to T$ maps all nodes but the root to their unique parent nodes. The distance function $d_{\mathfrak{T}} : T \to \mathbb{N}$ maps every node v to its distance from the root node. The leafs of a tree are denoted by $\mathsf{Leafs}_{\mathfrak{T}} = \{v \in T \mid s_{\mathfrak{T}}(v) = \emptyset\}$. A level of a tree is a maximal set of nodes $T' \subseteq T$ that all have the same distance from the root, i.e., for $v, v' \in T'$ we always have $d_{\mathfrak{T}}(v) = d_{\mathfrak{T}}(v')$.

A run of an alternating automaton on a word $w = a_1 \ldots a_n$ follows the idea of an inductive tree construction. It starts with the root node and associates this node with the initial state of the automaton. Then, a transition a $(p, a_1, f) \in \Delta$ is selected and child nodes are created for all states in $q \in \mathrm{dom}(f)$ with their different annotations $f(q)$. For all child nodes, this construction continues on the rest of the word $a_2, \ldots, a_n$. A run is called accepting if all the leaf nodes of the tree are associated with final states of the automaton. Moreover, such a run yields values from the annotation monoid by multiplying the annotations along each path. We formalize this idea in the following two definitions.

Definition 6. *A run of an annotated alternating automaton* $\mathfrak{A}$ *on a word* w *is a 4-tuple* $\rho = (\rho_Q, \rho_\Delta, \rho_M, \mathfrak{T})$ *of three labeling functions and a tree* $\mathfrak{T}$. *The function* $\rho_Q : T \to Q$ *is called state labeling function. The function* $\rho_\Delta : T \setminus \mathsf{Leafs}_{\mathfrak{T}} \to \Delta$ *is called transition labeling function. The function* $\rho_M : T \setminus \{t_0\} \to M$ *is the annotation labeling function. They satisfy the following consistency properties:*

1. $\rho_Q(t_0) \in \mathrm{In}$
2. *The state labeling and the transition labeling are consistent with each other and with the word* w*: For all* $v \in T \setminus \mathsf{Leafs}_{\mathfrak{T}}$ *with labeled state* $\rho_Q(v) = q$ *and selected transition* $\rho_\Delta(v) = t = (p, a, f)$ *we have* $w(d_{\mathfrak{T}}(v)) = a$, $q = p$, $\rho_Q(s_{\mathfrak{T}}(v)) = \mathsf{Succ}(t)$
3. *For each node* $v \in T \setminus \mathsf{Leafs}_{\mathfrak{T}}$ *with* $\rho_\Delta(v) = t$ *and every state* $q \in \mathsf{Succ}(t)$, *there is exactly one child per annotation. Let* $V_q = \{v' \in s_{\mathfrak{T}}(v) \mid \rho_Q(v') = q\}$. *We have* $|V_q| = |f(q)|$ *and* $\rho_M(V_q) = f(q)$.

We call a run partial *if it does not start in* In, *and call it* accepting *if* $\rho_Q(\mathsf{Leafs}_{\mathfrak{T}}) \subseteq F$. *If there is an accepting run of* $\mathfrak{A}$ *on* w, *we write* $w \in L(\mathfrak{A})$. *Moreover, if there is a run of* $\mathfrak{A}$ *on* w *with* $\rho_Q(\mathsf{Leafs}_{\mathfrak{T}}) \subseteq S$, *we write* $w \in L_S(\mathfrak{A})$.

Definition 7. *Let* $\rho = (\rho_Q, \rho_\Delta, \rho_M, \mathfrak{T})$ *be a runtree of an automaton* $\mathfrak{A} = (Q, \Sigma, \mathrm{In}, \Delta, F, \mathcal{M})$. *For each* $q \in \rho_Q(\mathsf{Leafs}_{\mathfrak{T}})$, *we define the value of* ρ *inductively over the runtree.*

$$\mathsf{val}^q_\rho(v) = \begin{cases} \{e_{\mathcal{M}}\} & \text{if } \rho_Q(v) = q \\ \emptyset & \text{otherwise} \end{cases} \qquad \text{if } v \in \mathsf{Leafs}_{\mathfrak{T}}$$

$$\mathsf{val}^q_\rho(v) = \max\{m_1 \circ m_2 \mid v' \in s_{\mathfrak{T}}(v), m_1 = \rho_M(v'), m_2 \in \mathsf{val}^q_\rho(v')\} \qquad \text{otherwise}$$

Additionally, we define the total value $\mathsf{val}_\rho(v) = \max \bigcup_{q \in \rho_Q(\mathsf{Leafs}_{\mathfrak{T}})} \mathsf{val}^q_\rho(v)$. *We write* $\mathsf{val}^q(\rho)$ *as a shorthand for* $\mathsf{val}^q_\rho(t_0)$ *and* $\mathsf{val}(\rho)$ *as a shorthand for* $\mathsf{val}_\rho(t_0)$.

In the context of resource reachability games on pushdown graphs, we take the annotation monoid $\mathcal{M}$ to be the vector of counter profiles with its dimension matching the number of counters. Furthermore, we use the idea of P-automata to read pushdown configurations. For a pushdown system $\mathcal{P}$ with control states P and an automaton $\mathfrak{A}$ reading configurations from $\mathcal{P}$, we assume that P is part of the state space of $\mathfrak{A}$. A run of $\mathfrak{A}$ on some pushdown configuration pw then starts in state p of $\mathfrak{A}$. This way $\mathfrak{A}$ operates only on Σ and we do not need to distinguish the different kinds of input symbols (from P and Σ).

The traditional saturation approach for reachability in pushdown systems gradually extends a finite automaton with transitions that enable the automaton to simulate replacement steps of the pushdown system. It starts with a P-automaton $\mathfrak{A}$ that recognizes some set F of pushdown configurations to be reached. For a pushdown rule $pa \to qu$ it searches states q' that can be reached in $\mathfrak{A}$ when reading the configuration qu. Then, an a-transition from p is added to q'. This new transition enables the automaton to behave as if it had read qu although it actually read pa. Hence, the automaton can now simulate the pushdown rule $pa \to qu$. This is repeated until no more transitions can be added. The resulting automaton recognizes the set of all predecessor configurations of F. A complete presentation of this basic idea can be found in [2].

In [4], T. Cachat used alternating automata to lift this basic idea to reachability games. His approach uses the similarities between games and the semantics of alternating automata in the following way. Let $\mathfrak{A}$ be an alternating automaton recognizing a regular goal set F of the reachability game. Consider a player 1 state p of the pushdown system and let $pa \to q_1u_1, \ldots, pa \to q_nu_n$ be all pushdown rules originating in p with letter a on top of the stack. The saturation method now looks for states q'_i in $\mathfrak{A}$ that can be reached when reading q_iu_i ($i \in \{1, \ldots, n\}$) on $\mathfrak{A}$. It then adds an a-transition from p to $\{q'_1, \ldots, q'_n\}$ in $\mathfrak{A}$. By the semantics of alternating automata, a run of the automaton is continued in all target states of the transition. This reflects the fact that player 1 can choose among $pa \to q_1u_1, \ldots, pa \to q_nu_n$ and player 0 has to be able to reach the goal set F for all possible choices in order to win the reachability game. The case of player 0 states can be handled similar to the case of mere reachability. Again, this procedure is repeated until no more transitions can be added to the automaton. The resulting alternating automaton recognizes the winning region of player 0.

We extend this idea with the overall goal of constructing an automaton with counters that recognizes a pushdown configuration with cost k iff player 0 has

a strategy to win the resource reachability game with a resource limit of k. To realize this, the designed saturation method has to keep track of the resource counter operations executed by the simulated pushdown transitions. We use the monoid annotation of the automaton to store the counter profile associated with the resource counter operation of the simulated pushdown rule. The major difficulty arises from the fact that there are incomparable choices in the game. While it is easy to see that two increments are better than three, there are (even for finite game graphs) situations that do not have a unique best choice. For example, consider a resource pushdown system with two counters and two nearly identical pushdown rules. The first rule resets the first counter and increments the second whereas the other increments the first and resets the second. The decision which one is better depends on the context in this case. Thus, the designed method has to represent all such situations.

We illustrate the intuition behind newly added transitions with an example. Again, let $\mathfrak{A}$ be the automaton to be saturated. Consider a player 1 state p of the pushdown system and let $pa \xrightarrow{m_1} q_1u_1, \ldots, pa \xrightarrow{m_n} q_nu_n$ be all pushdown rules originating in p that can be applied with an a on top of the stack. For the sake of clarity, assume that there are linear (non-branching) runs of $\mathfrak{A}$ on q_iu_i. These runs end in a state q_i' and have an accumulated annotation m_i'. Then, the saturation procedure adds a transition $t = (p, a, f)$ with target states $\mathrm{dom}(f) = \{q_1', \ldots q_n'\}$. This part is identical to the situation without annotations. Similar to the target states of t, which represent the states into which player 1 can force player 0, the annotation in $\mathfrak{A}$ has to represent the possible annotations in the game that player 1 can enforce while reaching this state. Consequently, the annotation $f(q)$ for a target state $q \in \mathrm{dom}(f)$ is the maximum over all combined annotations $m_i \circ m_i'$ (first apply the pushdown rule, then the skipped part of the original run) of runs that end in a state q_i' equal to q.

In order to simplify the presentation of the saturation algorithm and the subsequent termination argument, we introduce an order on the transitions of the automaton and show that this order is a well-partial order. First, we order sets of elements from ordered monoids. Let $\mathcal{M}$ be a well-partially ordered monoid and $A, B \subseteq M$ two sets. We say A is dominated by B and write $A \leq B$ if for all elements $a \in A$ there is an element $b \in B$ such that $a \leq b$. Now, let $t = (p, a, f), t' = (p', a', f')$ be two transitions of an annotated alternating automaton. We order t and t' by $t \leq t'$ if $p = p'$, $a = a'$, $\mathrm{dom}(f) = \mathrm{dom}(f')$ and for all $q \in \mathrm{dom}(f)$ $f(q) \leq f'(q)$. With the finiteness of the states, the alphabet, and arguments about well-partial orders from [12], we deduce:

Lemma 8. *Let $\mathfrak{A} = (Q, \Sigma, \mathrm{In}, \Delta, F, \mathcal{M})$ be an annotated alternating automaton. The order on Δ is a well-partial order.*

With all previous preparations, we are now able to present the complete saturation procedure in Algorithm 1. It operates on arbitrary monoid annotations and assumes the resource pushdown system to be labeled with counter profiles. The resulting automaton and the arguments are stated in terms of counter profiles. To this end, we say that player 0 wins the resource reachability game with profile bound B if the resulting play has a combined counter profile p such that

Algorithm 1. Saturation procedure

input : resource pushdown system $\mathcal{P} = (P, \Sigma, \Delta_{\mathcal{P}})$, state partition $P = P_1 \uplus P_2$, normalized annotated alternating P-automaton $\mathfrak{A} = (Q, \Sigma, P, \Delta, F, \mathcal{M})$ (all annotations are $\{e_{\mathcal{M}}\}$, and for all $(q, a, f) \in \Delta \mid \mathrm{dom}(f)| = 1$)

output: annotated alternating P-automaton $\mathfrak{A}^* = (Q, \Sigma, P, \Delta^*, F, \mathcal{M})$

1 $\mathfrak{A}^0 := \mathfrak{A}$; $i := 0$

2 **while** *automaton can be updated* **do**

3 **For** $p \in P_0, pa \xrightarrow{m} qw \in \Delta_{\mathcal{P}}$ **do**

4 **Find** *Runtree* $\rho = (\rho_Q, \rho_\Delta, \mathfrak{T})$ *of* $\mathfrak{A}^i$ *on* qw

5 $t := (p, a, f)$ with

$$f := q \mapsto \begin{cases} \max\{m \circ m_\rho \mid m_\rho \in \mathsf{val}^q(\rho)\} & \text{if } q \in \rho_Q(\mathsf{Leafs}_{\mathfrak{T}}) \\ \bot & \text{otherwise} \end{cases}$$

6 **if** $\exists t' \in \Delta^i$ *with* $t' > t$ **then**

7 $\Delta^{i+1} := \Delta^i \setminus \{t' \in \Delta^i \mid t' > t\} \cup \{t\}$; $i := i + 1$

8 **else if** $\neg(\exists t' \in \Delta^i$ *with* $t' \leq t)$ **then**

9 $\Delta^{i+1} := \Delta^i \cup \{t\}$; $i := i + 1$

10 **For** $p \in P_1$ *and* $pa \xrightarrow{m_1} q_1 w_1 \in \Delta_{\mathcal{P}}, \ldots, pa \xrightarrow{m_n} q_n w_n \in \Delta_{\mathcal{P}}$ *all* a*-pushdown rules starting in* p **do**

11 **Find** *Runtrees* $\rho^j = (\rho^j_Q, \rho^j_\Delta, \mathfrak{T}^j)$ *of* $\mathfrak{A}^i$ *on* $q_j w_j$ *for* $j \in \{1, \ldots, n\}$

12 $t := (p, a, f)$ with f defined by

$$q \mapsto \begin{cases} \max\{m_j \circ m_\rho \mid m_\rho \in \mathsf{val}^q(\rho^j), \; j \in \{1, \ldots, n\}\} & \text{if } q \in \bigcup_{j=1}^{n} \rho^j_Q(\mathsf{Leafs}_{\mathfrak{T}^j}) \\ \bot & \text{otherwise} \end{cases}$$

if $\exists t' \in \Delta^i$ *with* $t' > t$ **then**

13 $\Delta^{i+1} := \Delta^i \setminus \{t' \in \Delta^i \mid t' > t\} \cup \{t\}$; $i := i + 1$

14 **else if** $\neg(\exists t' \in \Delta^i$ *with* $t' \leq t)$ **then**

15 $\Delta^{i+1} := \Delta^i \cup \{t\}$; $i := i + 1$

Result: $\mathfrak{A}^* := \mathfrak{A}^i$

$\{p\} \leq B$. Accordingly, we write $W_0^{(B)}(F)$ to denote the region in which player 0 wins with profile bound B.

Lemma 9. *Algorithm 1 terminates for all inputs.*

Proof. We remark that the set of all transitions always forms an anti-chain by construction. If the algorithm does not terminate either ll. 7, 13 or ll. 9,15 are executed infinitely many times. Assume the instructions in ll. 7 or 13 are executed infinitely many times. Since there are only finitely many transitions in Δ^i at each point in time, there has to be a descending chain of transitions. This is a contradiction to the fact that the order is a well-partial order. Otherwise,

ll. 9 or 15 are executed infinitely often. Since both update rules only add a new transition relation, we obtain $\Delta^i \subsetneq \Delta^{i+1}$ for all $i > j$ for some threshold j. Thus, the set $\bigcup_{i=j}^{\infty} \Delta^i$ is an infinite anti-chain. Also a contradiction. □

In order to prove the correctness of the saturation procedure, we show a direct correspondence between the games winning with certain profiles and runs on the saturated automaton.

Lemma 10. *Let F be a regular set of configurations represented by a normalized annotated alternating P-automaton $\mathfrak{A}$ as stated in the precondition of Algorithm 1. Furthermore, let $\mathfrak{A}^*$ be the result of the algorithm and $B \in \mathsf{AntiChain}(M)$ such that $\{e_{\mathcal{M}}\} \leq B$.*

(i) Let $qw \in W_0^{(B)}(F)$. Then, there is an accepting run ρ of $\mathfrak{A}^*$ on qw such that $\mathsf{val}(\rho) \leq B$.

(ii) Let $\rho = (\rho_Q, \rho_\Delta, \rho_M, \mathfrak{T})$ be a run of $\mathfrak{A}^*$ on qw with $S = \rho_Q(\mathsf{Leafs}_{\mathfrak{T}})$. Then player 0 has a strategy σ_ρ to reach a configuration in $L(\mathfrak{A}_S)$. Moreover, for a play τ that is played according to σ_ρ and that ends in a configuration $q'w'$, let $\rho^{\mathfrak{A}}$ be an accepting run of $\mathfrak{A}$ on $q'w'$ with (single) final state r. The value of τ is bounded by $\mathsf{val}^r(\rho)$.

The above presented procedure effectively reduces the bounded winning problem to a boundedness problem for alternating B-automata. After exchanging the counter profiles back to direct counter operations, we obtain an automaton that recognizes a configuration pw with some cost k iff player 0 wins the resource reachability game from this configuration with bound k. Since the set of initial configurations A is regular, we can easily construct an automaton that recognizes the complement $\overline{A}$ with cost 0 and A with cost ∞. By the closure of B-automata under taking the minimum (cf. [10]) we obtain an automaton that yields value 0 for all elements outside of A. As a consequence, the boundedness of this automaton only depends on the boundedness on A. By [10], we know that boundedness for these automata is decidable. Altogether, we obtain our main result:

Theorem 11. *Let $\mathcal{P} = (P_0 \uplus P_1, \Sigma, \Delta, \Gamma)$ be a resource pushdown system. Let F be a regular goal set for the bounded reachability game on the configuration graph of $\mathcal{P}$ and A a regular set of start configurations. It is decidable whether there is a $k \in \mathbb{N}$ such that $A \subseteq W_0^{(k)}(F)$.*

We remark that the winning strategies constructed in the inductive proof cannot be directly transformed into pushdown strategies as in the traditional case. This difference arises from the fact that we do not have memoryless runs on our alternating automaton model because of incomparable annotations. However, once the bound k is determined, one can use the methods of the previous section to obtain a finite memory winning strategy.

5 Conclusion

We considered two-player games on pushdown graphs with additional non-negative integer counters as model for reactive, recursive systems with resource consumption. On these games, we examined a combined reachability and resource

limit winning condition. We showed that these games are determined and that for regular goal sets one can compute the winning region of player 0 with a certain resource limit as well as decide whether there is a resource limit such that player 0 wins from a given regular set of initial configurations with this limit.

In the main theorem, we solved the bounded winning problem by an extension of the traditional saturation idea. Our approach can be used for a variety of annotated pushdown games. We only require that the annotations form a well-partially ordered monoid. In the semantics, the value of a play has to be associated with the concatenation of all the values on the transitions along the play and the order has to reflect that smaller annotations are preferred. For the specific case of counter profiles as annotation, we obtain a reduction of the bounded winning problem to the boundedness problem of alternating B-automata.

References

1. Abdulla, P.A., Krcal, P., Yi, W.: R-automata. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 67–81. Springer, Heidelberg (2008)
2. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
3. Brázdil, T., Chatterjee, K., Kučera, A., Novotný, P.: Efficient controller synthesis for consumption games with multiple resource types. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 23–38. Springer, Heidelberg (2012)
4. Cachat, T.: Symbolic strategy synthesis for games on pushdown graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 704–715. Springer, Heidelberg (2002)
5. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003. LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)
6. Chatterjee, K., Doyen, L.: Energy parity games. Theoretical Computer Science 458, 49–60 (2012)
7. Chatterjee, K., Fijalkow, N.: Infinite-state games with finitary conditions. In: CSL, pp. 181–196 (2013)
8. Church, A.: Applications of recursive arithmetic to the problem of circuit synthesis. Summaries of the Summer Institute of Symbolic Logic 1, 3–50 (1957)
9. Colcombet, T.: Regular cost functions over words (2009)
10. Colcombet, T., Löding, C.: Regular cost functions over finite trees. In: LICS, pp. 70–79 (2010)
11. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
12. Higman, G.: Ordering by Divisibility in Abstract Algebras. Proceedings London Mathematical Society s3-2(1), 326–336 (1952)
13. Kirsten, D.: Distance desert automata and the star height problem. RAIRO - Theoretical Informatics and Applications 39, 455–509 (2005)
14. Suwimonteerabuth, D., Schwoon, S., Esparza, J.: jMoped: A java bytecode checker based on moped. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 541–545. Springer, Heidelberg (2005)
15. Walukiewicz, I.: Pushdown processes: Games and model-checking. Inf. Comput. 164(2), 234–263 (2001)