VPL INTERSECTION EMPTINESS

BACHELOR THESIS

JAN LEIKE

20. JULI 2010

Gutachter:
Prof. Dr. Andreas Podelski     Prof. Dr. Peter Thiemann
Abstract

The emptiness of the intersection of visibly pushdown languages [1] is decidable. However, if different alphabet partitions, or more precisely pushdown alphabets, are considered, the emptiness of this intersection becomes undecidable. It will be shown that this problem can be split into four cases. One of them has already been studied, another one is trivial. Of the last two cases one will be proven undecidable in this paper, the other one remains open.

1 Introduction

Pushdown languages play a significant role in program analysis, model checking and verification. Program control flow with invocation of nested and potentially recursive method calls is naturally modeled by these languages. Because of this, many questions in the fields mentioned above can be formulated as decision problems for pushdown automata [3, 4, 5, 6]. However, pushdown languages are not closed under some important operations, such as complementation or intersection, and the inclusion problem is undecidable. As a result of this, the restricted class of visibly pushdown languages has been proposed.

The corresponding visibly pushdown automata are similar to pushdown automata for context-free languages, with the difference that the stack operations are determined by the symbols of the alphabet. In other words, the automaton’s alphabet is partitioned into three parts: a set of call symbols, each of which requires the automaton to push on its stack, a set of return symbols, each of which requires the automaton to pop on its stack, and a set of local symbols, each of which prohibits stack interaction. This restriction results in the desired closure properties, such as being closed under union, intersection, complementation, renaming, concatenation, and Kleene-*. The deterministic and non-deterministic versions of visibly pushdown automata are equivalent and determinization can be performed in exponential time. Also, the inclusion problem becomes decidable for visibly pushdown automata [1].

For instance, one could use these results to verify if a program $P$ fits a specification $\varphi$, by abstracting the program and specification to visibly pushdown automata $M_P$ and $M_\varphi$, and checking that the intersection of $M_P$ and the complement of $M_\varphi$ is empty. As shown by Alur and Madhusudan, these operations are decidable, but Exptime-complete.

However, this is only true if the automata have the same alphabet partitions. The question of how the properties of the intersection hold up arises, if one considers two visibly pushdown languages with different alphabet partitions, or pushdown alphabets. The goal of this paper is to analyze which cases exist and to discuss their respective decidabilities.

It will be shown that this problem can be split into four cases, one of which, with equal partitions, has already been proven by Alur and Madhusudan. Another one implies the restriction to a regular language for at least one of the two visibly pushdown languages, where the emptiness of the intersection is decidable, which is not difficult to show. The largest part of this paper is about the third case, in which the intersection is undecidable. To show this, a specific alphabet and a specific partition thereof are picked, over which two visibly pushdown automata simulate a Turing Machine on an empty tape. Then this construction is generalized by projecting it on two visibly pushdown automata on an arbitrary alphabet partition satisfying the precondition of this case.
2 Preliminaries

These are the definitions of pushdown alphabets, visibly pushdown automata with their runs as well as visibly pushdown languages as given by Rajeev Alur and P. Madhusudan [1]. For the definition of Turing Machines, their configurations and their runs the common notation is used [2].

Definition 1 (Pushdown Alphabet). A pushdown alphabet is a tuple $\tilde{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_l \rangle$ that comprises three disjoint finite alphabets - $\Sigma_c$ is a finite set of calls, $\Sigma_r$ is a finite set of returns and $\Sigma_l$ is a finite set of local actions. For any such $\tilde{\Sigma}$, let $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$.

Pushdown automata are defined over $\tilde{\Sigma}$. Intuitively, the pushdown automaton is restricted such that it pushes onto the stack only when it reads a call, it pops the stack only at returns, and does not use the stack when it reads local actions. The input hence controls the kind of operations permissible on the stack—however, there is no restriction on the symbols that can be pushed or popped.

Definition 2 (Visibly Pushdown Automaton). A (nondeterministic) visibly pushdown automaton (VPA) over $\tilde{\Sigma}$ on finite words over $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$ is a tuple $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ where $Q$ is a finite set of states, $Q_{in} \subseteq Q$ is a set of initial states, $\Gamma$ is a finite stack alphabet that contains a special bottom-of-stack symbol ⊥, $\delta \subseteq (Q \times \Sigma_c \times \Sigma_r \times \Sigma_l \times \Gamma) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_l \times Q)$ and $Q_F \subseteq Q$ is a set of final states.

A transition $(q, a, q', \gamma)$, where $a \in \Sigma_c$ and $\gamma \neq \bot$, is a push-transition where on reading $a$, $\gamma$ is pushed onto the stack and the control changes from state $q$ to $q'$. Similarly, $(q, a, \gamma, q')$ is a pop-transition where $\gamma$ is read from the top of the stack and popped (if the top of stack is ⊥, then it is read but not popped), and the control state changes from $q$ to $q'$. Note that on local actions, there is no stack operation.

A VPA $(Q, Q_{in}, \Gamma, \delta, Q_F)$ is said to be deterministic if $|Q_{in}| = 1$ and for every $q \in Q$: 

- for every $a \in \Sigma_c$, there is at most one transition of the form $(q, a, q', \gamma) \in \delta$,
- for every $a \in \Sigma_r$, there is at most one transition of the form $(q, a, q', \gamma) \in \delta$, and
- for every $a \in \Sigma_l$, $\gamma \in \Gamma$, there is at most one transition of the form $(q, a, q', \gamma) \in \delta$.

A stack is a nonempty finite sequence over $\Gamma$ ending in the bottom-of-stack symbol ⊥; let us denote the set of all stacks as $St = (\Gamma \setminus \{\bot\})^*$ ⊥.

Definition 3 (Run of a VPA). For a word $w = a_1a_2...a_k$ in $\Sigma^*$, a run of $M$ on $w$ is a sequence $\rho = (q_0, \sigma_0), ..., (q_k, \sigma_k)$, where each $q_i \in Q$, $\sigma_i \in St$, $q_0 \in Q_{in}$, $\sigma_0 = \bot$ and for every $1 \leq i \leq k$ the following holds:

- Push: If $a_i$ is a call, then for some $\gamma \in \Gamma$, $(q_i, a_i, q_{i+1}, \gamma) \in \delta$ and $\sigma_{i+1} = \gamma \sigma_i$.
- Pop: If $a_i$ is a return, then for some $\gamma \in \Gamma$, $(q_i, a_i, \gamma, q_{i+1}) \in \delta$ and either $\gamma \neq \bot$ and $\sigma_i = \gamma \sigma_{i+1}$, or $\gamma = \bot$ and $\sigma_i = \sigma_{i+1} = \bot$.
- Local: If $a_i$ is a local action, then $(q_i, a_i, q_{i+1}) \in \delta$ and $\sigma_{i+1} = \sigma_i$.

A run $\rho = (q_0, \sigma_0), ..., (q_k, \sigma_k)$ is accepting iff the last state is a final state, that is, if $q_k \in Q_F$. A word $w \in \Sigma^*$ is accepted by a VPA $M$ if there is an accepting run of $M$ on $w$. The language of $M$, $L(M)$, is the set of words accepted by $M$.

Definition 4 (Visibly Pushdown Language). A language of finite words $L \subseteq \Sigma^*$ is a visibly pushdown language (VPL) with respect to $\tilde{\Sigma}$ (a $\tilde{\Sigma}$ - VPL) if there is a visibly pushdown automaton $M$ over $\tilde{\Sigma}$ such that $L(M) = L$. 

2
Example 5. For a better understanding of visibly pushdown languages, a VPA that accepts fully bracketed formulas from propositional logic will be discussed.

Let $\Sigma = \{\{, \}, \{, \}\}$ be a pushdown alphabet. It follows that the corresponding alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_t = \{[ , ] , \neg , \land , \lor , a, b, c\}$. The pushdown automaton $M = (Q,Q_{\text{in}}, \Gamma, \delta, Q_F)$ is defined by

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_F\}$$

$$Q_{\text{in}} = \{q_0\}$$

$$Q_F = \{q_F\}$$

$$\Gamma = \{0,1,2\}$$

$$\delta = \{(q_0, \neg, q_0), (q_1, \neg, q_1), (q_1, a, q_F), (q_1, b, q_F), (q_1, c, q_F), (q_2, \land, q_3), (q_2, \lor, q_3), (q_3, \neg, q_3), (q_3, a, q_4), (q_3, b, q_4), (q_3, c, q_4), (q_4, a, q_4), (q_4, b, q_4), (q_4, c, q_4), (q_4, \bot), 0, q_5), (q_5, [ , q_1, 0), (q_1, [ , q_1, 1), (q_4, [ , 1, q_2), (q_4, ] , 2, q_4)\}

This state diagram visualizes $M$:

Transitions labeled with multiple symbols separated with a "\;\;" represent multiple transitions that are displayed as one for clarity reasons. Note that the automaton $M$ is deterministic.

The language of words over $\Sigma$ that are accepted by $M$, $L(M)$, is the set of valid, fully bracketed formulae of propositional logic with the variables $a$, $b$ and $c$. Every time a bracket is opened a symbol is pushed on the stack. This symbol is 0 for the first bracket and then 1 or 2, depending on whether the bracket is opened in the first or in the second part of a con- or disjunction. Later, when this bracket is closed, the automaton pops a 0, 1 or 2 and continues it the according state.

To give an example the formula (word) $w$ below is in $L(M)$.

$$\neg[[\neg b \land [c \lor \neg a]] \land \neg c]$$

A run $\rho$ of $M$ on $w$ would be

$$(q_0, \bot) (q_0, \bot) (q_1, 0\bot) (q_1, 10\bot) (q_1, 10\bot) (q_2, 10\bot) (q_2, 10\bot) (q_3, 210\bot)$$

$$(q_2, 210\bot) (q_3, 210\bot) (q_3, 210\bot) (q_4, 210\bot) (q_4, 10\bot) (q_2, 0\bot) (q_3, 0\bot)$$

$$(q_3, 0\bot) (q_4, 0\bot) (q_F, \bot)$$
3 Emptiness of the Intersection

In order to formulate the four different cases for the emptiness of the intersection of the two VPAs precisely, definitions to compare the two pushdown alphabets are needed.

Definition 6 (Stack alteration function). Let \( \tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l) \) be a pushdown alphabet and \( \Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l \). The function \( \xi : \Sigma \to \mathbb{Z} \) is called the stack alteration function for \( \tilde{\Sigma} \), if \( \xi \) is defined as follows.

- For every \( a \in \Sigma_c \), \( \xi(a) = +1 \),
- for every \( a \in \Sigma_r \), \( \xi(a) = -1 \) and
- for every \( a \in \Sigma_l \), \( \xi(a) = 0 \).

\( \xi \) is extended to words in the natural way, that is for every word \( w = a_1 \ldots a_n \in \Sigma^* \), where every \( a_i \in \Sigma \), \( \xi(w) \) is defined as \( \sum_{i=1}^n \xi(a_i) \) and is called the stack alteration of \( w \).

Intuitively, the stack alteration function "measures" the number of symbols by which the stack of an automaton over the respective pushdown alphabet changes.

Example 7. Let \( \tilde{\Sigma}_1 = \{\{c, s\}, \{r\}, \{t\}\} \) be a pushdown alphabet. By its definition, \( \Sigma_{c1} = \{c, s\} \) is the set of call symbols, \( \Sigma_{r1} = \{r\} \) is the set of return symbols and \( \Sigma_{l1} = \{t\} \) is the set of local actions. Let \( \xi_1 \) be the stack alteration function for \( \tilde{\Sigma}_1 \). Then the following equations hold.

\[
\begin{align*}
\xi_1(c) &= 1 \quad \xi_1(s) = 1 \quad \xi_1(r) = -1 \quad \xi_1(t) = 0 \\
\xi_1(scs) &= 3 \quad \xi_1(crc) = 1 \quad \xi_1(rct) = 1 \\
\xi_1(\varepsilon) &= \sum_{\emptyset} = 0
\end{align*}
\]

This paper is about the intersection of two VPAs, therefore symbols occurring in only one of the languages’ alphabet cannot occur in the intersection of the automata and hence do not need to be considered here. In other words, in the following, if \( \tilde{\Sigma}_1 = (\Sigma_{c1}, \Sigma_{r1}, \Sigma_{l1}) \) and \( \tilde{\Sigma}_2 = (\Sigma_{c2}, \Sigma_{r2}, \Sigma_{l2}) \) are two pushdown alphabets, let \( \Sigma_1 = \Sigma_{c1} \cup \Sigma_{r1} \cup \Sigma_{l1} = \Sigma_{c2} \cup \Sigma_{r2} \cup \Sigma_{l2} = \Sigma_2 \).

Definition 8 (Enlarging symbols, neutral symbols). Let \( \tilde{\Sigma}_1 \) and \( \tilde{\Sigma}_2 \) be two pushdown alphabets and \( \xi_1 \) and \( \xi_2 \) their respective stack alteration functions. A symbol \( a \in \Sigma \) is called enlarging for \( \tilde{\Sigma}_1 \), if \( \xi_1(a) > \xi_2(a) \) and \( a \) is called enlarging for \( \tilde{\Sigma}_2 \), if \( \xi_2(a) > \xi_1(a) \). If \( \xi_1(a) = \xi_2(a) \) then \( a \) is called neutral.

Hence a symbol \( a \in \Sigma \) is enlarging for \( \tilde{\Sigma}_1 \), iff \( a \in \Sigma_{c1} \cap \Sigma_{c2} \), \( a \in \Sigma_{l1} \cap \Sigma_{l2} \) or \( a \in \Sigma_{c1} \cap \Sigma_{r2} \). \( a \) is enlarging for \( \tilde{\Sigma}_2 \), iff \( a \in \Sigma_{c1} \cap \Sigma_{c2} \), \( a \in \Sigma_{l1} \cap \Sigma_{l2} \) or \( a \in \Sigma_{r1} \cap \Sigma_{r2} \) and \( a \) is called neutral, iff \( a \in \Sigma_{c1} \cap \Sigma_{l2} \), \( a \in \Sigma_{r1} \cap \Sigma_{r2} \) or \( a \in \Sigma_{c1} \cap \Sigma_{l1} \).

Intuitively, transitions labeled with a symbol that is enlarging for the pushdown alphabet \( \tilde{\Sigma}_1 \) enable the stack of the first automaton to grow larger in relation to the second automaton’s stack. Neutral symbols do the same stack operations on both automata and therefore cannot be used to access the stacks independently.

Example 9. Let \( \Sigma = \{c, r, s, t\} \) be an alphabet and the pushdown alphabet \( \tilde{\Sigma}_1 = \{\{c, s\}, \{r\}, \{t\}\} \) like in Example 7. If the pushdown alphabet \( \tilde{\Sigma}_2 = \{\{c\}, \{r, t\}, \{s\}\} \), then there are two symbols \( s \) and \( t \) which are enlarging for \( \tilde{\Sigma}_1 \) and no symbols which are enlarging for \( \tilde{\Sigma}_2 \). The symbols \( c \) and \( r \) are neutral.

Proposition 10. Let \( M_1 \) and \( M_2 \) be two visibly pushdown automata over the pushdown alphabets \( \tilde{\Sigma}_1 = (\Sigma_{c1}, \Sigma_{r1}, \Sigma_{l1}) \) and \( \tilde{\Sigma}_2 = (\Sigma_{c2}, \Sigma_{r2}, \Sigma_{l2}) \) with \( \Sigma = \Sigma_1 = \Sigma_2 \). The following table illustrates the four cases for the emptiness of the intersection of \( M_1 \) and \( M_2 \), i.e. the question whether \( L(M_1) \cap L(M_2) = \emptyset \). To prove the respective decidability of these cases will be the remainder of this paper.
Otherwise decidable
(Case I-a)

<table>
<thead>
<tr>
<th>Case Description</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every symbol in $\Sigma$ is a neutral symbol</td>
<td>$\Sigma$ contains either symbols enlarging for $\bar{\Sigma}_1$ or symbols enlarging for $\bar{\Sigma}_2$ as well as any number of neutral symbols</td>
</tr>
<tr>
<td>Neither one of $\Sigma_{c1}$, $\Sigma_{c2}$, $\Sigma_{r1}$ or $\Sigma_{r2}$ is empty</td>
<td>$\Sigma$ contains either symbols enlarging for $\bar{\Sigma}_1$ or symbols enlarging for $\bar{\Sigma}_2$ as well as any number of neutral symbols</td>
</tr>
</tbody>
</table>

Because $\Sigma$ must contain either no enlarging symbols, enlarging symbols for one pushdown alphabet or enlarging symbols for both pushdown alphabets, the listed cases are exhaustive.

The two easiest cases will be dealt with first. In Case I-a, where every symbol is a neutral symbol, it follows that the pushdown alphabets must be equal. This has already been considered by Rajeev Alur and P. Madhusudan in the introduction of visibly pushdown languages and therefore their proof is referenced here.

**Theorem 11** (Case I-a). If the pushdown alphabets are the same, $\bar{\Sigma}_1 = \bar{\Sigma}_2$, then the emptiness of the intersection is decidable. [1]

Case II, in which at least one of the call- or return alphabets is empty, works analogously to the proof that context-free languages are closed under intersection with regular languages [2]. This proof is just sketched here.

**Theorem 12** (Case II). For all pushdown alphabets where $\Sigma_{c1} = \emptyset$, $\Sigma_{r1} = \emptyset$, $\Sigma_{c2} = \emptyset$ or $\Sigma_{r2} = \emptyset$ the emptiness of the intersection is decidable.

**Proof.** A $Vpl$ having no call oder return symbols is a regular language. By constructing the product automaton of the $Vpl$ and the regular language, it can be shown, that visibly pushdown languages are closed under intersection with regular languages. The emptiness of $Vpl$s is decidable [1], which concludes this proof. □

Now turn to Case I-c, in which there are both, symbols enlarging for the first and the second pushdown alphabet. The analysis of this case will be the remainder of this paper. At first two specific pushdown alphabets $\bar{\Sigma}_1$ and $\bar{\Sigma}_2$ are considered and this is later generalized to arbitrary ones. The undecidability of this case will be shown by simulation of a given Turing Machine on the empty tape which accepts if and only if the intersection of two visibly pushdown automata is not empty. In order to do this, a pair of $Vpas$, called simulation of $\tau$, must be constructed from the Turing Machine.

For a deterministic Turing Machine $\tau$ the following notation is used [2]: $\tau = (Q, \Gamma \setminus \{\|\}, \Gamma, \delta, q_0, \|, F)$ where $Q$ is a finite set of states, $\Gamma = \{0, 1, \|\}$ is the tape alphabet, $\|$ is the blank symbol, $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the partial transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of accepting states.

**Definition 13** (Simulation of a Turing Machine). Let $\Sigma^{sim} = \{b, d, x, y, z\}$ be a specific, fixed alphabet with five pairwise different symbols. Furthermore let $\tau = (Q, \Gamma \setminus \{\|\}, \Gamma, \delta, q_0, \|, F)$ be a deterministic Turing Machine.

The pair $(M^\tau_1, M^\tau_2)$ is called simulation of $\tau$, if $M^\tau_1 = (Q', q_0', \Gamma', \delta_1, F')$ and $M^\tau_2 = (Q', q_0', \Gamma', \delta_2, F')$ are two deterministic $Vpas$ over the pushdown alphabets $\bar{\Sigma}^{sim}_1 = \{\{b\}, \{d\}, \{x, y, z\}\}$ and $\bar{\Sigma}^{sim}_2 = \{\{d\}, \{b\}, \{x, y, z\}\}$ respectively and $Q' = Q \times \Gamma \times \{\#, 0, 1, 2, 3\}$, $q_0' = \langle q_0, \|, \# \rangle$, $\Gamma' = \Gamma \cup \{\|\}$, $F' = \{\langle q, \gamma, \# \rangle \mid q \in F, \gamma \in \Gamma\}$ and the transition functions $\delta_1$ and $\delta_2$ are defined as follows.
The run of tape and returns the head to the starting position.

Example of a very simple Turing Machine. Hence consider $M$ of state, the tape on the right side is stored in the stack of $M$ because the simulation of Turing Machines bloats in size significantly, there will be only an example. 

Intuitively, the symbol currently under the read/write head of the Turing Machine $\tau$ is stored in the control state, the tape on the right side is stored in the stack of $M_2^\tau$ and the tape on the left side is stored in the stack of $M_1^\tau$. The following lemma will formalize this statement, but first there is an example.

**Example 14.** Because the simulation of Turing Machines bloats in size significantly, there will be only an example of a very simple Turing Machine. Hence consider $\tau = (Q, \Gamma \setminus \{\sqcup\}, \Gamma, \delta, q_0, \sqcup, F)$ which writes 10 on the tape and returns the head to the starting position. $\tau$ is formally defined below.

$$\begin{align*}
\delta_1 &= \{(q, \gamma, \#), d, 0, (q, \gamma, 0)\}, \\
&\quad \{(q, \gamma, 0), x, (q', 0, \#)\}, \\
&\quad \{(q, \gamma, \#), d, 1, (q, \gamma, 1)\}, \\
&\quad \{(q, \gamma, 1), y, (q', 1, \#)\}, \\
&\quad \{(q, \gamma, \#), d, \sqcup, (q, \gamma, 2)\}, \\
&\quad \{(q, \gamma, \#), d, \perp, (q, \gamma, 2)\}, \\
&\quad \{(q, \gamma, 2), z, (q', \sqcup, \#)\} \mid \delta(q, \gamma) = (q', \gamma, L) \\
\cup \\
&\quad \{(q, \gamma, \#), b, \gamma', (q, \gamma, 3)\}, \\
&\quad \{(q, \gamma, 3), x, (q', 0, \#)\}, \\
&\quad \{(q, \gamma, 3), y, (q', 1, \#)\}, \\
&\quad \{(q, \gamma, 3), z, (q', \sqcup, \#)\} \mid \delta(q, \gamma) = (q', \gamma', R) \\
\delta_2 &= \{(q, \gamma, \#), b, 0, (q, \gamma, 0)\}, \\
&\quad \{(q, \gamma, 0), x, (q', 0, \#)\}, \\
&\quad \{(q, \gamma, \#), b, 1, (q, \gamma, 1)\}, \\
&\quad \{(q, \gamma, 1), y, (q', 1, \#)\}, \\
&\quad \{(q, \gamma, \#), b, \sqcup, (q, \gamma, 2)\}, \\
&\quad \{(q, \gamma, \#), b, \perp, (q, \gamma, 2)\}, \\
&\quad \{(q, \gamma, 2), z, (q', \sqcup, \#)\} \mid \delta(q, \gamma) = (q', \gamma', R) \\
\cup \\
&\quad \{(q, \gamma, \#), d, \gamma', (q, \gamma, 3)\}, \\
&\quad \{(q, \gamma, 3), x, (q', 0, \#)\}, \\
&\quad \{(q, \gamma, 3), y, (q', 1, \#)\}, \\
&\quad \{(q, \gamma, 3), z, (q', \sqcup, \#)\} \mid \delta(q, \gamma) = (q', \gamma', L) \\
\end{align*}$$

The run of $\tau$ on the empty word is 

$$\rho = (q_0, \sqcup) (1, q_1, \sqcup) (q_2, 10).$$

The simulation of $\tau$, $(M_1^\tau, M_2^\tau)$ is illustrated by these two diagrams for $M_1^\tau$ and $M_2^\tau$. 

---

**Warning:** The above text contains mathematical notation and diagrams that are not well-rendered in the text format provided. For a more accurate representation, please refer to the original PDF or source material. The symbols and diagrams are crucial for understanding the context and may require specific knowledge or software to interpret correctly. If you need further assistance or clarification, please let me know.
The idea of the simulation is that with the symbols b or d one automaton pops a stack (tape) symbol and then the symbols x, y and z are used to "transfer" the symbol to the other automaton.

The intersection of $M^+_1$ and $M^+_2$, 

$$L(M^+_1) \cap L(M^+_2) = \{bdy\}$$

is not empty (in Theorem 16 will be shown that the fact that $\tau$ accepts the empty input is a sufficient and necessary condition for this). The runs of $M^+_1$ and $M^+_2$ on the word b0y are given by

$$\rho_1 = ((q_0, \bot, \#), \bot) ((q_0, \bot, 3), \bot) ((q_1, \bot, \#), \bot) ((q_1, \bot, 1), \bot) ((q_2, \bot, \#), \bot)$$

and

$$\rho_2 = ((q_0, \bot, \#), \bot) ((q_0, \bot, 2), \bot) ((q_1, \bot, \#), \bot) ((q_1, \bot, 3), 0\bot) ((q_2, \bot, \#), 0\bot).$$

**Lemma 15.** Let $\tau$ be a deterministic Turing Machine, the pair $(M^+_1, M^+_2)$ of VPAs a simulation of $\tau$ and let $\sigma_1, \sigma_2 \in \Gamma^*$ be two sequences over the stack alphabet, $q \in Q$ a control state, $\gamma \in \Gamma$ a stack symbol and $n \in \mathbb{N}$. In the run $\rho$ of $\tau$ of length $|\rho| = n$, the last configuration is $\sigma_1 \gamma \sigma_2$ if and only if, for every word $w \in L(M^+_1) \cap L(M^+_2)$ with $|w| \geq 2n$

- in the run $\rho_1$ of $M^+_1$ on $w$ of length $|\rho_1| = 2n$ the last configuration of $\rho_1$ is $(q, \gamma, \#), \sigma_1^R \bot)$ and
- in the run $\rho_2$ of $M^+_2$ on $w$ of length $|\rho_2| = 2n$ the last configuration of $\rho_2$ is $(q, \gamma, \#), \sigma_2 \bot)$. 

7
where $R$ is the reverse-operator. Also, for every run $\rho'$ the last control state of $M_T^1$ and $M_T^2$ is $\langle \cdot, \cdot, \# \rangle$ respectively if and only if $|\rho'|$ is even.

Note that because $\tau$ is deterministic, there is exactly one run of a given length on the empty input. The fact that there is only one run of any given length for the intersection of $M_T^1$ and $M_T^2$ is a result of this lemma. This explains the formulation "the run of length ..." as opposed to "for every run of length ...", which would have the same meaning here.

Proof by Induction over $n$. For $n = |\rho| = 1$ the Turing Machine $\tau$ is in the configuration $q_{0}\uparrow \bot$ just as $M_T^1$ and $M_T^2$ are in the configuration $\langle (q, \uparrow, \#), \bot \rangle$ respectively.

Induction step. $M_T^1$, $M_T^2$ and $\tau$ are deterministic, therefore only one way of the bi-implication has to be shown. Let $\rho$ be a run of $\tau$ of length $|\rho| = n$ and assume $\tau$’s transition function will move the read/write head on the tape to the left in the successor configuration of $\rho$, the other case will be symmetric.

Let $\sigma_1$ $\gamma$ $\sigma_2$ be the last configuration of $\rho$ while the last configurations of $\rho_1$ and $\rho_2$ are $\langle (q, \gamma, \#), \sigma_1^{R}\bot \rangle$ and $\langle (q, \gamma, \#), \sigma_2 \bot \rangle$ respectively for a $q \in Q$ and a $\gamma \in \Gamma$. Let $|\rho_1| = |\rho_2| = 2|\rho| = 2n$ and let $\langle q', \gamma', L \rangle = \delta(q, \gamma)$, then the successor configuration of $\rho$ will be $\sigma_1 q' \gamma'' \gamma' \sigma_2$ for a $\gamma'' \in \Gamma$ where either $\sigma_1 = \sigma_1^{R}$ or $\sigma_1' = \epsilon$ and $\gamma'' = \uparrow \bot$, if $\sigma_1 = \epsilon$.

Let $\rho_1' = \rho_1 \eta_1 \eta_2$ be the continued run of $\rho_1$ and let $\rho_2' = \rho_2 \eta_1 \eta_2$ be the continued run of $\rho_2$. Consider four possibilities for $\gamma''$:

- $\gamma'' = 0 \Rightarrow \eta_1_1 = \langle (q, \gamma, 0), \sigma_1^{R}\bot \rangle$ and $\eta_2_2 = \langle (q', 0, \#), \sigma_1^{R}\bot \rangle$ using the transitions $\langle (q, \gamma, 0), d, 0, (q, \gamma, 0) \rangle$ and then $\langle (q, \gamma, 0), x, (q', 0, \#) \rangle$. Therefore $\eta_1_1 = \langle (q, \gamma, 3), \gamma' \sigma_2 \bot \rangle$ and $\eta_2_2 = \langle (q', 0, \#), \gamma' \sigma_2 \bot \rangle$ using the transitions $\langle (q, \gamma, 3), d, \gamma', (q, \gamma, 3) \rangle$ and then $\langle (q, \gamma, 3), x, (q', 0, \#) \rangle$.

- $\gamma'' = 1 \Rightarrow \eta_1_1 = \langle (q, \gamma, 1), \sigma_1^{R}\bot \rangle$ and $\eta_2_2 = \langle (q', 1, \#), \sigma_1^{R}\bot \rangle$ using the transitions $\langle (q, \gamma, 1), d, 1, (q, \gamma, 1) \rangle$ and then $\langle (q, \gamma, 1), y, (q', 1, \#) \rangle$. Therefore $\eta_1_1 = \langle (q, \gamma, 3), \gamma' \sigma_2 \bot \rangle$ and $\eta_2_2 = \langle (q', 1, \#), \gamma' \sigma_2 \bot \rangle$ using the transitions $\langle (q, \gamma, 3), d, \gamma', (q, \gamma, 3) \rangle$ and then $\langle (q, \gamma, 3), y, (q', 1, \#) \rangle$.

- $\sigma_1 = \epsilon \Rightarrow \eta_1_1 = \langle (q, \gamma, 2), \bot \rangle$ and $\eta_2_2 = \langle (q', \uparrow, \#), \bot \rangle$ using the transitions $\langle (q, \gamma, 2), d, \uparrow, (q, \gamma, 2) \rangle$ and then $\langle (q, \gamma, 2), z, (q', \uparrow, \#) \rangle$. Therefore $\eta_1_1 = \langle (q, \gamma, 3), \gamma' \sigma_2 \bot \rangle$ and $\eta_2_2 = \langle (q', \uparrow, \#), \gamma' \sigma_2 \bot \rangle$ using the transitions $\langle (q, \gamma, 3), d, \gamma', (q, \gamma, 3) \rangle$ and then $\langle (q, \gamma, 3), z, (q', \uparrow, \#) \rangle$.

- $\gamma'' = \uparrow \bot (\sigma_1 \neq \epsilon) \Rightarrow \eta_1_1 = \langle (q, \gamma, 2), \sigma_1^{R}\bot \rangle$ and $\eta_2_2 = \langle (q', \uparrow, \#), \sigma_1^{R}\bot \rangle$ using the two transitions $\langle (q, \gamma, 2), d, \uparrow, (q, \gamma, 2) \rangle$ and then $\langle (q, \gamma, 2), z, (q', \uparrow, \#) \rangle$. Therefore $\eta_1_1 = \langle (q, \gamma, 3), \gamma' \sigma_2 \bot \rangle$ and $\eta_2_2 = \langle (q', \uparrow, \#), \gamma' \sigma_2 \bot \rangle$ using the transitions $\langle (q, \gamma, 3), d, \gamma', (q, \gamma, 3) \rangle$ and then $\langle (q, \gamma, 3), z, (q', \uparrow, \#) \rangle$.

In all four cases $\eta_1$ and $\eta_2$ contain a control state $\langle \cdot, \cdot, \# \rangle$ while $\eta_1_1$ and $\eta_2_2$ do not contain a control state of this kind.

Using this lemma it will now be shown that the simulation indeed does what its name suggests and the VPA’s intersection is empty if and only if the Turing Machine accepts the empty tape.

Theorem 16 $(M_T^1$ and $M_T^2$ simulate $\tau$ on $\epsilon$). Let $\tau$ be a deterministic Turing Machine and $(M_T^1, M_T^2)$ be simulation of $\tau$. Then $L(M_T^1) \cap L(M_T^2)$ is not empty if and only if $\tau$ accepts the empty word.

Proof.

$\Rightarrow$ Let $w \in L(M_T^1) \cap L(M_T^2)$. Therefore there is an accepting run $\rho_1$ of $M_T^1$ on $w$ and an accepting run $\rho_2$ of $M_T^2$ on $w$ with $|\rho_1| = |\rho_2|$. According to the construction of $M_T^1$ and $M_T^2$, the last state of $\rho_1$ is $\langle q_1, \gamma_1, \# \rangle$ and the last state of $\rho_2$ is $\langle q_2, \gamma_2, \# \rangle$ for some $q_1, q_2 \in F$ and $\gamma_1, \gamma_2 \in \Gamma$.

By Lemma 15 it follows that $|\rho_1|$ has to be even and that there is a run $\rho$ of $\tau$ with $|\rho_1| = 2|\rho|$ and the last state of $\rho$ will be $q_1 = q_2 \in F$, so $\rho$ is an accepting run of $\tau$ on the empty word.
Let \( \rho \) be an accepting run of \( \tau \) on the empty word and let \( q_f \in F \) be the last state in \( \rho \). According to Lemma 15, there is a run \( \rho_1 \) of \( M_1 \) and a run \( \rho_2 \) of \( M_2 \) with \( |\rho_1| = |\rho_2| = 2|\rho| \) where in the last configuration they both are in the state \( (q_f, \gamma, \#) \) respectively for a \( \gamma \in \Gamma \). From \( F' = \{(q, \gamma, \#) | q \in F, \gamma \in \Gamma \} \) follows that both, \( \rho_1 \) and \( \rho_2 \) are accepting i.e. there exists a \( w \in L(M_1) \cap L(M_2) \).

\[ \square \]

**Corollary 17.** From the undecidability of the word problem for Turing Machines it follows that the emptiness of \( L(M_1) \cap L(M_2) \) is undecidable.

At this point it has been shown that there are two visibly pushdown automata over a specific alphabet \( \Sigma \), of which the emptiness of the intersection is undecidable. Now it remains to generalize this result to arbitrary pushdown alphabets that meet the conditions of Case I-c.

This will be done by tailoring the beforehand constructed automata to two arbitrary pushdown alphabets with the required properties. Technically this is another simulation, but to avoid confusion, it will be called a projection under a symbol substitution. The idea is to replace every transition in the simulation VPAs by a series of transitions which are constructed in accordance to the pushdown alphabets. To do this a function that maps one alphabet to a series of symbols (i.e. words) of the other alphabet must be defined.

**Definition 18 (Symbol substitution).** Let \( \tilde{\Sigma}_1 \) and \( \tilde{\Sigma}_2 \) be two pushdown alphabets with the stack alteration functions \( \xi_1 \) and \( \xi_2 \) respectively. An injective function \( \beta : \Sigma_1 \rightarrow \Sigma_2 \setminus \{\varepsilon\} \) is called symbol substitution from \( \tilde{\Sigma}_1 \) to \( \tilde{\Sigma}_2 \), if for every symbol \( a \in \Sigma_1 \) the equation \( \xi_1(a) = \xi_2(\beta(a)) \) holds. Let \( M \) be a visibly pushdown automaton over \( \tilde{\Sigma}_1 \), the symbol substitution \( \beta \) is then called valid for \( M \) if for every control state \( q \in Q \) of \( M \) there are no two outgoing transitions labeled with different symbols \( a, c \in \Sigma_1 \) such that \( \beta(a) \) is a prefix of \( \beta(c) \).

Note that the key difference to alphabet renaming [1] is that every symbol is mapped to multiple symbols, of which each may have a different stack alteration.

**Example 19.** Consider the visibly pushdown automaton \( M \) over the pushdown alphabet \( \tilde{\Sigma} \) of Example 5. Let \( \tilde{\Sigma}_N = \{[\{\},\{\}],\{0,1,\,\,\,-,\,\,\,\,+\,\,\,\,\,\,\,*\}\} \) be a pushdown alphabet. The function \( \beta : \Sigma \rightarrow \Sigma_N \) is defined by

\[
\begin{align*}
\beta(\{\}) &= [ \quad & \beta(\{\}) &= ] \\
\beta(\vee) &= + \quad & \beta(\land) &= * \\
\beta(\neg) &= - \quad & \beta(a) &= 0 \\
\beta(b) &= 1 \quad & \beta(c) &= -[1 + 1]
\end{align*}
\]

Obviously \( \beta \) is injective. Let \( \xi_M \) be the stack alteration function for \( \tilde{\Sigma} \) and \( \xi_N \) be the stack alteration function for \( \tilde{\Sigma}_N \). Every symbol except \( c \) is mapped to a single symbol, and the stack alteration is retained. Now consider \( \beta(c) \), where \( c \) is mapped to six symbols in \( \Sigma_N \).

\[
\xi_M(c) = 0 \\
\xi_N(\beta(c)) = \xi_N(-[1 + 1]) = 0 + 1 + 0 + 0 + 0 - 1 = 0
\]

Hence it has been shown that \( \beta \) is a symbol substitution from \( \tilde{\Sigma} \) to \( \tilde{\Sigma}_N \). However \( \beta \) is not valid for \( M \) because for example in the state \( q_3 \) there is an outgoing transition labeled with \( \neg \) and one labeled with \( c \). \( \beta(\neg) = - \) and is therefore a prefix of \( \beta(c) = -[1 + 1] \).

Using a symbol substitution it will now be defined what it means to change the transitions in a VPA using this function.

**Definition 20 (Projection under a symbol substitution).** Let \( M \) be a (non-deterministic) visibly pushdown automaton over the pushdown alphabet \( \tilde{\Sigma}_M \), let \( \tilde{\Sigma}_N \) be a pushdown alphabet and let \( \beta : \Sigma_M \rightarrow \Sigma_N \) be a symbol substitution from \( \tilde{\Sigma}_M \) to \( \tilde{\Sigma}_N \) that is valid for \( M \).
The visibly pushdown automaton $N$ is called projection of $M$ under $\beta$, if it is constructed in the following way: $N$ will use the same stack alphabet and the same initial and accepting states as $M$. Its set of control states will be a superset of the one of $M$. Additionally, using $\beta$, every transition labeled with $a \in \Sigma_M$ in $M$ will be replaced by a series of $|\beta(a)|$ transitions in $N$ using $|\beta(a)| - 1$ new control states in between. The stack symbols used in the stack operations of $\beta(a)$ on $N$ will be the same symbol as a used in $M$ and if $\beta(a)$ contains subsequent return- and call-transitions, then these temporarily removed stack symbols will be stored in the control state.

**Example 21.** Looking at Example 5 and Example 19 the projection $N$ of $M$ under $\beta$ can now be constructed (disregarding the fact that $\beta$ is not valid for $M$). The automaton $N = (Q', Q_1, \Gamma', \delta', Q_F)$ is given by

\[
Q' = Q \cup \{q_0, q_{0.1}, q_{0.2}, q_{0.3}, q_{0.4}, q_{0.5}, q_{0.6}, q_{1.1}, q_{1.2}, q_{1.3}, q_{1.4}, q_{1.5}, q_{1.6}, q_{3.1}, q_{3.2}, q_{3.3}, q_{3.4}, q_{3.5}, q_{3.6}\}
\]

\[
\Gamma' = \Gamma \cup \{3\}
\]

and this state diagram.

![State diagram](image)

Note how the transitions $q_0 \rightarrow q_F$, $q_1 \rightarrow q_2$ and $q_3 \rightarrow q_4$ have been changed into six transitions respectively with five new states in between. The smaller size of the intermediate states for illustration purposes only.

As a result of this projection, the accepted language of $N$ is a now restricted set of arithmetical formulae on natural numbers.

As a next step the consistency under a projection has to be proven. In other words, the resulting automaton should accept exactly the same words as the original automaton just with every symbol substituted.

**Proposition 22.** Let $N$ be the projection of a visibly pushdown automaton $M$ under the valid symbol substitution $\beta : \Sigma_M \rightarrow \Sigma_N^*$. Then for every word $w = a_1a_2\ldots a_k \in \Sigma_M^*$ it holds that $w \in L(M)$ if and only if $w_\beta = \beta(a_1)\beta(a_2)\ldots \beta(a_k) \in L(N)$. In particular there are no words $w' \in L(N)$ that do not have a preimage in $L(M)$ under $\beta$.

**Proof.** Let $\rho$ be a run of $M$ on $w$ and let $\rho'$ be a run of $N$ on $w_\beta$. Induction by $|\rho|$.
For $|\rho| = 1$ it holds that $\varepsilon \in L(M)$ if and only if an initial state of $M$ is also an accepting state. By construction $\varepsilon \in L(N)$.

Induction step. Let $\rho = (q_0, \sigma_0) \ldots (q_k, \sigma_k)$.

"$\Rightarrow$" By definition will the transition $M$ uses by reading $a_k$ from $q_{k-1}$ to $q_k$ be available as a series of transitions in $N$ by reading $\beta(a_k)$ such that $(q_k, \sigma_k)$ is the last configuration of the run $\rho'$ on $w_\beta$.

"$\Leftarrow$" Let $u = \beta(a_1)\beta(a_2)\ldots\beta(a_{k-1})$ and $u' = u\beta(a_k)$ and let $(q, \sigma)$ be the last configuration in the run $\rho'$ on $u'$. By induction premise is $(q_k-1, \sigma_{k-1})$ the last configuration of the run $\rho'$ on $u$.

According to the definition will the transitions $N$ uses by reading $\beta(a_k)$ from $q_{k-1}$ to $q$ be available as a single transition in $M$ by reading $a_k$ such that $(q, \sigma) = (q_k, \sigma_k)$.

Because the symbol substitution $\beta$ is injective and valid for $M$, there will be no words in $L(N)$ that do not have a preimage in $L(M)$ under $\beta$.

Now turn to Case I-c, in which there are both, symbols enlarging for the first and for the second pushdown alphabet. The resulting automata of Theorem 16 will be projected on the given pushdown alphabets by constructing a valid symbol substitution. Because Proposition 22 states that the emptiness of the intersection does not change under projection, the undecidability is retained for the new pushdown alphabets.

**Theorem 23** (Case I-c). Let $M_1$, $M_2$ be two visibly pushdown automata over the pushdown alphabets $\tilde{\Sigma}_1$ and $\tilde{\Sigma}_2$ respectively, which specify two non-regular languages (Case II does not hold) and there is a symbol $l$ which is enlarging for $\tilde{\Sigma}_1$ and a symbol $r$ which is enlarging for $\tilde{\Sigma}_2$. Then the emptiness of the intersection of $M_1$ and $M_2$ is undecidable.

**Proof.** Define the function $\beta : \Sigma^\text{sim} \rightarrow \Sigma^*$ in the following way, where there are four cases for $r$ and $l$ respectively. Because Case II does not apply, these cases are exhaustive.

Is there a symbol $r \in \Sigma_{r1} \cap \Sigma_{r2}$, then $\beta(d) := r$. Otherwise, if there is a symbol $r \in \Sigma_{l1} \cap \Sigma_{c2}$, then there must exist a symbol $t \in \Sigma_{r1} \setminus \Sigma_{c2}$. If the symbol $t \in \Sigma_{l2}$, then $\beta(d) := tr$, else $\beta(d) := rtr$. Finally if there is a symbol $r \in \Sigma_{r1} \cap \Sigma_{l2}$, then there exists a symbol $u \in \Sigma_{c2} \setminus \Sigma_{r1} \setminus \Sigma_{l1}$, therefore the symbol $u \in \Sigma_{c1}$. In this case $\beta(d) := rur$.

For $\beta(b)$ proceed analogous using $l$. Formally $\beta$ is defined as follows.

$$
\beta(d) := \begin{cases} 
 r & \text{if } r \in \Sigma_{r1} \cap \Sigma_{c2} \\
 tr & \text{if } r \in \Sigma_{l1} \cap \Sigma_{c2}, \ t \in \Sigma_{r1} \cap \Sigma_{l2} \\
 rtr & \text{if } r \in \Sigma_{l1} \cap \Sigma_{c2}, \ t \in \Sigma_{r1} \cap \Sigma_{c2} \\
 rur & \text{if } r \in \Sigma_{r1} \cap \Sigma_{l2}, \ u \in \Sigma_{c1} \cap \Sigma_{c2} \\
 l & \text{if } l \in \Sigma_{c1} \cap \Sigma_{c2} \\
 tl & \text{if } l \in \Sigma_{c1} \cap \Sigma_{c2}, \ t \in \Sigma_{l1} \cap \Sigma_{c2} \\
 ttl & \text{if } r \in \Sigma_{c1} \cap \Sigma_{c2}, \ t \in \Sigma_{r1} \cap \Sigma_{c2} \\
 ttl & \text{if } r \in \Sigma_{l1} \cap \Sigma_{c2}, \ u \in \Sigma_{c1} \cap \Sigma_{c2} \\
 \end{cases}
$$

$$
\beta(x) := \beta(b)\beta(d)\beta(b)\beta(d) \\
\beta(y) := \beta(b)\beta(b)\beta(d)\beta(d) \\
\beta(z) := \beta(d)\beta(b)\beta(b)\beta(b)
$$

By construction $\beta$, $\beta(b)$ never contains symbols that are enlarging for $\tilde{\Sigma}_1$ and $\beta(d)$ never contains symbols that are enlarging for $\tilde{\Sigma}_2$. $\beta(b)$ however always contains $r$ and $\beta(d)$ always contains $l$. So neither $\beta(b)$ nor $\beta(d)$ are a prefix of the other and $\beta$ is injective.
Let $\xi_1, \xi_2, \xi_{M1}$ and $\xi_{M2}$ be the stack alteration functions for $\tilde{\Sigma}_1$, $\tilde{\Sigma}_2$, $\Sigma^{\text{sim}}_1$ and $\Sigma^{\text{sim}}_2$ respectively. Then for every symbol $a \in \Sigma^{\text{sim}}$ the equation $\xi_{M1}(a) = \xi_1(\beta(a))$ and the equation $\xi_{M2}(a) = \xi_2(\beta(a))$ hold. Hence $\beta$ is a symbol substitution.

Let $\tau$ be a deterministic Turing Machine and $(M^\tau_1, M^\tau_2)$ be a simulation of $\tau$. In states of the kind $\langle \cdot, \cdot, \# \rangle$ the outgoing transitions are labeled with $b$ or $d$ and in all other states the outgoing transitions are labeled with $x$, $y$ or $z$, according to Definition 13. Lemma 15 states that $M^\tau_1$ is in a state of the former mentioned kind if and only if $M^\tau_2$ is in a state of this kind. Therefore $M^\tau_1$ and $M^\tau_2$ each have the property that for every control state $q$ and every different $a, c \in \Sigma^{\text{sim}}$, $\beta(a)$ is not a prefix of $\beta(c)$ when $a$ and $c$ are labels of outgoing transitions in $q$, and hence $\beta$ is valid for $M^\tau_1$ and valid for $M^\tau_2$.

Let $N^\tau_1$ and $N^\tau_2$ be the projections of $M^\tau_1$ and $M^\tau_2$ under $\beta$. From Proposition 22 it follows that there is a word $w \in L(N^\tau_1) \cap L(N^\tau_2)$ if and only if there is a word $w' \in L(M^\tau_1) \cap L(M^\tau_2)$ and by Corollary 17 this is undecidable.

4 Conclusion

Out of the four cases listed in the table of Proposition 10, three (Case I-a, Case I-c and Case II) have been discussed in this paper. For Case I-a (Theorem 11), in which the pushdown alphabets are the same, the decidability proof was already given by Alur and Madhusudan. In Case II (Theorem 12), at least one of the VPLS is restricted to a regular language and the decidability proof has been sketched.

In Case I-c the pushdown alphabets contain at least one symbol enlarging for the first and one symbol enlarging for the second pushdown alphabet. The undecidability proof consisted of two parts. At first two specific pushdown alphabets were picked (Definition 13) and it was shown that for these specific alphabet partitions the intersection is undecidable by simulating an arbitrary Turing Machine on the empty input (Theorem 16 and Corollary 17). This idea was then generalized by introducing a way to project this result on automata over arbitrary pushdown alphabets, which meet the properties of Case I-c (Definition 18, 20 and Proposition 22). Finally, using this method the undecidability proof was completed (Theorem 23).

In the remaining Case I-b there is an alphabet partition in which there are neutral symbols and either symbols enlarging for the first pushdown alphabet or the second pushdown alphabet, and, because Case II does not apply, both languages are non-regular. The decidability in this case remains to be shown. However, if a decidability or undecidability proof was given for an arbitrary pushdown alphabet of this case, it could be shown, using the same projection method and a similar symbol substitution as in Theorem 23, that this result applies to every instance of this case. Therefore Case I-b is independent of the specific pushdown alphabets.

Acknowledgement

I would like to thank Alexander Malkis, especially for his ideas and support in trying to solve the last case. Furthermore, I would like to express my gratitude to Matthias Heizmann for his patient and critical review of my drafts.
References


Summary in German