Infinite alphabet pushdown automata:
various approaches and comparison
of their consequences

Yulia Dubov
Infinite alphabet pushdown automata:
various approaches and comparison
of their consequences

Research Thesis

Submitted in Partial Fulfillment
of the Requirements for the Degree of Master of Science
in Computer Science

Yulia Dubov

Submitted to the Senate of the Technion - Israel Institute of Technology
Adar I, 5768 Haifa February, 2008
The research thesis was done under the supervision of Assoc. Prof. Michael Kaminski in the Department of Computer Science.

I would like to express my great thanks to Michael for his guidance and support all the way through my second degree studies, for boosting my confidence and helping to maintain positive attitude in the face of obstacles, and last but not list, his invaluable contribution to this research.

I also thank my family. My dear husband Sasha I thank for standing by me during all the time it took me to complete my studies, urging me forward, enduring patiently the consequences of both the ups and the downs of research, and rightly convincing me that it should not prevent us from starting a family. And, finally, I thank my parents, my sister, my dear grandmothers and my parents-in-law for believing in me and extending their support and encouragement from afar in whichever way they could.

The generous financial help of the Technion is gratefully acknowledged.
# Contents

Abstract ................................................................. 1

Notation and Abbreviations ........................................... 3

1 Introduction ..................................................................... 5
  1.1 Definitions .............................................................. 8

2 Infinite alphabet pushdown automata with deterministic and
   non-deterministic reassignment: a separating example .......... 14
  2.1 Infinite-alphabet pushdown automata with deterministic reas-
      signment .................................................................. 15
  2.2 Example ...................................................................... 18
  2.3 Constraints on contents of the stack .............................. 20
  2.4 Final match .............................................................. 22

3 Finite-memory automata with non-deterministic reassignment
   and infinite alphabet pushdown automata with deterministic
   reassignment ................................................................. 25
  3.1 Infinite alphabet pushdown automata with deterministic reas-
       signment, a variation ................................................. 25
  3.2 Simulation ............................................................... 28

4 Adding a stack to finite-state unification based automata ...... 31
  4.1 FSUBA and its equivalent variations ............................. 31
  4.2 Main definitions and results ........................................ 35
  4.3 From grammars to automata ........................................ 39
  4.4 The $nM$-grammars .................................................... 40
  4.5 From automata to grammars ....................................... 41
Contents (cont.)

5 Unification based pushdown automata with deterministic re-assignment
5.1 Definitions ........................................ 46
5.2 Equivalence ........................................ 48

Appendices 54

A Auxiliary statements for Chapter 2 55

B Auxiliary statements for Chapter 3 61
B.1 Why is DR-IPDA_n not stronger than DR-IPDA? ......... 61
B.2 Proof of Theorem 13: the construction .................. 63
B.3 Proof of Theorem 13: the correctness .................... 65

C Auxiliary statements for Chapter 4 78
C.1 The equivalence of FSUBA and FSUBA_n ................. 78
C.2 Proof of Proposition 20 ................................ 83
C.3 Proof of Theorem 24: the formal construction and the proof
   of correctness ........................................... 88
C.4 Proof of Theorem 28 ................................... 93

D Corrections to the proof of Theorem 2 in [1] 103
D.1 The case $n = 0$ .................................... 103
D.2 The general case ($n > 0$) .............................. 104
D.3 The conclusion ....................................... 105

Bibliography 106
List of Figures

1.1 Previously defined models and their relations diagram . . . . . . 7
1.2 Model relation diagrams, featuring new results . . . . . . . . . 7

3.1 The simulation of NR-FMA by DR-IPDA $A'$: schematic illustration . . . . . . . . . . . . . . . . . . . . . . . . . 30

5.1 DR-UBPDA: “snapshot” of the simulation . . . . . . . . . . 50
5.2 Register layout of simulating DR-UBPDA . . . . . . . . . . 53
Abstract

The study of extension of classical automata to languages over infinite alphabets that blossomed during the 1990s counts numerous works. The main idea behind the models in question\(^1\) remains the same: it is analogous to the classical model in that an automaton has a finite set of states, while the ability to deal with infinite alphabets is achieved by equipping the machine with a finite number of registers, each capable of storing any letter from the infinite input alphabet. While reading the input symbol, the automaton is capable of comparing it with the contents of the registers and proceeding according to the results. It also must have a mechanism – reassignment – for altering the contents of the registers.

This general outline presents a range for variations, which affect the power of the model.

For example, finite state unification based automata (FSUBA), introduced in [7] have no constraints on the contents of the registers, while in finite-memory automata (FMA), defined in [4], a register may contain a symbol only if this symbol is not currently stored in any other register. The result is that the former automata are unable to accept a simple language \(\{\sigma_1 \sigma_2 \in \Sigma^2 | \sigma_1 \neq \sigma_2\}\), that is accepted easily by the latter.

In addition, there are various mechanisms for altering the contents of the registers. For example, FMA alters a register by copying the current input into it, i.e., the reassignment is deterministic. In contrast, the look-ahead finite-memory automata (LFMA) defined in [8] (which are similar in many ways to FMA) have non-deterministic reassignment. That is, they may replace the content of a register with an arbitrary symbol from the infinite alphabet, or, in other words, to “guess” a symbol. The implication of this difference is that the languages accepted by LFMA are closed under reversing, while those accepted by FMA are not. In addition, the ability to guess a future input symbol implies that LFMA languages can be defined by a kind of regular expressions, see [8].

At the same time, some variations of a specific model do not necessarily yield a model of different power. For example, as an auxiliary result, in this thesis, we show that adding non-deterministic reassignment to FSUBA does not add power to the model.

\(^1\)The models discussed here are referred to generally as register automata. There is another class of models – pebble automata – which does not extend to pushdown automata, and is not discussed in this work.
Another direction of developing the theory of automata over infinite alphabets is extending the model by adding a push-down store, and introducing the infinite-alphabet counterpart of context-free grammars, as was done in [1]. Their infinite-alphabet pushdown automata (IPDA) has infinite stack alphabet, non-deterministic reassignment, and can be seen as LFMA with a stack\(^2\).

In this thesis we define infinite-alphabet pushdown automata with deterministic reassignment (DR-IPDA) – the model very similar to IPDA, but with deterministic reassignment. We show that this variation does yield a weaker automaton by presenting a separating example. We also show that LFMA can be simulated by DR-IPDA.

Another part of this thesis is devoted to unification based automata. We define NR-UBPDA – unification based pushdown automata with non-deterministic reassignment, by adding non-deterministic reassignment and a push-down store to FSUBA. We also define a variation of context-free grammar over infinite alphabets, and show its equivalence to NR-UBPDA. Finally, we define DR-UBPDA – unification based pushdown automata with deterministic reassignment, and show its equivalence to NR-UBPDA.

\(^2\)Historically, the definition of IPDA was based on FMA, and LFMA was developed later as “IPDA without a stack”.
Notation and Abbreviations

- $\Sigma = \{\sigma_1, \sigma_2, \ldots\}$ denotes an infinite input alphabet.
- Words are always denoted by boldface letters (possibly indexed or primed).
- Bold low-case Greek letters denote words over $\Sigma$.
- Symbols which appear in a word denoted by a boldface letter are always denoted by the same non-boldface letter with some subscript. That is, symbols which appear in $\sigma$ are denoted by $\sigma_i$, symbols which appear in $w$ are denoted by $w_i$, symbols which appear in $X$ are denoted by $X_i$, etc.
- The symbol $#$ does not belong to $\Sigma$. It is either used to denote an empty register in an assignment, or as an element of stack alphabet.
- $|\sigma| = |\sigma_1 \sigma_2 \cdots \sigma_n| = n$ is the length of a word $\sigma$.
- $[w] = \{w_i \neq \# : i = 1, 2, \ldots, n\}$ is the content of $w = w_1 w_2 \cdots w_n$. It consists of all the symbols of $\Sigma$ that appear in the word $w$.
- $\Sigma^r$ is a set of all assignments of length $r$, where each assignment is a word $w = w_1 w_2 \cdots w_r$ over $\Sigma$, such that $w_i \neq w_j$ for all $i \neq j$. That is, each symbol from $\Sigma$ appears in an assignment at most one time.
- $\Sigma^r#$ is a set of all assignments of length $r$, where each assignment is a word $w = w_1 w_2 \cdots w_r$ over $\Sigma \cup \{\#\}$, such that $w_i \neq w_j$ for $i \neq j$, unless $w_i = w_j = \#$. That is, each symbol from $\Sigma$ appears in an assignment at most one time, while $\#$ may appear any number of times.
- FSDA is an abbreviation of finite state datalog automata, defined in [6].
- FMA is an abbreviation of finite-memory automata, defined in [3].
- NR-FMA is an abbreviation of finite-memory automata with non-deterministic reassignment, defined in [8], where it appears as LFMA (lookahead finite-memory automata).
• NR-IPDA is an abbreviation of infinite-alphabet pushdown automata with non-deterministic reassignment, defined in [1], where it appears as IPDA (infinite-alphabet pushdown automata).

• DR-IPDA is an abbreviation of infinite-alphabet pushdown automata with deterministic reassignment, defined in Chapter 2 on page 15.

• DR-IPDA_n is an abbreviation of new infinite-alphabet pushdown automata with deterministic reassignment, defined in Chapter 3 on page 26.

• FSUBA is an abbreviation of finite-state unification based automata, defined in [7].

• FSUBA_n is an abbreviation of new finite-state unification-based automata, defined in Chapter 4 on page 32.

• NR-FSUBA is an abbreviation of finite-state unification based automata with non-deterministic reassignment, defined in Chapter 4 on page 34.

• NR-UBPDA is an abbreviation of unification based pushdown automata with non-deterministic reassignment, defined in Chapter 4 on page 36.

• UBCFG is an abbreviation of unification based context-free grammar, defined in Chapter 4 on page 37.

• nM-grammar is an abbreviation of unification-based context-free grammar with multiple reassignment, defined in Chapter 4 on page 40.

• NR-UBPDA_n is an abbreviation of new unification based pushdown automata with non-deterministic reassignment, defined in Chapter 5 on page 46.

• DR-UBPDA is an abbreviation of unification based pushdown automata with deterministic reassignment, defined in Chapter 5 on page 47.
Chapter 1

Introduction

The study of extension of classical automata to languages over infinite alphabets that blossomed during the 1990s counts numerous works. It started as purely theoretical, but soon found its applications. Naturally, every time a string of words is considered - be it messages passing through the network, URLs clicked by Internet surfer, or XML tags - words are treated as atomic symbols, i.e. elements of an alphabet, and in the absence of a bound on the length of the words, the alphabet becomes infinite.

One of the approaches to constructing finite models over infinite alphabets is register automata.\(^1\) It first featured in [6] that introduced the model called finite state datalog automata (FSDA), intended for the abstract study of relational languages.

The common idea behind the group of models in question is as follows: it is analogous to the classical model in that it has a finite set of states, while the ability to deal with infinite alphabets is achieved by equipping the machine with a finite number of registers, each capable of storing any letter from the infinite input alphabet. While reading the input symbol, the automaton is capable of comparing it with the contents of the registers and proceeding according to the results. It also must have a mechanism – reassignment – for altering the contents of the registers.

This general outline presents a range for variations affecting the power of the model, many of which were explored since the introduction of the concept.

\(^1\)There is another class of models which address this problem, called pebble automata. These automata do not extend to pushdown automata, and are not discussed in our work.
For example, *finite state unification based automata* (FSUBA), introduced in [7] have no constraints on the contents of the registers, while in *finite-memory automata* (FMA), defined in [4], a register may contain a symbol only if this symbol is not currently stored in any other register. The result is that the former automata are unable to accept a simple language \( \{ \sigma_1 \sigma_2 \in \Sigma^2 | \sigma_1 \neq \sigma_2 \} \), that is accepted easily by the latter.

In addition, there are various mechanisms for altering the contents of the registers. For example, FMA alters a register by copying the current input into it, i.e., the reassignment is *deterministic*. In contrast, the *look-ahead finite-memory automata* (LFMA)\(^2\) defined in [8] (which are similar in many ways to FMA) have *non-deterministic* reassignment. That is, they may replace the content of a register with an arbitrary symbol of the infinite alphabet, or, in other words, to “guess” a symbol. The implication of this difference is that the languages accepted by LFMA are closed under *reversing*, while those accepted by FMA are not. In addition, the ability to guess a future input symbol implies that LFMA languages can be defined by a kind of *regular expressions*, see [8].

At the same time, some variations of a specific model do not necessarily yield a model of different power. For example, as an auxiliary result, in this thesis, we show that adding non-deterministic reassignment to FSUBA does not add power to the model.

Another direction of developing the theory of automata over infinite alphabets is extending the model by adding a push-down store, and introducing the infinite-alphabet counterpart of *context-free grammars*, as was done in [1]. Their *infinite-alphabet pushdown automata* (IPDA)\(^3\) has infinite stack alphabet, non-deterministic reassignment, and can be seen as NR-FMA with a stack, though historically it was defined based on FMA, before NR-FMA was introduced.

Schematically, the various models of automata mentioned above form the picture presented by Figure 1.1. The meaning of \( \rightarrow \) from model A to model B is that the class of languages accepted by A is a subset of the class of languages accepted by B.

In our research we complete this picture by adding new models and examining their place among the others. Our results are schematically summarized by Figure 1.2.

The thesis is organized as follows. In Chapter 2 we ponder the question why NR-IPDA has non-deterministic reassignment. We define infinite-

\(^2\)Starting from the next paragraph and throughout this thesis, we refer to this model
Figure 1.1: Previously defined models and their relations diagram

Figure 1.2: Model relation diagrams, featuring new results
alphabet pushdown automata with deterministic reassignment (DR-IPDA) –
the model otherwise very similar to NR-IPDA, and show that this variation
does yield a weaker automaton by presenting a separating example. In Chap-
ter 3 we show that NR-FMA can be simulated by DR-IPDA. In Chapter 4,
we define NR-UBPDA – unification based pushdown automata with non-
deterministic reassignment, by adding a push-down store to FSUBA. We also
define a variation of context-free grammar over infinite alphabets, and show
its equivalence to NR-UBPDA. Finally, in Chapter 5 we define DR-UBPDA –
unification based pushdown automata with deterministic reassignment, and
show its equivalence to NR-UBPDA.

In the following sections we introduce the basic notions used throughout
the thesis and recite the definitions of the previously defined models.

1.1 Definitions

Let \( \Sigma \) be an infinite alphabet, and let \( \# \not\in \Sigma \) be a “blank” symbol, usually
denoting an empty register.

Throughout the thesis we use the term assignment to refer to a word
representing the contents of all the registers of an automaton or a grammar.
However, since different models have different constraints on the contents
of the registers, the assignments may be of a different form. We use three types
of assignments.

1. The assignments of the form \( w_1 w_2 \cdots w_r \in (\Sigma \cup \{\#\})^* \) such that \( w_i \neq w_j \)
   for \( i \neq j \), unless \( w_i = w_j = \# \). That is, each symbol from \( \Sigma \) appears
   in an assignment at most one time, while \( \# \) may appear any number
   of times. We denote the set of all such assignments of length \( r \) by
   \( \Sigma^r\# \). The assignments of this type are featured in NR-FMA, defined
   in Subsection 1.1.1, and are used in Chapter 3.

2. The assignments of the form \( w_1 w_2 \cdots w_r \in \Sigma^* \) such that \( w_i \neq w_j \)
   for \( i \neq j \). That is, each symbol from \( \Sigma \) appears in an assignment at most
   one time. We denote the set of all such assignments of length \( r \) by
   \( \Sigma^r \). The assignments of this type are featured in NR-IPDA, defined
   in Subsection 1.1.2, and are used in Chapter 2.
(3) The assignments of the form \( w_1w_2 \cdots w_r \in (\Sigma \cup \{\#\})^* \) without any constraints. Naturally, the set of all such assignments of length \( r \) is denoted by \((\Sigma \cup \{\#\})^r\). The assignments of this type are featured in FSUBA, defined in Subsections 1.1.3, and are used in Chapter 4.

For a word \( w = w_1w_2 \cdots w_n \in (\Sigma \cup \{\#\})^* \), we define the content of \( w \), denoted \([w]\), by \([w] = \{w_i \neq \# : i = 1, 2, \ldots, n\}\). That is, \([w]\) consists of all the symbols of \( \Sigma \) which appear in the word \( w \).

Throughout this paper we use the following conventions.

- Words are always denoted by boldface letters (possibly indexed or primed).
- Boldface low-case Greek letters denote words over \( \Sigma \).
- Symbols which appear in a word denoted by a boldface letter are always denoted by the same non-boldface letter with some subscript. That is, symbols which appear in \( \sigma \) are denoted by \( \sigma_i \), symbols which appear in \( w \) are denoted by \( w_i \), symbols which appear in \( X \) are denoted by \( X_i \), etc.

### 1.1.1 Finite-memory automata with non-deterministic reassignment

**Definition 1.** [8, Definition 4] A finite-memory automaton with non-deterministic reassignment (NR-FMA)\(^4\), is a system \( \mathcal{A} = (Q, s_0, u, \mu, \rho, F) \), where

- \( Q \) is a finite set of states.
- \( s_0 \in Q \) is the initial state.
- \( u = u_1u_2 \cdots u_r \in \Sigma_\#^r \) is the initial assignment to the registers of \( \mathcal{A} \).
- \( \mu \subseteq Q \times (\{1, 2, \ldots, r\} \cup \{\epsilon\}) \times Q \) is the transition relation. Intuitively, if \( \mathcal{A} \) is in state \( p \), the input symbol is equal to the content of the \( k \)th register, and \( (p, k, q) \in \mu \), then \( \mathcal{A} \) may enter state \( q \) and pass to the next input symbol. Similarly, if \( (p, \epsilon, q) \in \mu \), then \( \mathcal{A} \) may make a non-deterministic reassignment using the function \( \rho \). That is, it may replace the content of the \( \rho(p, q) \)th register with any element of \( \Sigma \), that does

\(^4\)In [8] the model is called look-ahead finite-memory automata (LFMA).
not appear in any other register, and enter state \( q \) without reading the next input symbol.

- \( \rho : \{(p,q) : (p,\epsilon,q) \in \mu\} \rightarrow \{1,2,\ldots,r \cup \{\text{nil} \}\} \) is the reassignment function. Intuitively, if \( A \) is in state \( p \), \( (p,\epsilon,q) \in \mu \) and \( \rho(p,q) = k \), then for \( k \neq \text{nil} \), \( A \) can non-deterministically replace the content of the \( k \)th register with an element of \( \Sigma \) not appearing in any other registers and enter state \( q \).

- \( F \subseteq Q \) is the set of final states.

An instantaneous description of \( A \) is a member of \( Q \times \Sigma^r \times \Sigma^* \). The first component of an instantaneous description is the (current) state of the automaton, the second one is the assignment consisting of the contents of the registers (in the increasing order of their indices), and the third component is the portion of the input yet to be read.

Next, we define the relation \( \vdash \) (yielding in one step) between two instantaneous descriptions \( (p,v_1v_2\cdots v_r,\sigma\sigma) \) and \( (q,w_1w_2\cdots w_r,\sigma) \), \( \sigma \in \Sigma \cup \{\epsilon\} \). We write

\[
(p,v_1v_2\cdots v_r,\sigma\sigma) \vdash (q,w_1w_2\cdots w_r,\sigma),
\]

if the following holds.

- If \( \sigma = \epsilon \) and \( \rho(p,q) = k \), then for \( k \neq \text{nil} \),

\[
w_k \in \Sigma \setminus \{v_1,v_2,\ldots,v_{k-1},v_{k+1},\ldots,v_r\}
\]

and for each \( l \neq k \), \( w_l = v_l \). Otherwise, i.e., \( \sigma \neq \epsilon \) or \( k = \text{nil} \), \( w = v \).

- If \( \sigma \neq \epsilon \), then for some \( k \), \( \sigma = v_k \), and \( (p,k,q) \in \mu \).

We denote the reflexive and transitive closure of \( \vdash \) by \( \vdash^* \) and say that \( A \) accepts a word \( \sigma \in \Sigma^* \), if \( (s_0,u,\sigma) \vdash^* (f,v,\epsilon) \) for some \( f \in F \) and \( v \in \Sigma^r \). The set of all words accepted by \( A \) is denoted by \( L(A) \).

### 1.1.2 Infinite-alphabet pushdown automata with non-deterministic reassignment

**Definition 2.** [1, Definition 2] An infinite-alphabet pushdown automaton with non-deterministic reassignment (NR-IPDA)\(^5\) is a system \( A = (Q,s_0,u,\rho,\mu) \) where

\(^5\)In [1] the model is called infinite-alphabet pushdown automata (IPDA).
• $Q$ is a finite set of states.

• $S_0 \in Q$ is the initial state.

• $u = u_1u_2 \cdots u_r \in \Sigma^*$, is the initial assignment to the $r$ registers of $A$.

• $\rho : Q \rightarrow \{1,2,\ldots, r\}$ is a partial function from $Q$ to $\{1,2,\ldots, r\}$ called the reassignment. Intuitively, if $A$ is in state $q$, and $\rho(q)$ is defined, then $A$ can non-deterministically replace the content of the $\rho(q)$th register with a new symbol of $\Sigma$ not appearing in any other register.

• $\mu$ is a mapping from $Q \times (\{1,2,\ldots, r\} \cup \{\epsilon\}) \times \{1,2,\ldots, r\}$ to finite subsets of $Q \times \{1,2,\ldots, r\}$ called the transition function. Intuitively, if $(p,j_1j_2 \cdots j_n) \in \mu(q,k,i)$, then (after reassigning the $\rho(q)$th register) $A$, whenever it is in state $q$, with content of the $i$th register at the top of the stack, and the input symbol is equal to the content of $k$th register, can replace the top symbol on the stack with the content of $j_1$th,$j_2$th,$\ldots,j_n$th registers (in this order, read top-down), enter the state $p$, and pass to the next input symbol (possibly $\epsilon$). Similarly, if $(p,j_1j_2 \cdots j_n) \in \mu(q,\epsilon,i)$, then $A$, whenever it is in state $q$, with content of the $i$th register at the top of the stack, can replace the top symbol on the stack with the content of $j_1$th,$j_2$th,$\ldots,j_n$th registers, enter state $p$ (without reading the input symbol).

An instantaneous description is a member of $Q \times \Sigma^r \times \Sigma^* \times \Sigma^*$. The first component of an instantaneous description is the (current) state of the automaton, the second is the assignment consisting of the contents of the registers (in the increasing order of their indices), the third component is the portion of the input yet to be read, and the last one is the contents of the pushdown store, read top down.

Next we define the relation $\vdash$ (yielding in one step) between two instantaneous descriptions $(p,w_1w_2 \cdots w_r, \sigma\sigma, \tau\tau)$ and $(q,v_1v_2 \cdots v_r, \sigma, \alpha\tau)$, $\sigma \in \Sigma \cup \{\epsilon\}$, $\tau \in \Sigma$. We write $(p,w_1w_2 \cdots w_r, \sigma\sigma, \tau\tau) \vdash (q,v_1v_2 \cdots v_r, \sigma, \alpha\tau)$ if and only if the following holds.

• If $\rho(p)$ is not defined, then $v_k = w_k, k = 1,2,\ldots, r$. Otherwise $v_k = w_k$ for $k \neq \rho(p)$ and $v_{\rho(p)} \in \Sigma - \{w_1,\ldots, w_{\rho(p)-1}, w_{\rho(p)+1},\ldots, w_r\}$.

• If $\sigma = \epsilon$, then for some $i$, $\tau = v_i$ and there is $(q,j_1j_2 \cdots j_n) \in \mu(p,\epsilon,i)$ such that $\alpha = v_{j_1}v_{j_2} \cdots v_{j_n}$.
If \( \sigma \neq \epsilon \), then for some \( k \) and \( i \), \( \sigma = v_k \), \( \tau = v_i \), and there is 

\[(q, j_1, j_2, \ldots, j_n) \in \mu(p, k, i) \]

such that \( \alpha = v_{j_1}v_{j_2} \cdots v_{j_n} \).

We denote the reflexive and transitive closure of \( \vdash \) by \( \vdash^* \) and say that \( \mathcal{A} \) accepts a word \( \sigma \in \Sigma^* \) if \((s_0, u, \sigma, u_r) \vdash^* (p, v, \epsilon, \epsilon)\), for some \( p \in Q \) and some \( v \in \Sigma^* \). Recall that the initial assignment \( u = u_1u_2 \cdots u_r \) is of length \( r \). Thus, the language accepted by the automaton is defined by

\[ L(\mathcal{A}) = \{ \sigma \in \Sigma^* | \exists p \in Q, v \in \Sigma^* : (s_0, u, \sigma, u_r) \vdash^* (p, v, \epsilon, \epsilon) \} \]

### 1.1.3 Finite-state unification based automata

**Definition 3.** [7] A finite-state unification based automaton (over \( \Sigma \)) or, shortly, FSUBA, is a system \( \mathcal{A} = (\Sigma, Q, q_0, F, u, \Theta, \mu) \), where

- \( Q, q_0 \in Q \) and \( F \subseteq Q \) are a finite set of states, the initial state, and the set of final states, respectively.
- \( u = u_1u_2 \cdots u_r \in (\Sigma \cup \{\#\})^r, r \geq 1 \), is the *initial assignment* - register initialization: the symbol in the \( i \)th register is \( u_i \). If \( u_i = \# \), the \( i \)th register is considered empty.
- \( \Theta \subseteq [u] \) is the “read only” alphabet, whose symbols can not be copied into the empty registers.
- \( \mu \subseteq Q \times \{1, 2, \ldots, r\} \times 2^{\{1, 2, \ldots, r\}} \times Q \) is the transition relation whose elements are called transitions. The intuitive meaning of \( \mu \) is as follows. If the automaton is in state \( q \) reading symbol \( \sigma \) and there is a transition \((q, k, S, q') \in \mu \) such that the \( k \)th register either contains \( \sigma \) or is empty, then the automaton can enter state \( q' \), write \( \sigma \) in the \( k \)th register (if it is empty), and erase the content of the registers whose indices belong to \( S \). The \( k \)th register will be referred to as the transition register.

The pairs \((q, w)\), where \( q \in Q \) and \( w \in (\Sigma \cup \{\#\})^r \), are called *configurations* of \( \mathcal{A} \). The set of all configurations of \( \mathcal{A} \) is denoted \( Q^c \). The pair \((q_0, u)\), denoted \( q_0^c \), is called the *initial configuration*, and the configurations with the first component in \( F \) are called *final configurations*. The set of final configurations is denoted \( F^c \).

Transition relation \( \mu \) induces the following relation \( \mu^c \) on \( Q^c \times \Sigma \times Q^c \). Let \( q, q' \in Q, w = w_1w_2 \cdots w_r \) and \( w' = w'_1w'_2 \cdots w'_r \). A triple \(((q, w), \sigma, (q', w'))\)
belongs to \( \mu^c \) if and only if there is a transition \((q, k, S, q')\) in \( \mu \), such that the following conditions are satisfied.

- Either \( w_k = \# \) (i.e., the transition register is empty in which case \( \sigma \) is copied into it) and \( \sigma \notin \Theta \), or \( w_k = \sigma \) (i.e., the transition register contains \( \sigma \)).

- If \( k \notin S \), then \( w_k' = \sigma \), i.e., if the transition register is not reset in the transition, its content is \( \sigma \).

- For all \( j \in S \), \( w_j' = \# \).

- For all \( j \notin S \cup \{k\} \), \( w_j' = w_j \).

Let \( \sigma = \sigma_1 \sigma_2 \cdots \sigma_n \) be a word over \( \Sigma \). A run of \( \mathcal{A} \) on \( \sigma \) consists of a sequence of configurations \( c_0, c_1, \ldots, c_n \) such that \( c_0 \) is the initial configuration \( q_0^c \) and \( (c_{i-1}, \sigma_i, c_i) \in \mu^c \), \( i = 1, 2, \ldots, n \).

We say that \( \mathcal{A} \) accepts \( \sigma \), if there exists a run \( c_0, c_1, \ldots, c_n \) of \( \mathcal{A} \) on \( \sigma \), such that \( c_n \in F^c \). The set of all words accepted by \( \mathcal{A} \) is denoted by \( L(\mathcal{A}) \).
Chapter 2

Infinite alphabet pushdown automata with deterministic and non-deterministic reassignment: a separating example

The model in question, NR-IPDA, defined in the introduction, was introduced in [1] as a natural extension of the classical PDA model to infinite alphabets. The idea behind the definition of NR-IPDA is similar to that behind the definition of finite-memory automata introduced in [3], one of the features that distinguish it from the latter being the non-deterministic reassignment, i.e., the ability to change symbols in the registers before each transition non-deterministically, independently of the current input symbol or the symbol found on the top of the stack. An appropriate question is whether this ability is necessary, i.e., whether restricting the model by requiring the reassignment to be deterministic would yield the model of the same power.

In this chapter we prove that the answer to this question is negative by presenting a language, that can be accepted by an automaton of the original model, but can not be accepted by infinite alphabet pushdown automaton with deterministic reassignment, the model we define here.

Throughout this chapter, an assignment is a word $w_1 w_2 \cdots w_r \in \Sigma^*$ such that $w_i \neq w_j$ for $i \neq j$. That is, an assignment is a word over $\Sigma$, where
each symbol from Σ appears at most one time. We denote the set of all assignments of length r by \( \Sigma^r \).

### 2.1 Infinite-alphabet pushdown automata with deterministic reassignment

Here we define infinite-alphabet pushdown automata with deterministic reassignment (DR-IPDA). Instead of being able to “guess” symbols for its reassignment, this model is limited to altering its registers by replacing their contents with symbols currently appearing in the input (similarly to FMA of [3]) or at the top of the stack.

**Definition 4.** A infinite-alphabet pushdown automaton with deterministic reassignment (DR-IPDA) is a system \( A = \langle Q, q_0, u, \pi, \rho, \mu \rangle \) where
- \( Q \) is a finite set of states
- \( q_0 \in Q \) is the initial state
- \( u = u_1u_2...u_r \in \Sigma^r \) is the initial assignment to the \( r \) registers of \( A \)
- \( \pi, \rho : Q \rightarrow \{1, 2, ..., r\} \) both are functions from \( Q \) to \( \{1, 2, ..., r\} \) called stack-based reassignment and input-based reassignment respectively.
- \( \mu \) is a mapping from \( Q \times (\{1, 2, \ldots, r\} \cup \{\epsilon\}) \times \{1, 2, \ldots, r\} \) to finite subsets of \( Q \times \{1, 2, \ldots, r\}^* \) called the transition function.

Intuitively, a computation step of the automaton \( A \) in state \( q \) is composed of the following actions.
- First, the stack-based reassignment is performed\(^1\), namely, if the symbol that currently appears on the top of the stack does not appear in any register, the content of the register \( \pi(q) \) is replaced by this symbol, otherwise the memory is unchanged.
  - After that, \( A \) may perform \( \epsilon \)-transition if defined, and the computation step is over.

\(^1\)The order of the reassignments is significant. Stack-based reassignment has to be performed first to allow the automaton to perform an \( \epsilon \)-move, in which case input-based reassignment will not be performed at all.
If no \( \epsilon \)-transition was performed, \( A \) proceeds to input-based reassignment, namely, if the current input symbol does not appear in any register, the content of the register \( \rho(q) \) is replaced by this symbol, otherwise the memory is left unchanged.

- After that, the automaton may perform a transition, which involves reading from the input, if defined.

The definition of the \textit{instantaneous description} of DR-IPDA is similar to the corresponding definition for NR-IPDA. The relation \( \vdash \) (yielding in one step) between two instantaneous descriptions of DR-IPDA is defined as follows. We write \((p, w_1 w_2 \ldots w_r, \sigma \sigma, \tau \tau) \vdash (q, v_1 v_2 \ldots v_r, \sigma, \alpha \tau)\), where \( \sigma \in \Sigma \cup \{\epsilon\} \), \( \tau \in \Sigma \), if and only if the following holds:

- Let \( \sigma = \epsilon \):
  
  - If there is an \( i \) such that \( w_i = \tau \), then \( v_1 v_2 \cdot \cdot \cdot v_r = w_1 w_2 \cdot \cdot \cdot w_r \). Otherwise, \( v_{\pi(p)} = \tau \) and \( v_i = w_i \) for each \( i \neq \pi(p) \).
  
  - For some \( j \), \( v_j = \tau \) and there is \((q, k_1 k_2 \ldots k_l) \in \mu(p, \epsilon, j)\), such that \( \alpha = v_{k_1} v_{k_2} \ldots v_{k_l} \).

- Let \( \sigma \neq \epsilon \):

  Let \( w'_1, w'_2, \ldots, w'_r \) be defined as follows. If there is an \( i \) such that \( w_i = \tau \), then \( w'_1 w'_2 \cdot \cdot \cdot w'_r = w_1 w_2 \cdot \cdot \cdot w_r \). Otherwise, \( w'_{\rho(p)} = \tau \) and \( w'_i = w_i \) for each \( i \neq \rho(p) \). Then:
  
  - If there is an \( i \) such that \( u_i = \sigma \), then \( v_1 v_2 \cdot \cdot \cdot v_r = w'_1 w'_2 \cdot \cdot \cdot w'_r \). Otherwise, \( v_{\rho(p)} = \sigma \) and \( v_i = w'_i \) for each \( i \neq \rho(p) \).
  
  - For some \( i, j \), \( v_i = \sigma \), \( v_j = \tau \) and there is \((q, k_1 k_2 \ldots k_l) \in \mu(p, i, j)\), such that \( \alpha = v_{k_1} v_{k_2} \ldots v_{k_l} \).

The language accepted by DR-IPDA \( A \) is defined by

\[
L(A) = \{ \sigma \mid (q_0, u, \sigma, u_r) \vdash^* (p, v, \epsilon, \epsilon) \},
\]

where \( \vdash^* \) denotes the reflexive and transitive closure of relation \( \vdash \).
2.1.1 Some simple properties of DR-IPDA

For an instantaneous description $c = (q, v, x, \alpha)$, we shall call the set $[vx\alpha]$ the active alphabet of $c$. Due to the nature of deterministic reassignment, the instantaneous descriptions of DR-IPDA possess the property of active alphabet monotony, i.e., the active alphabet of subsequent instantaneous description is always a subset of an active alphabet of the preceding one. This straightforward observation is formulated by the following proposition.

**Proposition 5.** If for some DR-IPDA, $(p, v, x, \alpha) \vdash^* (q, w, y, \beta)$, then $[wy\beta] \subseteq [vx\alpha]$.

**Proof.** The proof easily follows from the definition of $\vdash$, by induction on the number of computation steps. \hfill \Box

The next observation will be used in the following sections.

**Proposition 6.** Let $\tau : \Sigma \rightarrow \Sigma$ be an automorphism of $\Sigma$. Then, for any DR-IPDA, $(p, v, x, \alpha) \vdash^* (q, w, \epsilon, \beta)$ if and only if $(p, \tau(v), \tau(x), \tau(\alpha)) \vdash^* (q, \tau(w), \epsilon, \tau(\beta))$.

**Proof.** The proof easily follows from the definition of $\vdash$, by induction on the number of computation steps. \hfill \Box

2.1.2 Why DR-IPDA is not stronger\textsuperscript{2} than NR-IPDA?

Here we shortly describe, without giving the formal construction, how NR-IPDA can simulate DR-IPDA. Essentially, the question is how to simulate deterministic reassignment with non-deterministic one.

To simulate $\epsilon$-moves, we distinguish between the following two cases. Let $j$ be the index of the register that is compared to the top of the stack during the move.

- $j = \pi(q)$, i.e. the top of the stack is compared to the “reassignment register”. In such moves, instead of copying a symbol from the stack into register $\pi(q)$, NR-IPDA “guesses” a symbol into the register, and then makes the move. If the symbol is guessed correctly, the transition is performed, otherwise, the computation halts.

\textsuperscript{2}Throughout this thesis, by saying that model A “is not stronger” than model B, we mean that the class of languages accepted by model A is a subset of a class of languages accepted by model B.
• \( j \neq \pi(q) \), i.e. symbol on the top of the stack appears in register \( j \) before the reassignment. In such case, guessing into register \( \pi(q) \) in NR-IPDA may result in register content that does not conform to the original computation of DR-IPDA. For such cases NR-IPDA utilizes an additional register, and “guesses” a symbol into it. In such way, the original content of the “simulation” registers is preserved, and if the “guessed” symbol does not belong to the active alphabet of DR-IPDA’s computation, nothing prevents further simulation.

If sometime during the computation a symbol from the active alphabet is “guessed” into the additional register, it may prevent the simulation of copying this symbol into a simulation register, in which case the computation halts.

As for the non-\( \epsilon \)-moves, which involve comparing the input to register \( i \), they are simulated by a pair of moves (unless \( \pi(q) = \rho(q) \), in which case one move suffices). The first is an \( \epsilon \)-move, responsible for simulating the stack-based reassignment, that leaves the top of the stack in place. The second move is to simulate the input-based reassignment. Both are done in the way described above. Of course, special cases, like \( i = j = \rho(q) \neq \pi(q) \), must be taken into account.

2.2 Example.

We proceed to presenting the example that distinguishes between DR-IPDA and NR-IPDA.

The idea behind our example is as follows. We observe that NR-IPDA can perform certain tasks, using both its stack and its guessing abilities, that DR-IPDA can not. Specifically, consider a computation involving an unknown future input and the current input. NR-IPDA can perform it by “guessing” the future input and using it throughout reading the input word. DR-IPDA, on the other hand, would need to employ its pushdown store to perform the same task by storing the input word in it until it reaches the input symbols it could not guess. Now, if we add a different task that requires the use of the pushdown store and has to be performed simultaneously, both of the tasks together cannot be performed by DR-IPDA, but can by NR-IPDA.

Thus, we define the language \( L \) over infinite alphabet \( \Sigma \), where \( $ \in \Sigma \), in
the following manner:

\[
L = \left\{ \$^{|w|} w \sigma \mid \$ \notin [w] \text{ and } \sigma \notin [w] \cup \{\$\} \right\}.
\]

It is easy to see that an NR-IPDA, which guesses and remembers \( \sigma \) in one of the registers at the beginning, then uses its stack to compare the number of \( \$ \)s to the number of symbols in \( w \), making sure each symbol in \( w \) is different from \( \sigma \) as it goes along, accepts \( L \). For a more formal discussion, we resort to an alternative method of showing that a language is accepted by NR-IPDA, namely, we show that it is quasi-context-free by presenting a quasi-context-free grammar.

The definition of infinite-alphabet context free grammar, introduced in [1], is as follows.

**Definition 7.** ([1, Definition 1]) An **infinite-alphabet context-free grammar** is a system \( G = (V, u, R, S) \), where

- \( V \) is a finite set of **variables**, \( V \cap \Sigma = \emptyset \).
- \( u = u_1 u_2 \cdots u_r \in \Sigma^r \) is the **initial** assignment.
- \( R \subseteq V \times \{1,2,\ldots,r\} \times (V \cup \{1,2,\ldots,r\})^* \) is a set of **productions**. For \( A \in V, i = 1,2,\ldots,r, \text{ and } a \in (V \cup \{1,2,\ldots,r\})^* \), we write the triple \((A,i,a) \) as \((A,i) \to a \).
- \( S \in V \) is the **start symbol**.

For \( A \in V, w = w_1 w_2 \cdots w_r \in \Sigma^r \), and \( X = X_1 X_2 \cdots X_n \in (\Sigma \cup (V \times \Sigma^r))^* \), we write \((A,w) \Rightarrow X \) if there exist a production \((A,i) \to a \in R, a = a_1 a_2 \cdots a_n \in (V \cup \{1,2,\ldots,r\})^*\) and \( \sigma \notin [w] \cup \{w_i\} \) such that the condition below is satisfied.

Let \( w' \in \Sigma^r \) be obtained from \( w \) by replacing \( w_i \) with \( \sigma \). Then, for \( j = 1,2,\ldots,n \) the following holds.

- If \( a_j = k \) for some \( k = 1,2,\ldots,r \), then \( X_j = w'_k \).
- If \( a_j = B \) for some \( B \in V \), then \( X_j = (B, w') \).

For two words \( X \) and \( Y \) over \( \Sigma \cup (V \times \Sigma^r) \), we write \( X \Rightarrow Y \) if there exist words \( X_1, X_2, \text{ and } X_3 \) over \( \Sigma \cup (V \times \Sigma^r) \), and \( (A,w) \in V \times \Sigma^r \), such that \( X = X_1(A,w)X_2, Y = X_1X_3X_2 \) and \((A,w) \Rightarrow X_3 \). As usual, the
reflexive and transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. The language $L(G)$ generated by $G$ is defined by $L(G) = \{ \sigma \in \Sigma^* : (S, u) \Rightarrow^* \sigma \}$ and is referred to as a quasi-context-free language.

In [1] it is shown that quasi-context-free languages are exactly the languages accepted by NR-IPDA. Therefore, it is enough to present a grammar that generates $L$ to show that it is accepted by NR-IPDA.

It is not hard to verify that the following grammar indeed generates $L$:

$$G = (\{S,A\}, \$ \sigma_1 \sigma_2, \{(S,2) \rightarrow A2, (A,3) \rightarrow 1A3|\epsilon\}, S).$$

Now, to the reason we are convinced that DR-IPDA cannot accept $L$. Intuitively, one can see, that since DR-IPDA is unable to guess $\sigma$ at the beginning of the computation, it leave us with the only option: to store $w$ on the stack until we reach $\sigma$, an then compare each symbol of $w$ to $\sigma$ by “popping” them one by one. The problem is, that the stack is also needed to compare the number of $\$$s to $|w|$, and apparently, two of these tasks can not be performed at the same time. Essentially, it is the same intuition that makes us see immediately that $\{a^i b^i c^i | i > 0\}$ is not an ordinary context-free language.

Despite its obviousness, as in most cases, obtaining the negative result is considerably harder than the positive. Thus, the rest of the chapter is devoted to the proof of the following theorem:

**Theorem 8.** For any DR-IPDA $A$, $L(A) \neq L$.

To begin, we assume to the contrary, that there is an $r$-register DR-IPDA

$$A =< Q, q_0, u_0, \pi, \rho, \mu >$$

that accepts $L$. The following section is devoted to the proof of various properties $A$ would possess if it existed, which, as we show in Section 2.4, inevitably bring us to a contradiction, that will complete the proof.

### 2.3 Constraints on contents of the stack

Let us consider a word $z = \$^{|w|}w \sigma \in L$ such that $|w|$ is greater than $r$ – the number of registers in $A$, and none of the symbols of $w$ appear in it twice, as well as none of the symbols of $w$ or $\sigma$ appear in the initial assignment of $A$. Let $c_0, c_1, \ldots, c_n$ be an accepting run of $A$ on $z$, where each $c_i$ is of
the form \((q, u, z', \alpha)\), \(q\) being the current state of the automaton, \(u\) - the assignment of the registers, \(z'\) - the remaining input, i.e. a suffix of \(z\), and \(\alpha\) - the contents of the stack read top down.

The statements below are related to the objects described above. The proofs are omitted, and are presented for completeness in Appendix A.

**Proposition 9.** For each \(c_i = (q, u, z', \alpha)\) such that \(z' \neq \epsilon\) in the above run, \([w] \subseteq [uz'\alpha]\).

The following lemma formalizes the claim, that somewhere between starting reading \(w\), and reading the \(r\)th symbol of \(w\), \(A\) reaches a configuration with a nonempty string in the bottom of the stack, which remains unchanged during the run until all the input is read.

**Lemma 10.** There is an instantaneous description \(c_{i_0} = (p, u, w'^\sigma, \alpha'\alpha')\), such that the following holds:

1. \(w'\) is a suffix of \(w\) and \(|w| - |w'| \leq r\),
2. \(\alpha' \neq \epsilon\),
3. \(|\alpha| < m\), where \(m\) is a constant that depends only on \(A\),
4. for each \(j \geq i_0\), if \(c_j = (q, v, w''\sigma, \beta)\), then \(\beta = \beta'\alpha'\).

By means of Lemma 10 we have located the instantaneous description \(c_{i_0}\), with \(\alpha'\) at the bottom of the stack which stays unchanged until the whole input is read.

The next lemma serves to locate an instantaneous description that describes the following situation: the whole input is read, but \(\alpha'\) at the bottom of the stack has not yet been altered.

**Lemma 11.** Let \(c_{i_0} = (p, u, w'^\sigma, \alpha'\alpha')\) be the instantaneous description provided by Lemma 10. Then there is an instantaneous description \(c_{i_1} = (q, v, \epsilon, \alpha')\) in the run of \(A\),\(^3\) such that for any \(c_j = (\tilde{p}, \tilde{u}, z', \gamma)\), such that \(i_0 \leq j \leq i_1\), \(\gamma\) is of the form \(\tilde{\alpha}\alpha'\).

\(^3\)Clearly, \(i_1 > i_0\).
2.4 Final match

To arrive at the desired contradiction and prove Theorem 8, we consider two words, $z_1 = \$ |w_1| \sigma$ and $z_2 = \$ |w_2| \sigma$ in $L$, such that $|w_1| > r$, $|w_2| > r$, and none of the symbols of $w_1$ or $w_2$ appear in it twice, as well as none of the symbols of $w_1$, $w_2$ or $\sigma$ appear in the initial assignment of $A$. In addition, let $|w_1| < |w_2|$, and let the prefixes of $w_1$ and $w_2$ of length $r$ be identical and equal to $\sigma_1 \sigma_2 \cdots \sigma_r$ (a fixed word over $\Sigma$).

Let $c_0, c_1, \ldots, c_n$ and $d_0, d_1, \ldots, d_m$ be accepting runs of $A$ on $z_1$ and $z_2$ respectively. Since the two words satisfy the conditions listed in the beginning of the previous section, by Lemmas 10 and 11 there exist instantaneous descriptions

$$c_i^0 = (p_1, u_1, w'_1 \sigma_1, \alpha_1 \alpha'_1), \quad c_i^1 = (q_1, v_1, \epsilon, \alpha'_1),$$
$$d_i^0 = (p_2, u_2, w'_2 \sigma_2, \alpha_2 \alpha'_2) \quad \text{and} \quad d_i^1 = (q_2, v_2, \epsilon, \alpha'_2)$$

with the properties provided by the lemmas.

Let $r'_i = |w_i| - |w'_i|$ - the number of symbols read from $w_i$ at the computation step $i_0^i (i = 1, 2)$, and let $\text{Fixed}(v_i) = \{ j : v_j \in [u_0] \cup \{ \sigma_1, \sigma_2, \ldots, \sigma_r, \$ \} \}$ be the set of indices which contain the fixed symbols at the stage $i_0^i$ of the computation ($v_i = v'_1 v'_2 \cdots v'_r$, $i = 1, 2$).

We note that since $\Sigma$ is infinite and there is no upper bound on the length of the words, there is an infinite number of such pairs $z_1$, $z_2$ that satisfy the conditions mentioned in the first paragraph. On the other hand, there is only finite number of the following parameters ($i = 1, 2$):

- $r'_i$ (because $r'_i \leq r$);
- $p_i$ and $q_i$ (because there is a finite number of states in the automaton $A$);
- $u_i$ and $\alpha_i$ (because at the stage $i_0$ of the computation the registers and the stack may contain only symbols from $[u_0] \cup \{ \sigma_1, \sigma_2, \ldots, \sigma_r, \$ \}$, and $|\alpha_i|$ is bound by a constant independent on $z_i$);
- the first symbol of $\alpha'_i$ (because it belongs to $\{ \sigma_1, \sigma_2, \ldots, \sigma_r \}$);
- $\text{Fixed}(v_i)$ (because there is a finite number of subsets of $\{ 1, 2, \ldots, r \}$);
- $v'_j$, $j \in \text{Fixed}(v_i)$ (follows from the definition of $\text{Fixed}(v_i)$).
Thus, we may choose $z_1$ and $z_2$ in the way that all of the parameters listed above agree, i.e. there exist $z_1$ and $z_2$ as described above, such that

- $r'_1 = r'_2 = r'$;
- $p_1 = p_2 = p$ and $q_1 = q_2 = q$;
- $u_1 = u_2 = u$ and $\alpha_1 = \alpha_2 = \alpha$;
- the first symbol of $\alpha'_1$ equals the first symbol of $\alpha'_2$ – we shall denote it by $\alpha$;
- Fixed($v_1$) = Fixed($v_2$) = $F$;
- $v^j_1 = v^j_2 = v_j$ for $j \in F$.

Let $z_1$ and $z_2$ be chosen in this manner, and let $w_1 = \sigma_1 \sigma_2 \cdots \sigma_r w'_1$ and $w_2 = \sigma_1 \sigma_2 \cdots \sigma_r w'_2$. Also, let $\nu$ be an automorphism of $\Sigma$, that, for $j \notin F$, maps each $v^j_1$ to $v^j_2$ and vice versa, and every other symbol in $\Sigma$ – to itself.

Now, consider the word $z = ||w_2||w_1\sigma$. Since we have chosen $w_1$ and $w_2$ to be of different length, it follows that $z \notin L$. Let us construct a run of $A$ on $z$ in the following manner:

\[d'_0, d'_1, \ldots, d'_{i_0^1-1}, d'_{i_0^1}, c'_{i_1^1+1}, \ldots, c'_{i_1^1-1}, c'_1, d'_{i_1^2+1}, \ldots, d'_{m-1}, d'_m,\]

where

- for $j \leq i_0^1$, $d_j = (q, u, z', w'_2, \sigma, \beta)$ for some $q$, $u$, $z'$ and $\beta$, and $d'_j = (q, u, z'w'_1, \sigma, \beta)$;
- for $j > i_1^2$, $d_j = (q, u, \epsilon, \beta)$ for some $q$, $u$ and $\beta$, and $d'_j = (q, \nu(u), \epsilon, \nu(\beta))$;
- and for $i_0^2 < j \leq i_1^1$, $c_j = (q, u', z', \beta \alpha'_1)$ for some $q$, $u$, $z'$ and $\beta$, and $c'_j = (q, u', z', \beta \alpha'_2)$.

The correctness of these definitions and the fact that this is indeed a run of $A$ on $z$ follows from the choice of $w_1$ and $w_2$ and the choice of instantaneous descriptions $c_{i_0^1}$, $c_{i_1^1}$, $d_{i_0}^1$ and $d_{i_1}^2$, as shown below (expressions (2.1-2.4)). Since $d_m$ is of an accepting sort, it follows that $d'_m$ is also accepting. Thus, we have found an accepting run of $A$ on a word $z \notin L$, which brings us to the desired contradiction.
It remains to show that
\[ d'_{j-1} \vdash d'_j \text{ for } j \leq i_0^2, \]  
(2.1)
\[ c'_j \vdash c'_{j+1} \text{ for } i_0^2 < j \leq i_1^2, \]  
(2.2)
\[ d'_j \vdash d'_{j+1} \text{ for } j > i_1^2, \]  
(2.3)
as well as
\[ d'_{i_0^0} \vdash c'_{i_0^0+1} \text{ and } c'_{i_1^0} \vdash d'_{i_1^0+1}. \]  
(2.4)

Statement (2.1) follows directly from the definition of \(d'_j\)s and the fact that \(d_j \vdash d_j\).

Statement (2.2) follows from the definition of \(c'_j\)s, the properties of \(c_{i_0}\) and \(d_{i_0}\), the fact that \(c_j \vdash c_{j+1}\), and the choice of \(w_1\) and \(w_2\) according to which the first letters of \(\alpha'_1\) and \(\alpha'_2\) are identical (this condition is needed for the case of \(\beta = \epsilon\)).

Statement (2.3) follows directly from Proposition 6.

To show (2.4), note that \(d'_{i_0} = (p, u, w'_1\sigma, \alpha_2')\), and \(c_{i_0} = (p, u, w'_1\sigma, \alpha_1')\). Therefore, \(d'_{i_0} \vdash c'_{i_0+1}\) follows from the fact that \(\alpha \neq \epsilon\). Finally, recall that \(c'_{i_1} = (q, v_1, \epsilon, \alpha'_2)\) and \(d_{i_1} = (q, v_2, \epsilon, \alpha'_2)\). Since \(d_{i_1} \vdash d_{i_1+1}\), by Proposition 6, \(d'_{i_1} \overset{\text{def}}{=} (q, \nu(v_2), \epsilon, \nu(\alpha'_2)) \vdash d'_{i_1+1}\). Since \(\nu(v_2) = v_1\) and \(\nu(\alpha'_2) = \alpha'_2\), \(d'_{i_1} = c'_{i_1} \vdash d'_{i_1+1}\), which completes the proof.
Chapter 3

Finite-memory automata with non-deterministic reassignment and infinite alphabet pushdown automata with deterministic reassignment

In this chapter we show that the class of languages defined by finite-memory automata with non-deterministic reassignment is a subset of the class of languages defined by pushdown automata with deterministic reassignment.

Throughout this chapter, an assignment is a word \( w_1w_2 \cdots w_r \in (\Sigma \cup \{\#\})^* \) such that \( w_i \neq w_j \) for \( i \neq j \), unless \( w_i = w_j = \# \). That is, an assignment is a word over \( \Sigma \cup \{\#\} \), where each symbol from \( \Sigma \) appears at most one time. We denote the set of all assignments of length \( r \) by \( \Sigma^{r\#} \). Note the difference with the assignments of the previous chapter.

3.1 Infinite alphabet pushdown automata with deterministic reassignment, a variation

We define a new model of infinite-alphabet pushdown automata with deterministic reassignment (DR-IPDA\(_n\)), very similar to DR-IPDA defined in the
previous chapter (Definition 4), but with some additional capabilities, which will help us to simulate NR-FMA. These capabilities do not add power to the model, the reasons for it are discussed in Appendix B.

**Definition 12.** A new infinite-alphabet pushdown automaton with deterministic reassignment (DR-IPDA$_n$) is a system $\mathcal{A} = < Q, q_0, \Delta, u, o, \pi, \rho, \mu >$, where

- $Q$ is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\Delta$ is a finite alphabet, such that $\Delta \cap \Sigma = \emptyset$, called the stack alphabet,
- $u = u_1u_2\cdots u_r \in (\Sigma \cup \Delta)^{\#}$ is the initial assignment to the $r$ registers of $\mathcal{A}$,
- $o : Q \to \{1, 2, \ldots, r\} \cup \{\text{nil}\}$ is a function from $Q$ to $\{1, 2, \ldots, r\} \cup \{\text{nil}\}$, called the reset function,
- $\pi, \rho : Q \to \{1, 2, \ldots, r\}$ both are functions from $Q$ to $\{1, 2, \ldots, r\}$ called the stack-based reassignment and the input-based reassignment, respectively,
- $\mu$ is a mapping from $Q \times (\{1, 2, \ldots, r\} \cup \{\epsilon\}) \times \{1, 2, \ldots, r\}$ to finite subsets of $Q \times \{1, 2, \ldots, r\}^*$, called the transition function.

Intuitively, a computation step of the automaton $\mathcal{A}$ in state $q$ is composed of the following actions.

- First, the reset is performed, namely, the $o(q)$th register is reset by replacing its contents with $\#$.

- It is followed by the stack-based reassignment: the content of the $\pi(q)$th register is replaced by a symbol that is currently on the top of the stack, if it does not appear in any other register (unless the symbol is $\#$, in this case it may appear in other registers), otherwise the memory is unchanged.
  - After that, $\mathcal{A}$ may perform $\epsilon$-transition if one is defined by $\mu$, and the computation step is complete.
If no transition was performed, \( A \) proceeds to input-based reassignment, namely, the content of the \( \rho(q) \)th register is replaced by a current input symbol, if it does not appear in any other register, otherwise the memory is left unchanged.

- After that, the automaton may perform a transition, which involves reading from the input, if one is defined by \( \mu \).

An instantaneous description is a member of \( Q \times (\Sigma \cup \Delta)^r_\# \times \Sigma^* \times (\Sigma \cup \Delta \cup \{\#\})^* \). The first component of an instantaneous description is the (current) state of the automaton, the second is the assignment consisting of the contents of the registers (in the increasing order of their indices), the third component is the portion of the input yet to be read, and the last one is the contents of the pushdown store, read top down.

The relation \( \vdash \) (yielding in one step) between two instantaneous descriptions of \( \text{DR-IPDA}_n \) is defined as follows. We write \( (p,w_1w_2\cdots w_r,\sigma \sigma,\tau \tau) \vdash (q,v_1v_2\cdots v_r,\sigma,\tau) \), where \( \sigma \in \Sigma \cup \{\epsilon\} \), \( \tau \in \Sigma \cup \Delta \cup \{\#\} \), if and only if the following holds.

Let \( w_1'w_2'\cdots w_r' \) be defined as follows: \( w_i' = w_i \) for each \( i \neq \pi(p) \), and \( w_{\pi(p)}' = \# \).

- If \( \sigma = \epsilon \):
  - If there is an \( i \neq \pi(p) \) such that \( w_i' = \tau \) and \( w_i' \neq \# \), then \( v_1 = w_1', v_2 = w_2', \ldots, v_r = w_r' \). Otherwise, \( v_{\pi(p)} = \tau \) and \( v_i = w_i' \) for each \( i \neq \pi(p) \).
  - There is a \( j \), such that \( v_j = \tau \) and there is \( (q,k_1k_2\ldots k_l) \in \mu(p,\epsilon,j) \), such that \( \alpha = v_{k_1}v_{k_2}\ldots v_{k_l} \).

- If \( \sigma \neq \epsilon \):
  Let \( u_1, u_2, \ldots, u_r \) be defined as follows. If there is an \( i \neq \pi(p) \) such that \( w_i' = \tau \) and \( w_i' \neq \# \), then \( u_1 = w_1', u_2 = w_2', \ldots, u_r = w_r' \). Otherwise, \( u_{\pi(p)} = \tau \) and \( u_i = w_i' \) for each \( i \neq \pi(p) \). Then:
  - If there is an \( i \neq \rho(p) \) such that \( u_i = \sigma \), then \( v_1 = u_1, v_2 = u_2, \ldots, v_r = u_r \). Otherwise, \( v_{\rho(p)} = \sigma \) and \( v_i = u_i \) for each \( i \neq \rho(p) \).
  - There are \( i, j \), such that \( v_i = \sigma \), \( v_j = \tau \) and there is \( (q,k_1k_2\ldots k_l) \in \mu(p,i,j) \), such that \( \alpha = v_{k_1}v_{k_2}\ldots v_{k_l} \).
The language accepted by DR-IPDA_n \( A \) is defined by
\[
L(A) = \{ \sigma | (q_0, u, \sigma, u_r) \vdash^* (p, v, \epsilon, \epsilon) \},
\]
where \( \vdash^* \) denotes the reflexive and transitive closure of relation \( \vdash \).

The main result of this chapter is stated by the following theorem.

**Theorem 13.** The automata model DR-IPDA_n is stronger than NR-FMA.

The definition of NR-FMA is found in the Introduction. The models are not equivalent, since they are separated by, for example, \( \{ w \$^{we} : w \in \Sigma \setminus \{ \$ \} \} \). (The restriction of this language to a final alphabet yields a classic CFL. Therefore, it may not be accepted by NR-FMA, whose languages, when restricted to finite alphabets, are regular.)

The following sections are devoted to the proof of the fact that DR-IPDA_n is at least as strong as NR-FMA, which completes the proof of the Theorem.

### 3.2 Simulation

In this section we show how NR-FMA can be simulated by DR-IPDA_n.

Let \( A = (Q, q_0, u_0, \mu, \rho, F) \) be an \( r \)-register NR-FMA, \( u_0 = \tilde{u}_1^0 \tilde{u}_2^0 \cdots \tilde{u}_r^0 \).

We may assume without loss of generality, that \( A \) has the following property: for each \( f \in F \), \( q \in Q \) and \( i \in \{1, 2, \ldots, r\} \), \((f, i, q) \notin \mu \). That is, the automaton stops reading from the input once it reaches one of its final states.\(^1\)

We construct an DR-IPDA_n \( A' \), such that \( L(A) = L(A') \), where \( A' \) simulates \( A \) in the following manner.

Each time \( A \) “guesses a symbol into a register \( i \)”, \( A' \) resets the register \( i \), and logs the operation and the resulting contents of \( A \)’s registers on the stack. It also remembers in its state the set \( S \) of \( A \)’s registers that have been reset and have not yet been filled, which corresponds to the set of registers of \( A \), that contain symbols that were guessed but have not yet been used in an input-reading move. Each time \( A \) makes a move that involves reading from the input and comparing it to register \( i \), \( A' \) behaves as follows. If \( i \) contains a symbol from \( \Sigma \), i.e. \( i \notin S \), \( A' \) compares the current input symbol to the content of \( i \). (In such moves, the input symbol is assigned to an auxiliary register, to leave the simulation registers unchanged. If the input

\(^1\)This can be easily shown by introducing an additional state \( f' \), making it the only final state of the automaton, and adding to \( \mu \) an \( \epsilon \)-move from each former final state to \( f \).
reassignment succeeds, it means that the comparison with the $i$th register fails, and the computation halts.) Otherwise, i.e. if $i$ has been reset and contains #, $\mathcal{A}'$ assigns the current input symbol to $i$, removes $i$ from the current $S$, and makes a move by $i$, then logs the operation on the stack. This continues until $\mathcal{A}$ reaches a final state.

In this way $\mathcal{A}'$ simulates the actions of $\mathcal{A}$ by performing a “delayed reassignment” – the symbol guessed by $\mathcal{A}$ is copied into the registers by $\mathcal{A}'$ as soon as it appears in the input.

The only problem that may occur with this course of action is the following. Let $\sigma$ be a symbol guessed by $\mathcal{A}$ into register $i$ at a certain point during the computation, and not guessed by $\mathcal{A}'$, which emptied register $i$ instead. Note, that $\sigma$ can not appear in other registers of $\mathcal{A}$ from this point and until it is erased from the register $i$ by some other symbol. On the other hand, nothing prevents from $\sigma$ to appear in other registers of $\mathcal{A}'$ from this point and until it is actually copied into $i$ from the input. The problem is solved by checking for such situations after the simulation is over by processing the logs kept in the stack. If the check succeeds - the word is accepted by empty stack.

The check is performed as follows. At the end of the simulation the stack contains a sequence of vectors representing the contents of the simulation registers. Each pair of consequent vectors differ by at most one entry, and are separated by the descriptor of the operation that brought to the change, which can be “reset register $i$”, or “copied input into register $i$”. Since the automaton can retrieve this information in the order reverse to the order of computation, at each point it can determine which symbols were copied from the input, and check that neither of these symbols appears in the assignments starting from the moment where the registers they were copied into were reset.

This is implemented in the following manner: during its backwards movement through the logs as it reads the stack from the top down, $\mathcal{A}'$ remembers in its state a set of registers that contain symbols that were copied from the input, but it has not yet reached the point where they had been reset. (Note, that this means, that in the assignments that precede to the copying and come after the reset, these registers contain #.) The set is updated each time $\mathcal{A}'$ reads a descriptor of copy or reset operation.

The use of the set is as follows. Each time $\mathcal{A}'$ loads the next assignment vector from the stack, it skips the registers from the remembered set, i.e. it leaves them unchanged from some previously seen assignment. It means that
these registers continue to contain the symbols which were copied from the input. As the result, if the loaded assignment contains one of these symbols in one of its registers (that are not contained in the set), the loading will not succeed, and the computation will halt, rejecting the input word.

The idea of the simulation is schematically illustrated by Figure 3.1. The formal construction and the proof of its correctness is presented in Appendix B.

Figure 3.1: The simulation of NR-FMA $\mathcal{A}$ by DR-IPDA $\mathcal{A}'$: featuring $\mathcal{A}$ and $\mathcal{A}'$ after steps $s$ and $t$ ($s < t$). (The assignments $w_s$, $w_t$, and $w_l$ are pictured horizontally in the stack merely for the sake of graphical composition.)
Chapter 4

Adding a stack to finite-state unification based automata

In this chapter we generalize the notion of finite-state unification based automata, by adding a push-down store to the model. We also define a version of a context-free grammar over infinite alphabet, which is equivalent to the push-down automaton. These new notions are counterparts of NR-IPDA and infinite-alphabet context-free grammar of [1], NR-IPDA being essentially NR-FMA with a push-down store, and infinite-alphabet context-free grammar being an infinite-alphabet counterpart of context-free grammar equivalent to NR-IPDA.

4.1 FSUBA and its equivalent variations

Before we proceed, we give another definition of FSUBA, which will be more convenient for our generalization purposes. The original definition of FSUBA is present in the introduction.

4.1.1 An alternative definition of finite-state unification based automata

The following alternative definition implements the idea, that we do not have to use the empty register symbol (#) to reset the registers, but rather we may reassign them with the input symbol. This will effectively erase the previous content of the registers, without adding any essential information. The use
of \# remains for the resetting purposes in cases when the input symbol is of “read-only” type and, therefore, cannot be written into the registers that need to be reset.

**Definition 14.** A new finite-state unification based automaton (over $\Sigma$) or, shortly, FSUBA_n, is a system $A = (\Sigma, Q, q_0, F, u, \Theta, \mu)$, where

- $Q$, $q_0 \in Q$ and $F \subseteq Q$ are a finite set of states, the initial state, and the set of final states, respectively.
- $u = u_1u_2\cdots u_r \in (\Sigma \cup \{\#\})^r$, $r \geq 1$, is the initial assignment - register initialization: the symbol in the $i$th register is $u_i$.
- $\Theta \subseteq [u]$ is the “read only” alphabet, whose symbols can not be copies into the registers.
- $\mu \subseteq Q \times 2\{1,2,\ldots,r\} \times \{1,2,\ldots,r\} \times Q$ is the transition relation. The intuitive meaning of $\mu$ is as follows. If the automaton is in state $q$ reading symbol $\sigma$ and there is a transition $(q,S,k,q') \in \mu$, then, if $k \in S$ and $\sigma \notin \Theta$, the automaton enters state $q'$, otherwise, it enters state $q'$ only if the register $k$ contains $\sigma$. In both cases, after entering $q'$ the registers whose indices are in $S$ are either assigned $\sigma$, if $\sigma \notin \Theta$, or $\#$ (are emptied). In other words, if $\sigma \notin \Theta$, then the registers of $S$ are reassigned with $\sigma$, and after that the transition is completed if and only if after the reassignment the $k$th register contains $\sigma$, while in the case when $\sigma \in \Theta$, the register is compared to the input prior to reassignment, and the reassignment is performed with the symbol $\#$ instead of the input symbol.

The definitions of configurations, the run of the automaton, and the language accepted by the automaton are very similar. The only difference is in the definition of $\mu^c$ – the relation induced by $\mu$ on $Q^c \times \Sigma \times Q^c$. Let $q$, $q' \in Q$, $w = w_1w_2\cdots w_r$ and $w' = w'_1w'_2\cdots w'_r$. A triple $((q,w),\sigma,(q',w'))$ belongs to $\mu^c$ if and only if there is a transition $(q,S,k,q')$ in $\mu$, such that the following conditions are satisfied.

- Either $\sigma \notin \Theta$ and $k \in S$ (i.e. the transition register is reassigned with the input symbol), or $w_k = \sigma$.
- $w'_i = w_i$ for $i \notin S$, while for $i \in S$ $w'_i = \sigma$ if $\sigma \notin \Theta$, and $w'_i = \#$, otherwise.
To make the proof of the equivalence of the two definitions easier, we use the following two propositions. Both deal with the “Θ-registers” of the automata, the first regards the new definition, and the second – the original one. From hereon, we call Θ-registers the registers of an automaton (or a grammar), which initially contain symbols from Θ. The propositions state that for each automaton there is an equivalent automaton that never replaces or erases symbols in its Θ-registers.

**Proposition 15.** For any FSUBA \( A \) there is an FSUBA \( A' \), such that \( L(A) = L(A') \), and \( A' \) never empties its Θ-registers.

**Proposition 16.** For any FSUBA\(_n\) \( A \) there is an FSUBA\(_n\) \( A' \), such that \( L(A) = L(A') \), and \( A' \) never reassigns its Θ-registers.

The Proposition 15 is very similar to [5, Lemma 5.1], thus, we omit the proof. The proof of Proposition 16 is presented in Appendix C.

**Proposition 17.** The automata models FSUBA and FSUBA\(_n\) are equivalent.

The proof of Proposition 17 is also found in Appendix C.

### 4.1.2 FSUBA with non-deterministic reassignment

We now define FSUBA with non-deterministic reassignment (NR-FSUBA), and show that it, in fact, possesses the same power as FSUBA. The definition serves to motivate the addition of non-deterministic reassignment, along with the stack, to FSUBA during the definition of unification based infinite alphabet push-down automata (NR-UBPDA) that we define in the next section. Without non-deterministic reassignment, it would be very hard to show that the latter is equivalent to some variant of infinite alphabet context-free grammar.\(^1\) However, in Chapter 5 we define a variation of unification based pushdown automata featuring deterministic reassignment, and show that it is in fact not weaker than NR-UBPDA.

Besides the non-deterministic reassignment, NR-FSUBA features another difference from FSUBA: \(\epsilon\)-moves. Adding \(\epsilon\)-moves to FSUBA is similar to

---

\(^1\)The specific point where the non-deterministic reassignment is needed is during the simulation of a grammar by an automaton, to simulate the reassignment of the grammar and to be able to guess the symbol used in the reassignment, while it does not appear immediately in the input. Putting constraints on the grammars that force the reassignment symbol to appear in the input of the automaton “at the right moment” makes it difficult to simulate an automaton by such a grammar.
adding them to finite automata - nothing can be done during such move but a change of state, thus, we do not examine this variation. On the other hand, the combination of \( \epsilon \)-moves with non-deterministic reassignment allows to perform reassignment without reading any input, which gives a reason to introduce these two capabilities at the same stage. Clearly, in stack automata, \( \epsilon \)-moves are essential to allow reading from the stack without reading from the input.

The NR-FSUBA definition is based on FSUBA\(_n\) model. Note that the new model does not make use of empty register symbol, as it never copies input symbols, particularly those in \( \Theta \), into the registers.

**Definition 18.** A finite-state unification based automaton (over \( \Sigma \)) with non-deterministic reassignment, or, shortly, NR-FSUBA, is a system

\[
\mathcal{A} = (\Sigma, Q, q_0, F, u, \Theta, \mu),
\]

where

- \( Q, q_0 \in Q \) and \( F \subseteq Q \) are a finite set of states, the initial state, and the set of final states, respectively.
- \( u = u_1 u_2 \cdots u_r \in \Sigma^r, r \geq 1 \), is the initial assignment - register initialization: the symbol in the \( i \)th register is \( u_i \).
- \( \Theta \subseteq [u] \) is the "read only" alphabet, whose symbols can not be "guessed."
- \( \mu \subseteq Q \times 2^{\{1,2,\ldots,r\}} \times \{1,2,\ldots,r,\epsilon\} \times Q \) is the transition relation. The intuitive meaning of \( \mu \) is as follows. If the automaton is in state \( q \) and there is a transition \((q,S,k,q') \in \mu\), then, the automaton may reassign the registers whose indices are in \( S \) with an arbitrary symbol form \( \Sigma \setminus \Theta \), and then enter state \( q' \) if and only if \( k = \epsilon \) or the automaton’s \( k \)th register (after the reassignment) contains the symbol currently read in the input.

The definitions of configurations and the language accepted by the automaton are identical to the corresponding definitions given for FSUBA and FSUBA\(_n\). The definition of \( \mu^c \) and the run of the automaton are as follows.

Let \( q, q' \in Q \), \( w = w_1 w_2 \cdots w_r \) and \( w' = w'_1 w'_2 \cdots w'_r \). Then a triple \((q, w, \sigma), (q', w'), \sigma \in \Sigma \cup \{\epsilon\}\), belongs to \( \mu^c \) – the relation induced by \( \mu \) on \( Q^c \times \Sigma \times Q^c \) – if and only if there is a transition \((q, S, k, q') \) in \( \mu \), such that the following conditions are satisfied.
• If $\sigma = \epsilon$, then $k = \epsilon$ and there exists a symbol $\tau \in \Sigma \setminus \Theta$, such that $w'_i = w_i$ for $i \notin S$, while for $i \in S$ $w'_i = \tau$. In this case an $\epsilon$-move is performed, along with reassignment of registers in $S$ with an arbitrary symbol $\tau$.

• Otherwise, $\sigma \in \Sigma$.
  
  – If $\sigma \notin \Theta$ and $k \in S$, then for each $i \in S$ $w'_i = \sigma$, while for each $i \notin S$ $w'_i = w_i$. In this case the registers of $S$, including the transition register, are reassigned with the input symbol.
  
  – Otherwise, $k \notin S$, $w_k = \sigma$, and there exists a symbol $\tau \in \Sigma \setminus \Theta$, such that $w'_i = w_i$ for $i \notin S$, while for $i \in S$ $w'_i = \tau$. In this case the transition register is not reassigned, and the reassignment is performed with an arbitrary symbol $\tau$.

Let $\sigma = \sigma_1 \sigma_2 \cdots \sigma_n$ be a word over $\Sigma$. A run of $A$ on $\sigma$ consists of a sequence of configurations $c_0^1, c_2^1, \ldots, c_{m_0}^1, c_1^1, c_2^1, \ldots, c_{m_1}^1, \ldots, c_1^n, c_2^n, \ldots, c_{m_n}^n$ such that $c_0^i$ is the initial configuration $q_0^i$, for $i = 0, 1, \ldots, n$, $j = 1, 2, \ldots, m_i - 1$, $(c_j^i, \epsilon, c_{j+1}^i) \in \mu^c$, and, for $i = 1, 2, \ldots, n$, $(c_{m_{i-1}}^i, \sigma_i, c_1^i) \in \mu^c$.

**Proposition 19.** For any NR-FSUBA $A$ there is an NR-FSUBA $A'$, such that $L(A) = L(A')$, and $A'$ never reassigns its $\Theta$-registers.

The proof of this proposition is similar to the proof of Proposition 16, and is omitted.

We now claim that NR-FSUBA defines the same class of languages as FSUBA.

**Proposition 20.** The automata models NR-FSUBA and FSUBA are equivalent.

**Proof.** See Appendix C.2. \qed

### 4.2 Main definitions and results

#### 4.2.1 Definition of NR-UBPDA

We now proceed to the main subject of this chapter and define the unification based infinite alphabet pushdown automata by simply adding a push-down store to NR-FSUBA model.
Definition 21. An \( r \)-register unification based infinite alphabet pushdown automaton with non-deterministic reassignment (NR-UBPDA) over infinite alphabet \( \Sigma \) is a system \( \mathcal{A} = \langle Q, q_0, u, \Theta, \rho, \mu \rangle \) where

- \( Q \) is a finite set of states,
- \( q_0 \in Q \) is the initial state,
- \( u = u_1u_2 \ldots u_r \in \Sigma^* \) is the initial assignment to the \( r \) registers of \( \mathcal{A} \),
- \( \Theta \subseteq [u] \) is the “read only” alphabet whose symbols cannot be “guessed” into the registers,
- \( \rho : Q \to 2\{1,2,\ldots,r\} \) is a function from \( Q \) to subsets of \( \{1,2,\ldots,r\} \) called reassignment,
- \( \mu \subseteq Q \times (\{1,2,\ldots,r\} \cup \{\epsilon\}) \times \{1,2,\ldots,r\} \times Q \times \{1,2,\ldots,r\}^* \) is transition relation whose elements are called transitions.

Intuitively, the automaton operates as follows. While in state \( q \) with registers assigned with \( w \), first the reassignment is performed according to reassignment function \( \rho \) by placing an arbitrary symbol from \( \Sigma \setminus \Theta \) (“guessing”) into the registers with indices in \( \rho(q) \), resulting in assignment \( w' = w'_1w'_2 \ldots w'_r \). Following this, the transition is performed. That is, if there is a transition \((q,\epsilon,j,p,k_1k_2\ldots k_n)\in \mu \) and register \( j \) contains the symbol that appears at the top of the stack, then the automaton may, without reading from the input, move into state \( p \) and replace the symbol at the top of the stack with the string \( w'_1w'_2 \ldots w'_n \), read top down. If there is a transition \((q,i,j,p,k_1k_2\ldots k_n)\in \mu , i \neq \epsilon \), register \( i \) contains the next input symbol, and register \( j \) contains the symbol that appears at the top of the stack, then, as before, the automaton may move into state \( p \) and replace the symbol at the top of the stack with the string \( w'_1w'_2 \ldots w'_n \), read top down.

An instantaneous description of the automaton \( \mathcal{A} \) is an element of \( Q \times \Sigma^* \times \Sigma^* \times \Sigma^* \). The first component of an instantaneous description is the (current) state of the automaton, the second is the (current) assignment of the registers, the third component is the portion of the input yet to be read, and the last one is the contents of the push down store, read top down.

Next we define the yielding in one step relation (denoted by \( \vdash \)) between two instantaneous descriptions of the automaton \( \mathcal{A} \) \((p,w_1w_2 \ldots w_r,\sigma\sigma,\gamma\gamma)\) and \((q,v_1v_2 \ldots v_r,\sigma,\alpha\gamma)\), \( \sigma \in \Sigma \cup \{\epsilon\}, \gamma \in \Sigma \). We write

\[(p,w_1w_2 \ldots w_r,\sigma\sigma,\gamma\gamma) \vdash (q,v_1v_2 \ldots v_r,\sigma,\alpha\gamma),\]
if and only if the following holds.

- There is a symbol $\tau \in \Sigma \setminus \Theta$, such that for all $i = 1, 2, \ldots, r$, if $i \in \rho(p)$, then $v_i = \tau$, otherwise, $v_i = w_i$.
- If $\sigma \neq \epsilon$, there is a transition $(p, i, j, q, k_1k_2 \ldots k_r) \in \mu$, such that $v_i = \sigma$, $v_j = \gamma$, and $\alpha = v_{k_1}v_{k_2} \ldots v_{k_n}$.
- If $\sigma = \epsilon$, there is a transition $(p, \epsilon, j, q, k_1k_2 \ldots k_r) \in \mu$, such that $v_j = \gamma$, and $\alpha = v_{k_1}v_{k_2} \ldots v_{k_n}$.

We denote the reflexive and transitive closure of $\vdash$ by $\vdash^*$, and say that $A$ accepts a word $\sigma \in \Sigma^*$ if $(q_0, u, \sigma, u_r) \vdash^* (p, v, \epsilon, \epsilon)$, for some $p \in Q$ and some $v \in \Sigma^r$ (accepting by empty stack). A language $L$ is said to be accepted by the automaton, if for any word $\sigma \in \Sigma^*$, $\sigma \in L$ if and only if $\sigma$ is accepted by the automaton.

### 4.2.2 The definition of unification based infinite alphabet context free grammar

**Definition 22.** An $r$-register unification based infinite alphabet context free grammar (UBCFG) over alphabet $\Sigma$ is a system $G = (V, u, \Theta, R, A_0)$, where

- $V$ is a set of variables, $V \cap \Sigma = \emptyset$.
- $u = u_1u_2 \ldots u_r \in \Sigma^r$ is the initial assignment.
- $\Theta \subseteq \mathbb{N}[u]$ is the “fixed” alphabet whose symbols cannot be “guessed” into the registers,
- $R \subseteq V \times (2^{\{1, 2, \ldots, r\}} \setminus \{\emptyset\}) \times (\{1, 2, \ldots, r\} \cup V)^*$ is a set of productions. For $A \in V$, $S \subseteq \{1, 2, \ldots, r\}$, and $a \in (\{1, 2, \ldots, r\} \cup V)^*$, where $S \neq \emptyset$, the triple $(A, S, a)$ is denoted as $(A, S) \rightarrow a$.
- $A_0 \in V$ is the start symbol.

For $A \in V$, $w = w_1w_2 \ldots w_r \in \Sigma^r$, and $X = X_1X_2 \ldots X_n \in (\Sigma \cup (V \times \Sigma^r))^*$, we write $(A, w) \Rightarrow X$, if there exist a symbol $\sigma \in \Sigma \setminus \Theta$ and a production $(A, S) \rightarrow a \in R$, $a = a_1a_2 \ldots a_n$, such that the conditions below are satisfied.

Let $w' \in \Sigma^r$ be obtained from $w$ by replacing any $w_j$ such that $j \in S$ with $\sigma$. Then for $j = 1, 2, \ldots, n$ the following holds.
• If $a_j = k$ for some $k = 1, 2, \ldots, r$, then $X_j = w'_k$.

• If $a_j = B$ for some $B \in V$, then $X_j = (B, w')$.

For two words $X$ and $Y$ over $\Sigma \cup (V \times \Sigma^*)$, we write $X \Rightarrow Y$ if there exist words $X_1$, $X_2$, and $X_3$ over $\Sigma \cup (V \times \Sigma^*)$, such that $X = X_1(A, w)X_2$, $Y = X_1X_3X_2$, and $(A, w) \Rightarrow X_3$. As usual, the reflexive and transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. The language $L(G)$ generated by $G$ is defined by $L(G) = \{ \sigma \in \Sigma^*: (S, u) \Rightarrow^* \sigma \}$.

**Example 23.** Let $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$. Then, the following $(n + 1)$-register grammar $G$ generates the language $\{ww^R: w \in \Sigma^*\}$:

$$G = (\{A\}, \sigma_1\theta_1\theta_2\cdots\theta_n, \Theta, R, A),$$

where

$$R = \{(A, \{1\}) \rightarrow 1A12A2\cdots|(n + 1).A(n + 1)|\epsilon\}$$

To derive a word $\theta_1\sigma\theta_1$, where $\sigma \notin \Theta$, one must apply subsequently the rules $(A, \{1\}) \rightarrow 2A2$, $(A, \{1\}) \rightarrow 1A1$, and $(A, \{1\}) \rightarrow \epsilon$, obtaining the derivation sequence

$$(A, \sigma_1\theta_1\theta_2\cdots\theta_n) \Rightarrow \theta_1(A, \sigma'\theta_1\theta_2\cdots\theta_n)\theta_1$$
$$\Rightarrow \sigma\theta_1(A, \sigma_1\theta_1\theta_2\cdots\theta_n)\theta_1\sigma$$
$$\Rightarrow \sigma_1\theta_1\sigma.$$

### 4.2.3 The results

In the following sections we will show equivalence between these two models, by proving the following theorems.

**Theorem 24.** For every language generated by an UBCFG there is a NR-UBPDA which accepts that language.

**Theorem 25.** For every language accepted by an NR-UBPDA there is a UBCFG which generates that language.
4.3 From grammars to automata

In this section we show how to build NR-UBPDA equivalent to a given UBCFG, and by that prove Theorem 24. To simplify the construction, we use the following proposition.

**Proposition 26.** For any UBCFG $G$ there is a UBCFG $G'$ that “never guesses into its $\Theta$-registers”, such that $L(G) = L(G')$.

**Proof.** See Appendix C.3

Now, let $G = (V, u, \Theta, R, V_0)$, where $V = \{V_0, V_1, \ldots, V_m\}$ and $u = u_1 u_2 \cdots u_r$, be a UBCFG over $\Sigma$. By Proposition 26, assume $G$ never guesses into its $\Theta$-registers.

We define a NR-UBPDA $A = (Q, q_0, u', \Theta', \rho, \mu)$ over $\Sigma' = \Sigma \cup V$, where $\Theta' = \Theta \cup V$, that will simulate the leftmost derivations of $G$ in the following manner. The variables will be simulated by $V$ - a read-only subset of the alphabet; the elements of $V$ may never appear in a legitimate input word, as will follow from the construction of $A^2$, therefore, $L(A) \subseteq \Sigma^*$. Starting with only a start variable on its stack, every time the automaton finds a variable at the top of the stack, it will substitute it by an $\epsilon$-move with a string derived from that variable according to one of its productions. If the symbol found at the top of the stack is not a variable, the automaton will compare it to the input symbol, and proceed to the next input while “popping” the top of the stack only if they are found equal. Thus, only the words derivable by $G$ is accepted by $A$ by empty stack.

The formal construction of $A$ and the proof of its correctness are presented in Appendix C.3.

---

2Particularly, it is because the input is never compared with the last $m$ registers, which contain elements of $V$, and these elements can not appear in other registers during the computation, for they can not be guessed.

3To avoid extending the input alphabet, we could define an equivalent model of NR-UBPDA, featuring a finite number of symbols for use in the stack in addition to input alphabet, similarly to the variation of DR-IPDA we use in Chapter 3 (Definition 12). For the discussion of why the addition of this feature does not add power to a model, see Appendix B.
4.4 The $nM$-grammars

In this section we define another type of grammar, $nM$-grammar, that will help us to prove Theorem 25 in the manner similar to [1]. We claim that any language generated by $nM$-grammar can also be generated by some UBCFG. This fact is stated in Theorem 28 below, and Appendix C.4 is devoted to its proof. Then we show that for each NR-UBPDA there is an $nM$-grammar that generates the language accepted by the automaton (Section 4.5). Together, these two results yield Theorem 25.

The definition of $nM$-grammar is very similar to the definition of $M$-grammar of [1], the only difference is that the partition vectors used in $M$-grammars are replaced with equality constraint sets (ECS, for short) – the sets of subsets of \(\{1, 2, \ldots, r\}\). These constraints demand that the registers with indices belonging to the same subset contain the same terminal in the assignment for the production to be applied. Each ECS may contain any number of such subsets, which do not have to be disjoint, do not have to cover all the registers, and do not impose any conditions of distinctness on registers’ contents.

**Definition 27.** A unification based infinite alphabet context free grammar with multiple reassignment, or, shortly, $nM$-grammar (standing for “new $M$-grammar”), is a system $G = (V, u, \Theta, \Delta, R, A_0)$, where

- $V$ is a finite set of variables.
- $u = u_1 u_2 \cdots u_r \in \Sigma^r$ is the initial assignment. Note that unlike the $M$-grammar in [1], the definition of $nM$-grammar does not differ in this aspect from our main grammar definition.
- $\Theta \subseteq [u]$ is the “fixed” alphabet whose symbols cannot be “guessed”.
- $\Delta$ is a finite non-empty alphabet.
- $R \subseteq (V \times P_r) \times (\{1, 2, \ldots, r\} \cup \Delta \cup (V \times (\{1, 2, \ldots, r\} \cup \Delta)^r)^*)$ is a set of productions, where $P_r$ is the set of all possible equality constraint sets of the form $p = \{p_1, p_2, \ldots, p_m\}$, $p_i \subseteq \{1, 2, \ldots, r\}$, $i = 1, 2, \ldots, m$, i.e., $P_r = 2^{\{1, 2, \ldots, r\}}$. For $A \in V$, $p \in P_r$, and $a \in (\{1, 2, \ldots, r\} \cup \Delta \cup (V \times (\{1, 2, \ldots, r\} \cup \Delta)^r)^*)$ we write the triple $(A, p, a)$ as $(A, p) \rightarrow a$.
- $A_0 \in V$ is the start symbol.
Let \( w = w_2w_2 \cdots w_r \in \Sigma^r \) and \( p = \{p_1, p_2, \ldots, p_m\} \in P_r \). We say that the assignment \( w \) \textit{conforms to} the ECS \( p \), if for all \( i = 1, 2, \ldots, m \), for all \( j, k \in p_i \), \( w_j = w_k \). For example, if an ECS contains only one-element constraints, like \( \{\{1\}, \{5\}\} \), any assignment conforms to it. On the other hand, the only form of assignment that conforms to \( \{\{1, 2, \ldots, r\}\} \) is \( \sigma \sigma \cdots \sigma \).

We say that two ECSs \( p_1 \) and \( p_2 \) are \textit{equivalent}, if for any assignment \( w \), \( w \) conforms to \( p_1 \) if and only if \( w \) conforms to \( p_2 \). Clearly, replacing two overlapping constraints in ECS with their union and eliminating empty and one-element constraints, yields equivalent ECS. For example, the ECSs \( \{\{1, 3, 5\}, \{3, 7\}, \{1\}\} \) and \( \{\{1, 3, 5, 7\}, \emptyset\} \) are equivalent.

For a function \( f : \Delta \rightarrow \Sigma \setminus \Theta \), \( A \in V \), \( w = w_1w_2 \cdots w_r \in \Sigma^r \), and \( X = X_1X_2 \cdots X_n \in (\Sigma \cup (V \times \Sigma^*))^r \), we write \((A, w) \xrightarrow{f} X\) if there exists a production \((A, p) \rightarrow a \in R\), \( a = a_1a_2 \cdots a_n\), such that the conditions below are satisfied.

- \( w \) conforms to \( p \).
- If \( a_i = j \in \{1, 2, \ldots, r\} \), then \( X_i = w_j \).
- If \( a_i = \delta \in \Delta \), then \( X_i = f(\delta) \).
- If \( a_i = (B, b_1b_2 \cdots b_r) \in V \times (\{1, 2, \ldots, r\} \cup \Delta)^r \), then \( X_i = (B, v_1v_2 \cdots v_r) \), where \( v_k, k = 1, 2, \ldots, r \), is defined as follows. If \( b_k = j \in \{1, 2, \ldots, r\} \), then \( v_k = w_j \). If \( b_k = \delta \in \Delta \), then \( v_k = f(\delta) \).

For two words \( X \) and \( Y \) over \( \Sigma \cup (V \times \Sigma^*) \), we write \( X \Rightarrow Y \) if there exist words \( X_1 \), \( X_2 \) and \( X_3 \) over \( \Sigma \cup (V \times \Sigma^*) \), and \((A, w) \in V \times \Sigma^r \) such that \( X = X_1(A, w)X_2 \), \( Y = X_1X_3X_2 \), and for some function \( f : \Delta \rightarrow \Sigma \setminus \Theta \), \((A, w) \xrightarrow{f} X_3\). As usual, the reflexive and transitive closure of \( \Rightarrow \) is denoted by \( \Rightarrow^* \). The language \( L(G) \) generated by \( G \) is defined by \( L(G) = \{\sigma \in \Sigma^* : (A_0, u) \Rightarrow^* \sigma\} \), and is referred to as an \( nM \)-language.

**Theorem 28.** Every \( nM \)-language is generated by UBCFG.

The proof of the theorem is presented in Appendix C.4.

### 4.5 From automata to grammars

In this section we show how to simulate NR-UBPDA by \( nM \)-grammars, which together with Theorem 28 will imply Theorem 25. An immediate
corollary to this result is the equivalence of UBCFG and nM-grammars. The
construction below is very similar to that in the proof of the corresponding
result in [1].

Let \( A = \langle Q, q_0, \mathit{u}, \Theta, \rho, \mu \rangle, \mathit{u} = u_1u_2 \cdots u_r \) be an \( r \)-register NR-
UBPDA. Assume \( A \) “never guesses into it’s \( \Theta \)-registers”. That is, if \( u_i \in \Theta \),
then for all \( q \in Q \) \( i \notin \rho(q) \). This assumption can be made without loss of
generality, since it can be shown that for any \( A \) there is an \( A' \) with this
property, that accepts the same language. The proof is similar to the proof
of the corresponding property of UBCFG. Also, to simplify notation, assume
without loss of generality, that for some \( r' \in \{0, 1, \ldots, r\} \), \( u_i \in \Theta \) if and only
if \( i > r' \). That is, \( \mathit{u} = \mathit{u}'\mathit{u}'' \), where \( \mathit{u}' \in (\Sigma \setminus \Theta)^{r'} \), and \( \mathit{u}'' \in \Theta^{r-r'} \). Thus, for
all \( q \in Q \) \( \rho(q) \subseteq \{1, 2, \ldots, r'\} \), and the last \((r - r')\) registers are fixed during
the computation.

Consider a \((2r + 1)\)-register nM-grammar \( G = (V, \mathit{u}\mathit{u}_r, \mathit{u}, \Theta, \Delta, R, A_0) \)
defined below.

\[ V = Q \times Q \cup \{A_0\}, \] where \( A_0 \) is the new symbol. The intuitive mean-
ing of variable \((s,t)\) is as follows: \((s,t), \mathit{v}\sigma\mathit{w} \Rightarrow G^* \sigma \in \Sigma^* \) if and only if
\((s, \mathit{v}, \sigma, \sigma) \xrightarrow{G}^* (t, \mathit{w}, \epsilon, \epsilon) \). That is, \((s,t), \mathit{v}\sigma\mathit{w}) \) derives the portion \( \sigma \) of the
input word that must be read between a point in time when \( A \) is in the
configuration \((s, \mathit{v})\) with \( \sigma \) on the top of the stack and a point in time when
\( A \) removes that \( \sigma \) from the stack and enters the configuration \((t, \mathit{w})\).

The alphabet \( \Delta \) is defined by stating its size, which we set to be \( r \cdot (N+1), \)
where

\[ N = \max\{|n|(q_1, j_1j_2 \cdots j_n) \in \mu(q, k, i), q, q_1 \in Q; k, i = 1, 2, \ldots, r\}, \]

i.e. \( N \) is the maximum number of symbols pushed into the stack during one
step of the automaton.

\( R \) contains the following three groups of productions.

\[ (A_0, \emptyset) \rightarrow ((q_0, q), 12 \cdots r(r + 1)\delta_1\delta_2 \cdots \delta_r\cdot \mathit{c}), \] for each \( q \in Q \), where \( \mathit{c} \)
here and below denotes the (possibly empty) string \((r' + 1)(r' + 2) \cdots r\).

A production in this group “says”, essentially, that the goal is to pass
from configuration \((q_0, \mathit{u})\) with \( \mathit{u} \) in the stack to the configuration

\[ (q, f(\delta_1)f(\delta_2) \cdots f(\delta_r)\mathit{u}_2) \]

with the empty stack.
• For
  - each \((q, k, i) \in Q \times \{1, 2, \ldots, r\} \cup \{\epsilon\} \times \{1, 2, \ldots, r\}\),
  - each \((q_1, j_1, j_2 \cdots j_n) \in \mu(q, k, i), n > 0\),
  - each \(q_2, \ldots, q_n, q_{n+1} \in Q\) and
  - each \(\delta_2, \delta_3, \ldots, \delta_n \in \Delta''\),

\(R\) contain the production

\[
(q, q_{n+1}, p) \rightarrow a((q_1, q_2), b_{a_1} \delta_2 c) \cdots ((q, q_{l+1}), \delta_l c a_l \delta_{l+1} c) \cdots ((q_n, q_{n+1}), \delta_n c a_n (r + 2) \cdots (2r + 1)),
\]

where \(p, b, a\) and \(a_i\)'s, \(l = 1, 2, \ldots, n\), are defined as follows.

- \(p = \{\{i, r + 1\}\}, \) if \(i \notin \rho(q)\), and \(p = \emptyset\), otherwise.
  This guarantees that in cases when the symbol compared to the stack head is not guessed, the transition is only possible if the assignment is such that the register simulating the stack head and the register that simulates the \(i\)'th register indeed contain the same symbol.
- \(b = b_1 b_2 \cdots b_r\), where \(b_l = l\) for \(l \notin \rho(q)\), and for \(l \in \rho(q)\), \(b_l = \delta_0\), if \(i \notin \rho(q)\), and \(b_l = r + 1\), otherwise.
  Since the \(b\) part of the derived assignment simulates the assignment of \(\mathcal{A}\) after the transition, the registers that are not reassigned are unchanged \((b_l = l)\), and the registers that must be reassigned are either guessed \((b_l = \delta_0)\), or, in case \(i \in \rho(q)\), generated from \((r + 1)\)st register, that simulates the stack head.
- \(a = b_k\) if \(k \neq \epsilon\), and \(a = \epsilon\) otherwise, whereas \(a_l = b_{j_l}, l = 1, 2, \ldots, n\).
  Again, since the \(b\) part of the derived assignment is responsible for simulating the assignment of \(\mathcal{A}\) after the transition, the symbol \(a\), generated by the grammar and simulating the symbol read from the input, must be the same as \(b_k\) – the symbol in the \(k\)-th register of the automaton after the transition. That is, unless \(k = \epsilon\), in which case no symbol is read from the input, and no symbol is generated. Similarly, \(a_l\) simulates the \(l\)th symbol that is pushed onto stack, that is the symbol in \(j_l\)-th register of the automaton after the transition, and, therefore, is \(b_{j_l}\).
According to the above intuitive remarks, this production states that instead of removing the \(i\)th register symbol from the stack we have to remove the symbols which appear in the \(j_1\)th, \(j_2\)th, \ldots, \(j_n\)th registers, (after the reassignment of the registers in \(\rho(q)\)).

- For each \((q,k,i) \in Q \times (\{1,2,\ldots,r\} \cup \\{\epsilon\}) \times \{1,2,\ldots,r\}\) and each \((q_1,\epsilon) \in \mu(q,k,i)\), \(R\) contains the production

\[
((q,q_1),p) \rightarrow a,
\]

where \(p\) and \(a\) are defined as follows.

- As in the case \(n > 0\), \(p\) contains the set \(\{i,r+1\}\), if and only if \(i \notin \rho(q)\). Otherwise, it contains the set \(\{r+1,r+1+i\}\). In addition, \(p\) contains the following constraints:
  
  1. \(\{l,l+r+1\}\) for each \(l \notin \rho(q)\);
  2. \(\{l+r+1\}|l \in \rho(q)\};

  This is the case when the transition “pops” a symbol from the stack. Therefore, the assignment to the left hand side of the derivation must simulate two consequent assignments of the automaton: the one before, and the one after the transition. The added constraints state, that all the reassigned registers must contain the same symbol after the transition, implying (2), and the rest of the registers must remain the same after the transition, implying (1). Besides, if \(i \in \rho(q)\), then the \(i\)th register must equal to the stack head, which is reflected by the constraint \(\{r+1,r+1+i\}\).

- \(a = r+1+k\) if \(k \neq \epsilon\), and \(a = \epsilon\) otherwise.

Since the assignment after the transition in this case is simulated by the last \(r\) registers of the grammar before the derivation is applied, the grammar produces the symbol simulating the input from the \(k\)th register of these last \(r\) registers, unless \(k = \epsilon\), in which case no symbol is generated.

According to the above intuitive remarks, this production states that the \(i\)th register symbol is “popped” from the stack.

It immediately follows from the definition of \(G\), that

\[
(q,v,\sigma,\tau) \vdash (q_1,v',\epsilon,v_1'v_2'\cdots v_n'),
\]

44
\( \sigma \in \Sigma \cup \{\epsilon\}, \ n > 0, \) if and only if for any \( q_2, q_3, \ldots, q_{n+1} \in Q, \) and any \( w_2, w_3, \ldots, w_{n+1} \in \Sigma^r, \)

\[
((q, q_{n+1}), v_{\tau} w_{n+1}) \Rightarrow \sigma((q_1, q_2), v'_1 w_2)((q_2, q_3), w_2 v'_2 w_3) \cdots \cdots ((q_{n-1}, q_n), w_{n-1} v'_{n-1} w_n)((q_n, q_{n+1}), w_n v'_{n} w_{n+1}),
\]

and

\[
(q, v, \sigma, \tau) \vdash (q_1, v', \epsilon, \epsilon),
\]

\( \sigma \in \Sigma \cup \{\epsilon\}, \) if and only if

\[
((q, q_1), v_{\tau} v') \Rightarrow \sigma.
\]

Now the proof is exactly as that of [2, Theorem 5.4, pp. 116-119] and will be omitted.

\footnote{To apply the proof of [2, Theorem 5.4, pp. 116-119] to our models we view \( G \) as a “context-free grammar with infinitely many terminals \( \Sigma \) and variables \( V \times \Sigma^{2r+1} \) and \( A \) as “pushdown automaton with infinitely many states \( Q \times \Sigma^r \) and infinite input and stack alphabets \( \Sigma \)”. Then we just rewrite the “variable” \( ((q, p), v_{\tau} w) \) of \( G \) as \( [(q, v), \tau, (p, w)] \), where \( (q, v) \) and \( (p, w) \) are the “states” of \( A \). Note that the proof in [2] does not use the finiteness conditions on the grammar or the automaton.}
Chapter 5

Unification based pushdown automata with deterministic reassignment

In this chapter we define a variation of unification based pushdown automata with deterministic reassignment and show its equivalence to NR-UBPDA, defined in Chapter 4.

5.1 Definitions

We start with the changes to the model of NR-UBPDA.

We define new unification base push-down automata with non-deterministic reassignment, NR-UBPDA\_n – a model very similar to NR-UBPDA, with the only difference: the reassignment function is defined as

\[ \rho : Q \rightarrow \{1, 2, \ldots, r\} \cup \{\text{nil}\}, \]

i.e. its values are not sets of register indices, but single registers. This variation to the model does not change its power. Indeed, any NR-UBPDA\_n can be viewed as NR-UBPDA whose values of \( \rho(q) \) contain exactly one element or are empty, and any NR-UBPDA may be simulated by NR-UBPDA\_n by using \( \epsilon \)-moves to reassign all the registers in \( \rho(q) \) one by one, while – in cases \( |\rho(q)| > 1 \) – making sure the symbol guessed into each of them is the same.

Now, we proceed to the main definition of this chapter.
**Definition 29.** An \(r\)-register unification based infinite alphabet pushdown automaton with deterministic reassignment (DR-UBPDA) over infinite alphabet \(\Sigma\) is a system \(\mathcal{A} = \langle Q, q_0, \Delta, u, \Theta, \pi, \rho, \mu \rangle\) where

- \(Q\) is a finite set of states,
- \(q_0 \in Q\) is the initial state,
- \(\Delta\) is a finite stack alphabet,
- \(u = u_1 u_2 \cdots u_r \in (\Sigma \cup \Delta \cup \{\#\})^r\) is the initial assignment to the \(r\) registers of \(\mathcal{A}\),
- \(\Theta \subset [u] \cap \Sigma\) is the “read only” alphabet whose symbols cannot be copied into the registers,
- \(\pi, \rho : Q \to \{1, 2, \ldots, r\}\) are both function from \(Q\) to \(\{1, 2, \ldots, r\}\) called stack-based reassignment and input-based reassignment, respectively.
- \(\mu \subseteq (Q \times (\{1, 2, \ldots, r\} \cup \{\epsilon\}) \times (\{1, 2, \ldots, r\}) \times (Q \times \{1, 2, \ldots, r\}^*)\) is the finite transition relation whose elements are called transitions.

Intuitively, the automaton operates as follows.

- While in state \(q\) with registers assigned with \(w\), first, the stack-based reassignment is performed, namely, the symbol that currently appears on the top of the stack is placed into the \(\pi(q)\)th register, unless it is a symbol from \(\Theta\). In the latter case, the \(\pi(q)\)th register is erased by assigning it with \# . The resulting assignment is \(w' = w_1' w_2' \cdots w_r'\).

  - After that, \(\mathcal{A}\) may perform \(\epsilon\)-transition if defined. That is, if there is a transition \((q, \epsilon, j, p, k_1 k_2 \cdots k_n) \in \mu\) and \(w_j'\) equals the top of the stack symbol, then the automaton may, without reading from the input, move into state \(p\) and replace the symbol at the top of the stack with the string \(w_{k_1}' w_{k_2}' \cdots w_{k_n}'\), read top down, and the computation step is complete.

- If no \(\epsilon\)-transition was performed, \(\mathcal{A}\) proceeds to input-based reassignment, namely, the current input symbol is placed into the \(\rho(q)\)th register, unless it is a symbol from \(\Theta\). In the latter case, the \(\rho(q)\)th register is erased by placing \# into it. The resulting assignment is \(w'' = w_1'' w_2'' \cdots w_r''\).
After that, the automaton may perform a transition, which involves reading from the input, if defined. That is, if there is a transition \((q, i, j, p, k_1 k_2 \ldots k_n) \in \mu, i \neq \epsilon, w_i'' = \text{the input symbol, and } w_j'' = \text{the top of the stack symbol, then, as before, the automaton may move into state } p \text{ and replace the symbol at the top of the stack with the string } w_k' w_k' \ldots w_k', \text{ read top down.}

An instantaneous description of the automaton \(A\) is defined exactly as that of NR-UBPDA.

The yielding in one step relation (denoted by \(\vdash\)) between two instantaneous descriptions of the automaton \(A(p, w, \sigma, \gamma)\) and \((q, v, \sigma, \alpha \gamma)\), \(\sigma \in \Sigma \cup \{\epsilon\}, \gamma \in \Sigma \cup \Delta \cup \{\#\}, w = w_1 w_2 \ldots w_r, v = v_1 v_2 \ldots v_r\), is defined as follows. We write \((p, w, \sigma, \gamma) \vdash (q, w, \sigma, \alpha \gamma)\), if and only if the following holds.

Let \(w' = w_1' w_2' \ldots w_r'\) be defined by \(w_i' = w_i\) for \(l \neq \pi(q)\), and for \(l = \pi(q)\) \(w_i' = \gamma\) if \(\gamma \notin \Theta\), otherwise, \(w_i' = \#\).

- If \(\sigma = \epsilon\), then \(v = w'\) and there is a transition \((p, \epsilon, j, q, k_1 k_2 \ldots k_n) \in \mu, \text{ such that } v_j = \gamma, \text{ and } \alpha = v_{k_1} v_{k_2} \ldots v_{k_n}\).
- Otherwise, \(v\) is defined by \(v_l = w_l'\) for \(l \neq \rho(q)\), for \(l = \rho(q)\) \(v_l = \sigma\) if \(\sigma \notin \Theta\), otherwise, \(v_l = \#\), and there is a transition \((p, i, j, q, k_1 k_2 \ldots k_r) \in \mu, \text{ such that } v_i = \sigma, v_j = \gamma, \text{ and } \alpha = v_{k_1} v_{k_2} \ldots v_{k_n}\).

The definition of \(\vdash^*\), as well as the definition of acceptance of a word and a language, are the same as for NR-UBPDA.

5.2 Equivalence

First, let us see why DR-UBPDA is not stronger than NR-UBPDA

Note, that in addition to deterministic reassignment, DR-UBPDA features two more differences from NR-UBPDA:

- the use of the empty register symbol \(\#\) (it is necessary to perform the reassignment, since the reassignment is performed by copying letters into registers, and letters in \(\Theta\) cannot be copied),

- and the use of the additional stack alphabet \(\Delta\).
None of these differences strengthen the model, since each of them may be simulated by NR-UBPDA$^n$. We discuss the way to simulate the additional symbols used in the stack, though in a different context, on page 61. The idea of simulating deterministic reassignment with non-deterministic is discussed on page 17. Even though the context is different, and the assignments model is different (the discussion refers to assignments in $\Sigma^\epsilon$, whereas here we deal with assignments in $\Sigma^*$), the general idea is the same.

Simulating NR-UBPDA$^n$ with DR-UBPDA is much more complicated, as it involves the simulation of “guessing” with deterministic reassignment. The idea is the same as in simulating NR-FMA with DR-IPDA$^n$ (see Chapter 3): the reassignment is “delayed” until the moment when the reassigned register is compared to some known symbol. When the moment arrives, deterministic reassignment is used to copy the symbol into the register. The simulating automaton needs to remember (by a state) the list of registers that were reassigned but whose contents have not yet been determined. We denote this list by $U$, for “unknown”.

The same idea is utilized in showing that NR-FSUBA is not stronger than FSUBA$^n$. Note, that in both cases, the “guessing” automatons were of a finite-state variety, that is, they did not have a pushdown store.

However, in our case there is a problem with this approach. The symbol “guessed” into a register may be pushed into the stack before it is compared to a known symbol, i.e., before the simulation had a chance to determine it using deterministic reassignment. The simulation “has no means to know” what symbol to push into the stack.

The solution to this problem is as follows. Since the content of the reassigned register $i$ is unknown, DR-UBPDA pushes into the stack “a variable” $x_i$ that denotes it. Here is where the additional stack alphabet is useful – “the variables” are symbols that have to be different from any input symbol. In general, from here on we assume that DR-UBPDA can save any useful auxiliary information in the stack using its stack alphabet.

When eventually the value of $x_i$ is determined to be $\sigma$, (after register $i$ has been compared to some known symbol), DR-UBPDA has to treat $x_i$ as $\sigma$. Therefore, it needs a “translation table”, where it may keep the values for the variables that have been discovered. Clearly, another set of $r$ registers can be used for that, and a list of such variables (or their indices) is remembered in the state. We denote this list by $D$, for “discovered”. (Recall that we already have a set of register indices $U$ attached to the state.)

However, the value of $x_i$ may change during the computation, as register
NR-UBPDA: \[(\alpha, \beta, \tau, \sigma \in \Sigma)\]

“guessed” \(\alpha\) into register \(i\),
that previously contained \(\beta\)

DR-UBPDA:

register \(i\) remains unchanged
Before: \(i \in D \setminus U\)
After: \(i \in U \setminus D\)

\[\begin{array}{c|c}
\hline
i & \text{simulation registers} \\
\hline
\alpha & \text{translation table} \\
\tau & \vdots \\
\sigma & x_i \\
\hline
\end{array}\]

\[\begin{array}{c|c}
\hline
i & r + i \\
\hline
\beta & \sigma \\
\tau & \vdots \\
x_i & \\
\hline
\end{array}\]

Figure 5.1: NR-UBPDA and DR-UBPDA after “guessing” a symbol into a register, and pushing the result into the stack.

\(i\) may be reassigned again. Thus, DR-UBPDA needs to register this event. It does so by writing “Reset \(i\)” on the stack each time it occurs. At the same time, the entry in the “translation table” has to be devalidated by removing \(i\) from \(D\), as it is no longer relevant for the latest “incarnation” of \(x_i\). Indeed, the “translation” of \(x_i\) is valid only for variables that appear in the stack from the top down to the first “Reset \(i\)” symbol.

However, the previous value of \(x_i\) has to be saved, as it will be needed when the stack pointer will reach the previous “incarnation” of the variable. Therefore, if \(i\) was in \(D\) at the moment of the reset, just under “Reset \(i\)” DR-UBPDA writes “\(T(i) = \sigma\),” meaning that the previous “translation” of \(x_i\) was \(\sigma\). This operation is illustrated by Figure 5.1. When the stack pointer eventually reaches this record, DR-UBPDA will revalidate its current entry for \(x_i\) and load the value into the table.

Note, that this ensures that exactly \(r\) additional registers suffice for DR-UBPDA to keep track of the values of register variables in the stack.

What happens when DR-UBPDA encounters \(x_i\) on the stack? If \(i \in D\) – it replaces it with the value from the “translation table”, and proceeds as
NR-UBPDA$_n$. But what if $i \notin D$?

If the top of the stack needs to be compared to a register $j$ with a known content $\sigma (j \notin U)$, it is, in fact, another way to discover the value of $x_i$, and DR-UBPDA updates its “translation table” and adds $i$ to $D$.

If the top of the stack needs to be compared to a register $j \in U$, since both of the values are unknown, i.e. can be any value, DR-UBPDA may assume that the comparison succeeded. But it also has to assume from that point on that the value of $x_i$ and the contents of register $j$ are the same. For this purpose it keeps another piece of data in its state, $E$. (Recall that we already have two subsets of $\{1, 2, \ldots, r\} - U$ and $D$.) As will become clear further, the best data structure for $E$ is equivalence relation over $\{x_i | i \notin D\} \cup \{y_j | j \in U\}$. Here, each $x_i$ represents the (undiscovered) value of the variable $x_i$ found on the stack from the top down to the first “Reset $i$,” and each $y_j$ represent the (unknown) current content of the register $j$.

In addition, there are cases when discovering the value of $x_i$ means also discovering the contents of some register(s), or even the values of other variables. For example, such a situation occurs if it was previously assumed (and reflected in $E$) that the value of $x_i$ equals the content of a register $j \in U$, as was described in the previous paragraph. Then discovering the value of $x_i$ means discovering the content of register $j$, and, perhaps, the values of other variables that were assumed to be equal to the contents of the $j$th register. Another such situation is when $x_i$ refers to the register $i \in U$ that has not been reassigned since its content was pushed into the stack (this fact has to be reflected in $E$ also by adding $(x_i, y_i)$ to it). Then along with the value of $x_i$, the content of register $i$ is discovered.

DR-UBPDA takes care of this problem in the following manner. It goes over $E$, first fetching any register numbers whose content is assumed to be equal to the value of $x_i$, and updating the registers according to the newly-discovered value of $x_i$. Then it goes over $E$ to fetch the indices of variables whose values are assumed equal to the value of $x_i$, and updates the “translation table”. (Naturally, all this is done by means of stack-based reassignments during $\epsilon$-moves.) After this process is complete, $U$, $D$ and $E$ are updated accordingly.

When does $E$ need to be updated?

As mentioned before, every time register $i$ is “reset,” we need to reflect this fact on the stack by writing “Reset $i$,” to indicate that the next variable $x_i$ placed on the stack has a different value. But writing auxiliary data on the stack is meaningless while simulating a “pop” move, i.e. replacing
the topmost symbol with $\epsilon$. In this case, DR-UBPDA remembers the fact that register $i$ was reset and does not correspond to $x_i$ found on the stack by removing $(x_i, y_i)$ from $E$. On the other hand, if $(x_i, y_i) \notin E$, and DR-UBPDA simulates a move that writes a non-empty string to the stack, it writes “Reset $i$” to the stack before writing the string, and adds $(x_i, y_i)$ back to $E$.

In addition, all the “old” pairs containing $y_i$ are removed from $E$ each time register $i$ is “reset,” regardless of whether the “reset” is reflected in the stack or not. Also, each time DR-UBPDA reads “Reset $i$” from the stack, it removes all the pairs containing $x_i$ from $E$, (as well as removes $i$ from $D$). These are the “irreversible” changes done to $E$, that are performed in cases when the previous state of $E$ must not be saved.

There are, however, changes that demand saving the previous state. That is, when “Reset $i$” is written to the stack, and, naturally, all the pairs containing $x_i$ have to be removed from $E$, as they refer to the value of the “previous incarnation” of $x_i$. On the other hand, when the stack pointer reaches that “incarnation” again, some of these equivalences will become relevant again. Thus, they are saved in the stack before the “Reset $i$” is written as a record of the form “$x_i$ equals to (a list of equivalences, not containing $x_i$ and $y_i$)”. (Implementation-wise, such record looks as follows: “$x_i$ equals” is a symbol of $\Delta$, placed on top of the list of variables (also symbols of $\Delta$), terminated on the bottom with $\$.)

Yet, there remains a question which of these equivalences are relevant when the stack pointer reaches them. To answer this question, we note the difference between two types of elements that may appear in the list of equivalences: $x_j$ and $y_j$.

$x_j$ refers to the value of $x_j$ found in the stack from the point of the equivalence’s record down to the first “Reset $j$”. Therefore, this equivalence is always relevant, since DR-UBPDA has the up-to-date information about that value of $x_j$ (or lack of thereof) at the point when the stack pointer reaches the equivalence record.

On the other hand, $y_j$ refers to the current content of register $j$, which is relevant only until the register is reset. Thus, we introduce an additional indicator attached to each state of DR-UBPDA: a subsets of $\{1, 2, \ldots, r\}$, denoted by $R$ (for “relevant”). (Recall, that we already had, $U$, $D$ and $E$.) Each time a record “$x_i$ equals to (a list)” is read from the stack, and the list contains $y_j$, the equivalence to $y_j$ is only relevant if $j \in R$.

The set $R$ is updated as follows. Each time register $i$ is “reset,” regardless
Figure 5.2: Register layout of simulating DR-UBPDA. There are 7 $r$-register blocks. The first two blocks are “variable” (contain symbols from $\Sigma$ or $\#$ and are subject to reassignment). Blocks 3-7 are “constant” (contain symbols from $\Delta$, used to write auxiliary records into the stack).

of whether the “reset” is indicated on the stack, and each time “Reset $i$” is read, $i$ is removed from $R$. Each time a list of equivalences containing $y_i$ is written to the stack (note, that it can not be written in the context with the reset of register $i$, as equivalences list of $x_i$ may not contain $y_i$), $i$ is added to $R$.

Now, whenever DR-UBPDA reads the record “$x_i$ equals to (a list)” from the stack, it updates $E$ by adding $(x_i, x_j)$ for each $x_j$ in the list, and $(x_i, y_j)$ for each relevant $y_j$ in the list.

The register layout of the simulating automaton is shown on Figure 5.2.

We note, that throughout the simulation DR-UBPDA only has access to the topmost symbol of the stack. Any reference to the contents of the stack below the “stack pointer” is for explanation only.
Appendix A

Auxiliary statements for Chapter 2

Proposition 9. For each \( c_i = (q, u, z', \alpha) \) such that \( z' \neq \epsilon \) in the above run, \( [w] \subseteq [uz'] \).

Proof. Suppose \( [w] \not\subseteq [uz'] \), let \( \sigma' \in [w] \) be a symbol not belonging to \( [uz'] \). Since \( z' \) is not empty, it follows that \( z' = z'' \sigma \). Thus, since \( c_0 \vdash^* c_i \), it follows that \( (q_0, u_0, S^{[w]} u, u_0^0) \vdash^* (q, u, z'', \alpha) \). Let automorphism \( \tau \) map \( \sigma \) to \( \sigma' \), \( \sigma' \) to \( \sigma \), and every other symbol of \( \Sigma \) to itself. Then, by Proposition 6, \( (q_0, \tau(u_0), \tau(S^{[w]} u), \tau(u_0^0)) \vdash^* (q, \tau(u), \tau(z''), \tau(\alpha)) \). Since \( \sigma \) does not appear in these instantaneous descriptions (as follows from the choice of \( z \) and Proposition 5), and \( \sigma' \) only appears in \( w \), by the choice of \( \sigma' \); by application of \( \tau \) we get \( (q_0, u_0, S^{[w]} \tau(u), u_0^0) \vdash^* (q, u, z'', \alpha) \), where \( \tau(w) \) is obtained by replacing \( \sigma' \) with \( \sigma \) in \( w \). Thus, by appending \( \sigma \) to the end of the input word, we get

\[
(q_0, u_0, S^{[\tau(w)]} \tau(w) \sigma, u_0^0) \vdash^* (q, u, z'', \alpha) = c_i \vdash^* c_n.
\]

The first instantaneous description in the last statement is, in fact, the initial instantaneous description of \( A \) on the word \( S^{[\tau(w)]} \tau(w) \sigma \notin L \), and since \( c_n \) is an accepting instantaneous description, it contradicts with \( L = L(A) \), which completes the proof.

Lemma 10. There is an instantaneous description \( c_{i_0} = (p, u, w' \sigma, \alpha \alpha') \), such that the following holds:

1. \( w' \) is a suffix of \( w \) and \( |w| - |w'| \leq r \),

55
2. \( \alpha' \neq \epsilon \),

3. \( |\alpha| < m \), where \( m \) is a constant that depends only on \( A \),

4. for each \( j \geq i_0 \), if \( c_j = (q, v, w''\sigma, \beta) \), then \( \beta = \beta'\alpha' \).

Proof. Let \( w = w_1w_2 \), such that \( |w_1| = r \), and let \( c_{i_0}' = (p', u', w_2\sigma, \gamma) \) be the last instantaneous description in the run whose third component is \( w_2\sigma \), i.e. the last before reading the \((r+1)\)th symbol of \( w \). We shall show that at least one symbol of \( w_1 \) is in \( [\gamma] \).

Indeed, if we assume the contrary, then, by Proposition 9, and since none of \( w_1 \)'s symbols appear in \( w_2 \) due to the choice of \( z \), all of \( w_1 \)'s symbols are stored in the registers, i.e. \([u'] = [w_1]\). To make the next step, the stack-based reassignment has to be performed, which will reassign one of the registers with the first symbol of \( \gamma \). Since none of \( \gamma \)'s symbols are from \( w_1 \), none of them appears in \( u' \). Thus, there is no way to perform the reassignment without losing one of the symbols of \( w_1 \), which contradicts Proposition 9.

Let \( \alpha' \) be the shortest suffix of \( \gamma \) that contains a symbol from \( w_1 \) (note, that it has to be the first symbol of \( \alpha' \), and \( \alpha' \neq \epsilon \)). Then \( \gamma = \alpha''\alpha' \). We contend that \( \alpha'' \) is also nonempty.

Indeed, were \( \alpha'' = \epsilon \), the first symbol of \( \alpha' \) would be the only one from \( w_1 \) in the stack. Thus, the rest \( r-1 \) symbols of \( w_1 \) fill \( r-1 \) registers of \( u \).

Therefore, when the stack-based reassignment is performed before the next step, the only register that may be reassigned with the first symbol of \( \alpha' \) is the remaining register, as follows from Proposition 9. After this reassignment, the content of the registers coincides with \([w_1]\). Since the next step cannot be an \( \epsilon \)-move due to the choice of \( c_{i_0}' \), one of the registers now has to be reassigned with the first symbol of \( w_2 \) during the input-base reassignment.

If the reassigned register is the one that contains the first symbol of \( \alpha' \), then the move cannot be made, since none of the registers is equal to the top of the stack. Otherwise, one of the symbols of \( w_1 \) is erased from the registers, which contradicts to Proposition 9. Therefore, \( \alpha'' \neq \epsilon \).

Now, we shall prove that condition 4 of the lemma holds for \( c_{i_0}' \), i.e. for each \( j \geq i_0 \), if \( c_j = (q, v, w''\sigma, \beta) \), then \( \beta = \beta'\alpha' \). For the sake of the proof, we strengthen the condition and request that \( \beta' \neq \epsilon \).

Consider \( c_j = (q, v, w''\sigma, \beta) \), where \( j = i_0' + k, k > 0 \). We prove by induction on \( k \), that \( \beta = \beta'\alpha' \), where \( \beta' \neq \epsilon \).
Basis. If $k = 1$, then $c_j$ is a successor of $c_{i_0}'$, and $\beta$ is obtained by replacing the first symbol of $\gamma$ with a string of symbols from $[v]$. Since $\alpha'' \neq \epsilon$, clearly $\alpha'$ stays unchanged, and we only need to show that $\beta' \neq \epsilon$. If $|\alpha''| > 1$, this is immediate, since the contents of the stack may decrease by one symbol at the most in one move. Otherwise, there are two possibilities: $\alpha'' = \alpha$ is a symbol of $w_1$, or otherwise.

- In the former case, by the argument used to show that $\alpha'' \neq \epsilon$, we can show that $\alpha$ is different from the first symbol of $\alpha'$. Thus, there are two registers that may receive $\alpha$ during stack-based reassignment (the remaining $r - 2$ are filled with $r - 2$ symbols of $w_1$ which are not in the stack, by Proposition 9). After $\alpha$ is assigned to one of them, the first symbol of $w_2$ is assigned to the other one during the input-based reassignment (it can be shown that there is only one choice of register for this last reassignment.) Now, to prove that $\beta' \neq \epsilon$, it suffices to show that the string pushed into the stack during the transition is not empty. It easily follows, once we note that after the transition, $r$ of the $r + 1$ symbols read from $w$ fill the $r$ registers (and are found only in the registers), and the $(r + 1)$th is at the top of the stack, which makes it impossible to perform the stack-based reassignment for the next step without erasing one of them. This contradicts Proposition 9 in the next instantaneous description.

- In the latter case, i.e., if $\alpha \notin [w_1]$, there is only one register to which $\alpha$ may be assigned to without loss of valuable symbols. Again, by the argument similar to that used to show $\alpha'' \neq \epsilon$, we arrive to the conclusion that this case is impossible.

Induction step. Suppose that the statement is true for $k$, and prove it for $k + 1$, i.e., if $c_j = (q, v, w''\sigma, \beta'\alpha')$, where $j > i_0'$ and $\beta' \neq \epsilon$, and $c_{j+1} = (\hat{q}, \hat{v}, \hat{w}''\sigma, \hat{\beta})$, then $\beta = \beta'\alpha'$, where $\beta' \neq \epsilon$.

Since $\beta' \neq \epsilon$, it suffices to prove that $\beta' \neq \epsilon$. If $|\beta'| > 1$, the prove is straightforward, since the stack lowers in each step by at most one symbol. Hence, we assume that $|\beta'| = 1$, and we may write $\beta' = \beta \in \Sigma$.

We distinguish between the following three cases.

- Let $|w''| = |w| - (r + 1)$. That is, $c_j$ is obtained from $c_{i_0'+1}$ by 0 or
more ε-moves.\(^1\) Thus, at the point in the computation represented by \(c_j\) exactly \(r + 1\) symbols are read from \(w\) and therefore have to appear either in the stack or in the registers.

- If \(\beta \notin [w]\), then only one of them appears at the stack (the one in \(\alpha'\)), and the rest \(r\) of them fill the registers. Thus, the stack-based reassignment cannot be performed without erasing one of the read symbols of \(w\) from the registers, contradictory with Proposition 9.
- If \(\beta\) is in \(w\), but it is equal to the first symbol of \(\alpha'\), it leads to the same problem.
- Thus, the only possible situation is when \(\beta\) is in \([w]\) and is different from the first symbol of \(\alpha'\). In this case there are 2 out of \(r + 1\) read symbols of \(w\) at the stack, and hence, there is one register eligible for the stack-based reassignment that can be performed. Since the move is on \(\epsilon\), the input-based reassignment is not performed. Let us examine the string which is pushed into the stack during the move. It cannot be \(\epsilon\), because if it was, it would be impossible to perform the stack-based reassignment for the following move without contradicting Proposition 9. This is, again, because only one of \(r + 1\) read symbols of \(w\) would be left at the stack, and the rest would be stored in the registers. This means that \(\beta' \neq \epsilon\), as was intended to show.

- Let \(|w| - |w''| = r + 2\), i.e., the number of symbols read from \(w\) is exactly \(r + 2\). There is only one possible distribution of these symbols between the stack and the registers: \(r\) of them are stored in the registers, one, as always, is in \(\alpha'\), and the last one is \(\beta'\). Since they are all different, as before, it is impossible to perform the stack-based reassignment without contradicting Proposition 9.
- Otherwise, \(|w| - |w''| > r + 2\), i.e., the number of symbols read from \(w\) is at list \(r + 3\). Since at most \(r\) of them may be stored in the registers, and only one is in \(\alpha'\), it follows that at list 2 of them have to be in \(\beta'\), which is impossible, since we are considering the case of \(|\beta'| = 1\).

Note, that we have established that there exists the instantaneous description \(c_i = (p', u', w_2, \gamma)\), that satisfies conditions 1, 2 and 4 of the

\(^1\)Recall, that \(c_{i+1}\) is obtained from \(c_i\) by reading a symbol from the input, therefore, the number of symbols read from \(w\) in \(c_{i+1}\) is \(r + 1\).
lemma. It does not, however, necessarily satisfies condition 3. To satisfy it, we need to locate another instantaneous description, $c_{i_0}$, that satisfies all of the conditions.

Let $i_0 = k + 1$, where

$$ k = \max \left\{ j \mid c_j = (q, v, z', \gamma), j < i_0', \text{ and } \alpha' \text{ is not a suffix of } \gamma \right\}. $$

That is, $c_k$ is the last instantaneous description in the computation before $c_{i_0}'$, whose stack content does not contain $\alpha'$ as a suffix. Then, in the transition from $c_k$ to $c_{i_0}$ the first symbol of $\alpha'$ is placed in the stack and is not removed until the whole input is read.

Let us see that $c_{i_0}$, when defined in this manner, satisfies all the conditions of the proposition.

It follows directly from the definition that $c_{i_0} = c_{k+1} = (p, u, w'\sigma, \alpha\alpha')$, where $\alpha'$ is as in the beginning of the proof, and $w_2$ is a suffix of $w'$. Thus, conditions 1 and 2 of the Lemma are satisfied. Condition 4 also follows easily from the definition of $c_{i_0}$ and from the fact that it holds for $c_{i_0}'$.

To see that condition 3 is also satisfied, suppose $c_k = (q, v, z', \gamma \gamma)$. Then, $\alpha\alpha'$ is obtained by replacing $\gamma$ with some string $\gamma'$, i.e. $\alpha\alpha' = \gamma'\gamma$. Since we know that $\gamma\gamma$ does not contain $\alpha'$ as a suffix, it follows that $\gamma$ is a proper suffix of $\alpha'$. $\alpha$ is a proper prefix of $\gamma'$. Now, we define $m$ to be the maximum length of a string pushed into the stack during one step, i.e. $m = \max\{l : (q, i, k, p, j_1, j_2 \cdots j_l) \in \mu\}$ is a constant, that depends only on the automaton $A$ itself. Then, $|\alpha| < |\gamma'| \leq m$ implies that the third condition is satisfied, which completes the proof of the Lemma.

\textbf{Lemma 11.} Let $c_{i_0} = (p, u, w'\sigma, \alpha\alpha')$ be the instantaneous description provided by Lemma 10. Then there is an instantaneous description $c_{i_1} = (q, v, \epsilon, \alpha')$ in the run of $A$,\footnote{Clearly, $i_1 > i_0$.} such that for any $c_j = (\tilde{p}, \tilde{u}, z', \gamma)$, such that $i_0 \leq j \leq i_1$, $\gamma$ is of the form $\tilde{\alpha}\alpha'$.

\textbf{Proof.} Consider $c_{i_1}' = (q', v', \sigma, \beta)$ - the last instantaneous description before reading $\sigma$. It follows from the proof of Lemma 10 (specifically, from the proof of the fourth property for $c_{i_0}$ and the definition of $c_{i_0}$), that $\beta = \beta'\alpha'$, where $\beta' \neq \epsilon$. Since at most one symbol may be removed from the stack in one step, it follows that $c_{i_1+1}$ is of the form $(q'', v'', \epsilon, \beta''\alpha')$.  

59
Recall that the run under consideration is an accepting run. Thus, \( c_n \) is of the form \((p', u', \epsilon, \epsilon)\), and, again, since at most one symbol is removed from the stack in one step, there exists \( c_j = (p'', u'', \epsilon, \alpha') \), where \( i'_1 + 1 \leq j < n \). Therefore, we may define \( i_1 \) by \( i_1 = \min\{ j : c_j = (p'', u'', \epsilon, \alpha'), ~ i'_1 + 1 \leq j < n \} \). The correctness of such definition follows from the properties of \( c_{i'_1 + 1} \). \qed
Appendix B

Auxiliary statements for Chapter 3

B.1 Why is DR-IPDA_n not stronger than DR-IPDA?

The part of this discussion regarding the addition of a finite number of stack symbols is also relevant for the proof of Theorem 24 of Chapter 4.

Here we discuss the reason why additional features of DR-IPDA_n do not make it stronger than DR-IPDA. We opt not to give the formal proof, which involves another elaborate construction, but instead, present the idea of such.

DR-IPDA_n differs from DR-IPDA by the following three features.

• DR-IPDA_n, as NR-FMA it was created to simulate, utilizes a blank symbol # in its assignments, hence allowing for a register to be “empty.” DR-IPDA does not allow empty registers.

• DR-IPDA_n has a finite number of symbols in addition to Σ and # that may be used in the assignments and the stack. DR-IPDA’s stack alphabet is restricted to Σ.

• Besides the input-based and stack-based reassignment functions of DR-IPDA, DR-IPDA_n features a reset function, that allows to “erase” contents of a register by writing # into it.

Clearly, DR-IPDA_n is at list as strong as DR-IPDA: any DR-IPDA can
be viewed as DR-IPDA\(_n\) such that \# does not appear in \(u\), \(\Delta = \emptyset\) and \(o(q) = \text{nil}\) for all \(q \in Q\).

To show that the features listed above do not strengthen the model, we need to simulate a given DR-IPDA\(_n\) by some DR-IPDA.

To simulate symbols of \(\Delta\) and \# in the stack, each symbol in the stack of DR-IPDA\(_n\) is represented by two symbols in the stack of DR-IPDA, both symbols of \(\Sigma\): the first (indicator) would indicate should the second be interpreted as symbols of \(\Sigma\), or as symbol of \(\Delta \cup \{\#\}\). It means that each step of DR-IPDA\(_n\) is simulated by two steps of DR-IPDA, the first one merely telling the second how to interpret the symbol on the top of the stack.

For this purpose we need to designate \(|\Delta| + 1 \geq 2\) symbols of \(\Sigma\) not appearing in the initial assignment (a symbol for each element of \(\Delta \cup \{\#\}\), two of them serving as indicators), and make sure that they are always present in the assignment.

To simulate empty registers, we keep a list of the registers considered empty in the state of the automaton.

These two solutions lead to the following consequences.

- In DR-IPDA, we need to keep a number of registers always filled with constant symbols. What happens if these symbols have to appear in a different register in DR-IPDA\(_n\) during the run?

- In DR-IPDA, the registers are never reset, but only added to the list of reset registers. What happens if a symbol “erased” by reset in DR-IPDA\(_n\) has to appear in a different register, but cannot in DR-IPDA, because the “not-erased” register still contains it?

These two problems have a single solution: register mapping. We equip each state of DR-IPDA with a vector, telling for each register of DR-IPDA\(_n\), which register of DR-IPDA simulates it. This vector can also indicate if a register is empty, eliminating the need for a list of empty registers. A similar construction is employed in the proof of Proposition 17 on page 33.

Appropriate changes would be made to input-based and stack-based re-assignments and to the transition function.
B.2 Proof of Theorem 13: the construction

The idea of the construction is presented on page 28.

Let $A = (Q, q_0, u_0, \mu, \rho, F)$ be an $r$-register NR-FMA, $u_0 = \tilde{u}_1^0 \tilde{u}_2^0 \ldots \tilde{u}_r^0$. We may assume without loss of generality, that $A$ has the following property: for each $f \in F$, $q \in Q$ and $i \in \{1, 2, \ldots, r\}$, $(f, i, q) \notin \mu$. That is, the automaton stops reading from the input once it reaches one of its final states.

We construct a DR-IPDA $A'$, such that $L(A) = L(A')$, where $A'$ simulates $A$ in the following manner. Let $A' = <Q', q'_0, \Delta, u'_0, o, \pi, \rho', \mu'>$, where $Q', q'_0, \Delta, u'_0, o, \pi, \rho'$ and $\mu'$ are defined as follows.

- $Q' = Q_{\text{simulate}} \cup Q_{\text{load}} \cup Q_{\text{RC}} \cup \{q'_0, f\}$, where $Q_{\text{simulate}} = \left\{ q^S_\xi \middle| q \in Q, S \subseteq \{1, 2, \ldots, r\}, \xi \in \{1, 2, \ldots, r\} \right\}$, $Q_{\text{load}} = \left\{ q^{S,i}_\xi \middle| i \in \{1, 2, \ldots, r + 1\}, S \subseteq \{1, 2, \ldots, r\} \right\}$, $Q_{\text{RC}} = \left\{ q^S_\xi, q^S_R | S \subseteq \{1, 2, \ldots, r\} \right\}$.
- $\Delta = \{\delta_1, \delta_2, \ldots, \delta_r, R, C, \perp\}$.
- $u'_0 = u_0 \delta_1 \delta_2 \ldots \delta_r \#\text{RC}\perp$.

To make the definition of $\mu'$ more readable, we will use the following notation. To introduce it, we fix the last $r + 4$ registers to be constant, i.e., the automaton will never reset or overwrite them. The symbols contained in these registers are $\#$ and the symbols of $\Delta$, which never appear in the input and are used by $A'$ to code the logs of $A$'s actions into the stack. Therefore, wherever comparison to the content of one of these registers or use of their content is due, instead of writing the number of the register, we will write the symbol assigned to it by the initial assignment, i.e., the symbol the number of the register actually refers to. For example, instead of $\mu(q, \epsilon, 2r + 1) = (p, (2r + 1)12 \cdots jr(2r + 2)(r + 1))$ we shall write $\mu(q, \epsilon, \#) = (p, \#12 \cdots rR\delta_1)$.

Using this notation, we define $\mu'$, together with $o$, $\pi$ and $\rho'$, in the following manner.

1. $o(q'_0) = \text{nil}$,
2. $\pi(q'_0) = 2r + 4$ (\perp on the stack),

63
\( \rho' \) may be defined arbitrarily,
\( \mu'(q'_0, \epsilon, \bot) = \{(q_0^0, \#12 \cdots r, \bot) \mid \xi \in \{1, 2, \ldots, r\} \cup \{p : (q_0, \epsilon, p) \in \mu\}\}. \\
2. For each \( q'_p \in Q', q, p \in Q \),
\( o(q'_p) = \rho(q, p) \),
\( \pi(q'_p) = 2r + 1 \) (# on the stack),
\( \rho' \) may be defined arbitrarily,
\( \mu'(q'_p, \epsilon, \#) = \{(q, \rho_p, \# \alpha) \mid (q, \rho_p, \# \alpha) \in \mu, q \in \{1, 2, \ldots, r\} \cup \{q' : (p, \epsilon, q') \in \mu\}\}, \)
where \( S' = S \cup \{\rho(q, p)\} \) and \( \alpha = 12 \cdots r \delta_p(q, p) \) (“reset register \( i \)”), if \( \rho(q, p) \neq \text{nil} \), whereas \( S' = S \) and \( \alpha = \epsilon \), otherwise.
3. For each \( q'_i \in Q', i \in \{1, 2, \ldots, r\} \),
\( o(q'_i) = \text{nil} \),
\( \pi(q'_i) = 2r + 1 \) (# on the stack),
\( \rho'(q'_i) = \begin{cases} 
  i, & \text{if } i \in S \\
  2r + 1, & \text{otherwise.}
\end{cases} \)
\( \mu'(q'_i, i, \#) = \{(q, i, \rho, \# \alpha) \mid (q, i, \rho, \# \alpha) \in \mu, q \in \{1, 2, \ldots, r\} \cup \{q' : (p, \epsilon, q') \in \mu\}\}, \)
where \( \alpha = \epsilon \) if \( i \notin S \), and \( \alpha = 12 \cdots r \delta_i \) (“copied input into register \( i \)”), otherwise.

The above transitions implement the simulation as described above. The transitions below are “responsible” for checking the data that was logged in the stack during the simulation, to verify its correctness. For each state mentioned from hereon, \( \rho' \) may be defined arbitrarily, since for none of them the moves involve reading from the input.

4. For all \( q'_k \in Q' \) such that \( q \in F \),
\( o(q'_k) = \text{nil} \),

\(^1\)Note, that this is the only place where we allow reassignment of one of the registers declared as constant, that may actually alter it (all the other cases of reassigning a constant register involve copying a known symbol from the stack into the register that already contains that symbol). However, in this specific case it does not matter, since if such reassignment in fact succeeds, the computation will immediately halt, because there is no move that involves comparing input to a register with index greater than \( r \).
\[
\pi(q^S_i) = 2r + 1 \quad (# \text{ on the stack}),
\]
\[
\mu'(q^S_i, \epsilon, \#) = \{(q^1_{\text{load}}, \epsilon)\}.
\]

5. For all \(i \in \{1, 2, \ldots, r\}\), \(S \subseteq \{1, 2, \ldots, r\}\), such that \(i \notin S\),
- \(o(q^S_i) = \text{nil}\),
- \(\pi(q^S_i) = i\),
- \(\mu'(q^S_i, \epsilon, i) = \{(q^S_{i+1} \text{Load}, \epsilon)\}\).

6. For all \(i \in \{1, 2, \ldots, r\}\), \(S \subseteq \{1, 2, \ldots, r\}\), such that \(i \in S\),
- \(o(q^S_i) = \text{nil}\),
- \(\pi(q^S_i) = 2r + 1 \quad (# \text{ on the stack})\),
- \(\mu'(q^S_i, \epsilon, \#) = \{(q^S_{i+1} \text{Load}, \epsilon)\}\).

7. For all \(S \subseteq \{1, 2, \ldots, r\}\),
- \(o(q^{S,r+1}_{\text{Load}}) = \text{nil}\),
- \(\pi(q^{S,r+1}_{\text{Load}}) = 2r + 4 \quad (\text{R, C, or } \perp \text{ on the stack})\),
- \(\mu'(q^{S,r+1}_{\text{Load}}, \epsilon, r) = \{(q^S_R, \epsilon)\}\),
- \(\mu'(q^{S,r+1}_{\text{Load}}, \epsilon, C) = \{(q^S_C, \epsilon)\}\),
- \(\mu'(q^{S,r+1}_{\text{Load}}, \epsilon, \perp) = \{(f, \epsilon)\}\).

8. For all \(i \in \{1, 2, \ldots, r\}\), \(S \subseteq \{1, 2, \ldots, r\}\),
- \(o(q^S_R) = \text{nil}\),
- \(\pi(q^S_R) = r + 1 \quad (\text{one of } \delta_i\text{s on the stack})\),
- \(\mu'(q^S_R, \epsilon, r + i) = \{(q^{S_{\{i\}}}_{\text{Load}}, \epsilon)\}\).

9. For all \(i \in \{1, 2, \ldots, r\}\), \(S \subseteq \{1, 2, \ldots, r\}\),
- \(o(q^S_C) = \text{nil}\),
- \(\pi(q^S_C) = r + 1 \quad (\text{one of } \delta_i\text{s on the stack})\),
- \(\mu'(q^S_C, \epsilon, r + i) = \{(q^{S_{\{i\}}}_{\text{Load}}, \epsilon)\}\).

### B.3 Proof of Theorem 13: the correctness

Here we shall prove that the DR-IPDA \(A'\) defined in the previous section indeed accepts \(L(A)\), and by such complete the proof of Theorem 13.

From hereon we denote the constant string \(\delta_1\delta_2 \cdots \delta_r\#RC\perp\) by \(u_c\).
B.3.1 $L(A) \subseteq L(A')$

Let $\sigma = \sigma_1 \sigma_2 \cdots \sigma_n \in A$, and let $c_0, c_1, c_2, \ldots, c_l, c_i = (q^i, u^i, \sigma^i), u^i = u^i_1 u^i_2 \cdots u^i_r$ be a sequence of instantaneous descriptions, representing an accepting run of $A$ on $\sigma$.

We define the following sequences.

- $\xi_0, \xi_1, \xi_2, \ldots, \xi_l$, where each $\xi_i$ is an element of $\{1, 2, \ldots, r\} \cup Q$, and is defined as follows:
  - if $i = l$, then $\xi$ may be defined arbitrarily;
  
  for $i < l$:
  - if $c_{i+1}$ is obtained from $c_i$ by an $\epsilon$-move, then $\xi_i = q^{i+1}$; 
  - if $c_{i+1}$ is obtained from $c_i$ using the transition $(q^i, j, q^{i+1}) \in \mu$, then $\xi_i = j$.

- $S_0, S_1, S_2, \ldots, S_l$, where each $S_i$ is a subset of $\{1, 2, \ldots, r\}$, and is defined recursively as follows:
  - $S_0 = \emptyset$;
  
  for $i > 0$:
  - if $\xi_{i-1} = q^i$ and $\rho(q^{i-1}, q^i) \neq \text{nil}$, then $S_i = S_{i-1} \cup \{\rho(q^{i-1}, q^i)\}$;
  - if $\xi_{i-1} = q^i$ and $\rho(q^{i-1}, q^i) = \text{nil}$, then $S_i = S_{i-1}$;
  - if $\xi_{i-1} = j$, then $S_i = S_{i-1} \setminus \{j\}$.

- $v^0, v^1, v^2, \ldots, v^l$, where each $v^i$ is an assignment in $\Sigma^{r\neq \#}$, such that $v^i = v^i_1 v^i_2 \cdots v^i_r$ and

$$v^i_j = \begin{cases} 
\# & \text{for } j \in S_i, \\
u^i_j & \text{for } j \notin S_i.
\end{cases}$$

Note, that correctness of the definition (the fact that $v^i$ is indeed an assignment) follows from the fact that $u^i$ is an assignment.

- $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^l$, where each $\alpha^i$ is a string over $\Sigma \cup \Delta$ and is defined recursively as follows:
Proposition 31.

For each $\mathcal{A}', d_{-1}, d_0, d_1, d_2, \ldots, d_l$, by putting

$$d_{-1} = (q'_0, u_0u_c, \sigma, \perp),$$

and for $i \geq 0$

$$d_i = ((q^i)^{S_i}, v^i u_c, \sigma^i, \#\alpha^i).$$

**Proposition 30.** The sequence $d_0, d_1, d_2, \ldots, d_l$ is a run of $\mathcal{A}'$ on $\sigma$.

**Proof.** Clearly, $d_{-1}$ is the initial instantaneous description of $\mathcal{A}'$. It is also clear, that, since $d_0 = (\langle q_0 \rangle^0_1, u_0u_c, \sigma, \#u_0\perp)$, $d_0$ can be obtained from $d_{-1}$ by a transition of type 1 (see the definition of $\mu'$). It remains to show that for each $i > 0$ $d_{i-1} \vdash_{\mathcal{A}'} d_i$, where, by the definition of $d_i$s, $d_{i-1} = (p^{i-1}, v^{i-1} u_c, \sigma^{i-1}, \#\alpha^{i-1})$ and $d_i = (p^i, v^i u_c, \sigma^i, \#\alpha^i)$. We distinguish between the following two cases.

- $c_i$ is obtained from $c_{i-1}$ by an $\epsilon$-move. In this case, by the definition of $\xi_{i-1}$, $p^{i-1} = (q^{i-1})^{S_{i-1}}$, while $p^i = (q^i)^{S_i}$. Then $d_i$ can obtained from $d_{i-1}$ by a transition of type 2.

- $c_i$ is obtained from $c_{i-1}$ using the transition $(q^{i-1}, j, q^i) \in \mu$. By the definition of $\xi_{i-1}$, in this case $p^{i-1} = (q^{i-1})^{S_{i-1}}$. Then $d_i$ can obtained from $d_{i-1}$ by a transition of type 3.

\[\Box\]

**Proposition 31.** For each $l' < l$, the string $\alpha''$ is of the form

$$v^{i_0} x^{i_0} v^{i_{m'}} x^{i_{m'-1}} \ldots v^{i_1} x^{i_1} v^{i_0} \perp,$$

where $i_0 = 0$, $i_{m'} \leq l'$, $x^{i_k} \in \{R, C\} \{\delta_1, \delta_2, \ldots, \delta_r\}$ for each $0 \leq k \leq m'$, and the following holds.
1. For each \( 0 < k \leq m' \) and for each \( i_{k-1} \leq i < i_k \), if \( i + 1 < i_k \), then either \( c_{i+1} \) is a result of \( \epsilon \)-move and \( \rho(q^i, q^{i+1}) = \text{nil} \), or \( c_{i+1} \) is obtained by the transition \( (q^i, j, q^{i+1}) \), \( j \notin S_i \) and \( v^i_j \) equals the first symbol of \( \sigma^i \). In addition, \( v^{i+1} = v^i = v^{i_k-1} \) and \( S_{i+1} = S_i = S_{i_k-1} \).

Also, for each \( i_{m'} \leq i < l' \), if \( i + 1 \leq l' \), then a similar statement holds.

That is, either \( c_{i+1} \) is the result of \( \epsilon \)-move and \( \rho(q^i, q^{i+1}) = \text{nil} \), or \( c_{i+1} \) is obtained by the transition \( (q^i, j, q^{i+1}) \), \( j \notin S_i \) and \( v^i_j \) equals the first symbol of \( \sigma^i \). In addition, \( v^{i+1} = v^i = v^{i_m'} \) and \( S_{i+1} = S_i = S_{i_m'} \).

2. For each \( 0 < k \leq m' \), if \( x^{i_k} = R\delta_j \), then \( c_{i_k} \) is obtained by an \( \epsilon \)-move, where \( \rho(q^{i_k-1}, q^{i_k}) = j \), \( S_{i_k} = S_{i_k-1} \cup \{ j \} \) and \( v^i_j = \# \), while \( v_{j'}^{i_k} = v_{j'}^{i_k-1} \) for \( j' \neq j \). That is, \( v^{i_k} \) is obtained from the previous assignment \( v^{i_k-1} \) by resetting the register \( j \).

3. For each \( 0 < k \leq m' \), if \( x^{i_k} = C\delta_j \), then \( c_{i_k} \) is obtained by the transition \( (q^{i_k-1}, j, q^{i_k}) \) such that \( j \in S_{i_k-1} \), \( S_{i_k} = S_{i_k-1} \setminus \{ j \} \), and \( v^i_j \) equals the first symbol of \( \sigma^{i_k-1} \), while \( v_{j'}^{i_k} = v_{j'}^{i_k-1} \) for \( j' \neq j \). That is, \( v^{i_k} \) is obtained from the previous assignment \( v^{i_k-1} \) by assigning the register \( j \), which was empty, with the next input symbol of instantaneous description \( c_{i_k-1} \).

**Proof.** The Proposition follows directly from the recursive definition of \( \alpha^l \) and the definitions of other sequences. \( \square \)

Since \( c_0, c_1, c_2, \ldots, c_l \) is an accepting run, it follows that \( q^l \in F \) and \( \sigma^l = \epsilon \). Thus, using a transition of type 4, we may write

\[
d_i = ((q^i)_{S_i}, v^i, \epsilon, \# \alpha^l) \vdash_{A'} (q^{j_1}_{\text{load}}, v^i, \epsilon, \alpha^l).
\]

Let

\[
\alpha^l = v^{i_m} x^{i_m} v^{i_{m-1}} x^{i_{m-1}} \ldots v^{i_1} x^{i_1} v^{i_0},
\]

as described in Proposition 31.

It follows from the definition of \( A' \) (moves of type 4 to 9) that starting from \( q^{j_1}_{\text{load}} \), the automaton proceeds emptying the stack a symbol at a time. We show that this procedure continues uninterrupted until the stack is emptied, and thus, \( A' \) reaches an accepting configuration.

To show it, we first define the following sequences.

- \( \{ T_k \}_{k=0}^m \), where each \( T_k \) is a subset of \( \{1, 2, \ldots, r\} \), and is defined recursively, in the following way:
- $T_m = \emptyset$;
- $T_{k-1} = \begin{cases} T_k \setminus \{j\} & \text{if } x^i_k = R\delta_j, \\ T_k \cup \{j\} & \text{if } x^i_k = C\delta_j. \end{cases}$

**Proposition 32.** We have:

1. $T_k \subseteq S_{ik}$.
2. If $T_{k-1} = T_k \cup \{j\}$, then $j \notin T_k$.
3. $T_0 = \emptyset$.

**Proof.** See Appendix B.

- $\{w_{k,i}^{m,i}\}_{k=0,i=0}^{m,r}$, where each $w_{k,i}^{k,i} = w_1^{k,i}w_2^{k,i} \ldots w_r^{k,i}$ is an assignment in $\Sigma_r^\#$, defined recursively as follows:
  - $w_{m,i}^{m,i} = v_{i}^{m}$ for all $0 \leq i \leq r$;

for $k < m$:

- $w_{k-1,i}^{k-1,i} = \begin{cases} w_{k,i}^{k,r} & \text{if } i = 0, \\ w_{k,i}^{k,r} & \text{for all } i, \text{ if, for some } j, \ x^i_k = C\delta_j, \\ w_{k,i}^{k,r} & \text{if } x^i_k = R\delta_j \text{ and } i < j, \\ \tilde{w}_{k,i}^{k,r} & \text{if } x^i_k = R\delta_j \text{ and } i \geq j, \end{cases}$

where $\tilde{w}_{k,i}^{k,r}$ is defined by

$$\tilde{w}_{j'}^{k,r} = \begin{cases} v_{j'}^{k-1}, & \text{for } j' = j, \\ w_{j'}^{k,r}, & \text{otherwise}. \end{cases}$$

The correctness of the definitions follows as a corollary from the following proposition.

**Proposition 33.** For each $0 \leq k \leq m$,

$$w_{i}^{k,r} = \begin{cases} v_{i}^{k,k}, & \text{for } i \notin T_k, \\ u_{i}^{k,k}, & \text{for } i \in T_k. \end{cases}$$

**Proof.** See Appendix B.

**Corollary 34.** $w_{m-k,i}^{m,k} \in \Sigma_r^\#$ for all $0 \leq k \leq m$, $0 \leq i \leq r$. 

69
Proof. For $k = 0$, the statement follows from the fact that $v_i^m \in \Sigma_{\#}^{r_i}$. From Proposition 33, part 1 of Proposition 32 and and the definition of $u_i^{m-k}$ we get
\[
w_j^{m-k,r} = \begin{cases} 
\# & \text{if } j \in S_i^{m-k} \setminus T_{m-k}, \\
u_j^{m-k} & \text{otherwise.}
\end{cases}
\]
Thus, $w_i^{m-k,r} \in \Sigma_{\#}^{r_i}$, as $u_i^{m-k} \in \Sigma_{\#}^{r_i}$, for all $0 \leq k \leq m$. Since for all $0 < k \leq m$, $0 \leq i \leq r$, $w_i^{m-k,i}$ equals either $w_i^{m-(k-1),r}$ or $w_i^{m-k,r}$, the statement easily follows by induction on $k$.

Let us denote the instantaneous description $(q_{\text{load}}^{0,1}, v^i, \epsilon, \alpha^i)$ mentioned above by $e_1^m$. By Proposition 31, part 1, $e_1^m = (q_{\text{load}}^{0,1}, v^i, \epsilon, \alpha^i)$, and, as mentioned, $d_i \vdash_{\mathcal{A}'} e_1^m$.

We now introduce the sequence of instantaneous descriptions
\[e_1^m, e_2^m, \ldots, e_r^m, e_1^{m-1}, e_2^{m-1}, \ldots, e_r^{m-1}, \ldots, e_1^1, e_2^1, \ldots, e_r^1, e_1^0, e_2^0, \ldots, e_r^0,
\]
defined as follows $^2$:

for $0 < k \leq m$, $1 \leq i \leq r + 1$, 
\[e_i^k = (q_{\text{load}}^{T_k,i}, w^{k,i-1}, \epsilon, \text{suff}_{r-i+1}(v^{i_k}) \alpha^{i_k-1}),
\]
and 
\[e_r^{r+2} = \begin{cases} 
(q_{T_k}^{R}, w^{k,r}, \epsilon, \delta_j \alpha^{i-1}), & \text{if } \alpha^i = R \delta_j, \\
(q_{T_k}^{C}, w^{k,r}, \epsilon, \delta_j \alpha^{i-1}), & \text{if } \alpha^i = C \delta_j,
\end{cases}
\]
and for $1 \leq i \leq r + 1$
\[e_1^0 = (q_{\text{load}}^{T_0,i}, w^{0,i-1}, \epsilon, \text{suff}_{r-i+1}(v^{i_0}) \perp),
\]
and 
\[e_r^{r+2} = (f, w^{0,r}, \epsilon, \epsilon).
\]

Since by this definition $e_{r+2}^0$ is of an accepting kind, it suffices to show that the sequence is a sub-run of $\mathcal{A}'$, i.e. for each $0 \leq k \leq m$ and $1 \leq i \leq r + 1$, $e_i^k \vdash_{\mathcal{A}'} e_{i+1}^k$, and, for each $0 < k \leq m$, $e_r^{r+2} \vdash_{\mathcal{A}'} e_{i+1}^i$.

It is not hard to make sure that $e_r^{r+1} \vdash_{\mathcal{A}'} e_r^{r+2}$ and $e_r^{r+2} \vdash_{\mathcal{A}'} e_{i+1}^i$ by referring to the definitions of $\mathcal{A}'$ (moves of type 7 and 8), $T_i$s and $w^{k,i}$s.

The substantial part of the proof is to show that, for each $0 \leq k \leq m$, $e_i^k \vdash_{\mathcal{A}'} e_{i+1}^k$, because that is where the correctness of the simulation is expressed.

$^2$We denote the prefix and the suffix of length $i$, $0 \leq i \leq |w|$, of a word $w$ by $\text{pref}_i(w)$ and $\text{suff}_i(w)$, respectively.
Proposition 35. For each $0 \leq k \leq m$ and $1 \leq i \leq r$, $e_i^k \vdash_{A'} e_{i+1}^k$.

Proof. By examining the definition of $A'$ we may conclude, that $e_i^k$ may yield $e_{i+1}^k$ by the moves of type 5 or 6, and $e_i^k \vdash_{A'} e_{i+1}^k$ if and only if the following holds:

- if $i \in T_k$, then $v_i^{i_k} = \#$, and $w_i^{k,i} = w_i^{k,i-1}$;
- if $i \notin T_k$, then, either $v_i^{i_k} = w_i^{k,i-1}$ and $w_i^{k,i} = w_i^{k,i-1}$, or $v_i^{i_k} \notin [w_i^{k,i-1}]$ and $w_i^{k,i}$ is obtained from $w_i^{k,j-1}$ by replacing $w_i^{k,j-1}$ with $v_i^{i_k}$.

Let us see that this condition indeed holds.

If $i \in T_k$, then, by Proposition 32, part 1, $i \in S_{i_k}$. By definition of $v_i^{i_k}$, it follows that $v_i^{i_k} = \#$. Also, if $x_{i+1} = C\delta_j$, then, $w_i^{k,i-1} = w_i^{k,i} = w_i^{k+1,r}$ by definition, and if $x_i = R\delta_j$, then $j \notin T_k$, therefor, $i \neq j$, thus, either $i = i - 1 < j$ and $w_i^{k,i} = w_i^{k,i-1} = w_i^{k+1,r}$, or $i, i - 1 \geq j$, and $w_i^{k,i} = w_i^{k,i-1} = w_i^{k+1,r}$.

If $i \notin T_k$, then let us distinguish between the following two cases.

- $x_i = C\delta_j$. Then, by definition, $w_i^{k,r} = w_i^{k,i} = w_i^{k,i-1} = w_i^{k+1,r}$, and it follows from Proposition 33, that $v_i^{i_k} = w_i^{k,i-1}$.
- $x_i = R\delta_j$. In this case, if $i = j$, then, by definition, $w_i^{k,i-1} = w_i^{k+1,r}$, and $w_i^{k,i} = w_i^{k,r}$, which is obtained by replacing $w_i^{k+1,r}$ with $v_i^{i_k}$ in $w_i^{k+1,r} = w_i^{k,i-1}$. (Indeed, if $w_i^{k+1,r} \neq v_i^{i_k}$, then $v_i^{i_k} \notin [w_i^{k,i-1}]$, otherwise, we would get contradiction to the fact that $w_i^{k,i}$ is an assignment.) Otherwise, i.e. if $i \neq j$, then, $w_i^{k,i} = w_i^{k,i-1} = w_i^{k+1,r}$, and $w_i^{k,i-1} = w_i^{k,r} = v_i^{i_k}$ by Proposition 33.

To summarize the proof of the inclusion $L(A) \subseteq L(A')$, given an accepting run of $A$ on a word $\sigma \in L(A) e_0, c_1, c_2, \ldots, c_l$, we have constructed a sequence of instantaneous descriptions

$$d_{-1}, d_0, d_1, d_2, \ldots, d_l, e_1^m, e_2^m, \ldots, e_{r+2}^m, \ldots, e_1^0, e_2^0, \ldots, e_{r+2}^0,$$

and shown it to be an accepting run of $A'$ on $\sigma$, which is an evidence of $\sigma \in L(A')$.

71
B.3.2 \( L(\mathcal{A}') \subseteq L(\mathcal{A}) \)

Let \( \sigma = \sigma_1 \sigma_2 \cdots \sigma_n \in \mathcal{A}' \). By examining the definition of \( \mu' \), we observe that an accepting run of \( \mathcal{A}' \) on \( \sigma \) has to be of the form

\[
d_{-1}, d_0, d_1, d_2, \ldots, d_l, e_1^m, e_2^m, \ldots, e_{r+2}^m, \ldots, e_1^0, e_2^0, \ldots, e_{r+2}^0,
\]

where the first \( l + 1 \) moves are made by transitions of type 1 to 3, the last \( (r + 2)(m + 1) \) by transitions of type 4 to 9.

The first \( l + 2 \) instantaneous descriptions must satisfy the following conditions.

- \( d_{-1} = (q_0^0, u_0 u_c, \sigma, \perp) \).
- \( d_i = ((q_i^i)_{\xi_i}^S, v^i u_c, \sigma^i, \# \alpha^i), 0 \leq i \leq l \), where
  - \( (q_i^i)_{\xi_i}^S \in Q_{\text{simulate}} \),
  - \( v^i = v_1^i v_2^i \ldots v_l^i \in \Sigma^\#, \)
  - \( \sigma^i \) is a suffix of \( \sigma \), such that \( |\sigma^{i+1}| \leq |\sigma^i|, \sigma^0 = \sigma \) and \( \sigma^l = \epsilon \),
  - \( \alpha^i \) is the string of the form \( v^{i_0} x^{i_1} \ldots x^{i_k} v^{i_0} \), where \( i_0 = 0, i_{m_i} \leq i \) and \( x^{i_k} \in \{R, C\} \Delta, 0 \leq k \leq m_i \),

and all of the parameters listed are bound by the following relations.

As follows from the definition of moves of type 1,

\[
d_0 = ((q_0^0)_{\xi_0}^0, u_0 u_c, \sigma, \# u_0^0 \perp),
\]

i.e. \( q_0^0 = q_0, S_0 = \emptyset, v_0 = u_0, \sigma^0 = \sigma \), and \( \alpha^0 = u_0^0 \perp \). In addition, for all \( 0 \leq i < l \):

- if \( \sigma^{i+1} = \sigma^i \), then \( d_{i+1} \) is a result of a move of type 2, thus,
  - \( (p_i, \epsilon, p_{i+1}) \in \mu; \)
  - \( \xi_i = p_{i+1}; \)
  - \( v^{i+1} \) is obtained form \( v^i \) by resetting the register with index \( o((q_i^i)_{\xi_i}^S) \), if it is not nil, and equals \( v^i \), otherwise. I.e., if \( o((q_i^i)_{\xi_i}^S) = \text{nil} \), then \( v^{i+1} = v^i \), otherwise, \( v^{i+1} = v^i_j \) for \( j \neq o((q_i^i)_{\xi_i}^S) \), and \( v_{o((q_i^i)_{\xi_i}^S)}^{i+1} = \#; \)
Note, that it easily follows from the definition of moves of type 1-3, that for all \( 0 \leq i \leq l \), \( S_i = \{ j | v^i_j = \# \} \).

Also, let us recall Proposition 31, that appears in the first part of the proof (\( L(A) \subseteq L(A') \)), and deals with the elements of the run of \( A' \), that was constructed from the given run of \( A \). Since for the description of the given run of \( A' \) in the current context (\( L(A') \subseteq L(A) \)) we have conveniently chosen the same notation, it follows from the relations described above that the proposition again holds. Nevertheless, the elements mentioned in Proposition 31 and those in this discussion, though named identically and having the same properties, are still different objects. Thus, we recite (with a minor correction \(^3\)) the statement here as Proposition 36.

**Proposition 36.** For each \( l' < l \), the string \( \alpha'' \) is of the form

\[
v^{i_{m'}}x^{i_{m'}}v^{i_{m'-1}}x^{i_{m'-1}}\ldots v^{i_1}x^{i_1}v^{i_0} \bot,
\]

where \( i_0 = 0 \), \( i_{m'} \leq l' \), \( x^k \in \{ R, C \} \{ \delta_1, \delta_2, \ldots, \delta_r \} \) for each \( 0 \leq k \leq m' \), and the following holds.

---

\(^3\)Proposition 31 mentions instantaneous descriptions \( c_i \), which are the elements of the given run of \( A \), irrelevant to this context. Not incidentally, the facts mentioned about \( c_i \)'s are also true for \( d_i \)'s that we have here, thus, we may substitute them without compromising the meaning of the proposition.
1. For each $0 < k \leq m'$ and for each $i_{k-1} \leq i < i_k$, if $i + 1 < i_k$ as well, then either $d_{i+1}$ is a result of $\epsilon$-move and $\rho(q^i, q^{i+1}) = \text{nil}$, or $d_{i+1}$ is obtained by the transition in $\mu'$ derived from $(q^i, j, q^{i+1}) \in \mu$, $j \notin S_i$, and $v_j^i$ equals the first symbol of $\sigma^i$. In addition, $v_{i+1} = v_i = v_{i-1}$ and $S_{i+1} = S_i = S_{i-1}$.

Also, for each $i_{m'} \leq i < l'$, if $i + 1 \leq l'$, then a similar statement holds. That is, either $d_{i+1}$ is a result of $\epsilon$-move and $\rho(q^i, q^{i+1}) = \text{nil}$, or $d_{i+1}$ is obtained by the transition in $\mu'$ derived from $(q^i, j, q^{i+1}) \in \mu$, $j \notin S_i$, and $v_j^i$ equals the first symbol of $\sigma^i$. In addition, $v_{i+1} = v_i = v_{i-1}$ and $S_{i+1} = S_i = S_{i-1}$.

2. For each $0 < k \leq m'$, if $x_{ik} = R\delta_j$, then $d_{ik}$ is obtained by an $\epsilon$-move, $\rho(q^{ik-1}, q^{ik}) = j$, $S_{ik} = S_{ik-1} \cup \{j\}$ and $v_{jk}^i = \#$, while $v_j^{i+1} = v_j^{i}-1$ for $j' \neq j$. That is, $v_{ik}$ is obtained from the previous assignment $v_{ik-1}$ by resetting the register $j$.

3. For each $0 < k \leq m'$, if $x_{ik} = C\delta_j$, then $d_{ik}$ is obtained by the transition in $\mu'$ derived from $(q^{ik-1}, j, q^{ik}) \in \mu$ such that $j \in S_{ik-1}$, thus $S_{ik} = S_{ik-1} \setminus \{j\}$, and $v_{jk}^i$ equals the first symbol of $\sigma^{ik-1}$, while $v_j^{i+1} = v_j^{i}-1$ for $j' \neq j$. That is, $v_{ik}$ is obtained from the previous assignment $v_{ik-1}$ by assigning the register $j$, which was empty, with the next input symbol of instantaneous description $d_{ik-1}$.

Proof. The Proposition follows directly from the structure of $d_i$s and the relations its components satisfy, as described above.

The last $(r + 2)(m + 1)$ instantaneous descriptions,

$e_1^m, e_2^m, \ldots, e_{r+2}^m, e_1^{m-1}, e_2^{m-1}, \ldots, e_{r+2}^{m-1}, \ldots, e_1^1, e_2^1, \ldots, e_{r+2}^1, e_1^0, e_2^0, \ldots, e_{r+2}^0,$

are as follows:

for $0 < k \leq m$, $1 \leq i \leq r + 1$,

$e_i^k = (q_{\text{load}}, w_{i}, w_{i-1}, \epsilon, \text{suff}_{r-i+1}(v_{ik})x_{ik}|\alpha_{ik-1}^-),$

and

$e_{r+2}^k = \left\{ \begin{array}{ll}
(q_{\text{load}}, w_{i}, w_{i-1}, \epsilon, \delta_j|\alpha_{ik-1}^-), & \text{if } x_{ik} = R\delta_j, \\
(d_{\text{C}}, w_{i}, w_{i-1}, \epsilon, \delta_j|\alpha_{ik-1}^-), & \text{if } x_{ik} = C\delta_j,
\end{array} \right.$

and for $1 \leq i \leq r + 1$

$e_i^0 = (q_{\text{load}}, w_{0}, w_{i-1}, \epsilon, \text{suff}_{r-i+1}(v_{ik})\bot),$
and 
\[ e_{r+2}^0 = (f, w_{0,r}^0, \epsilon, \epsilon), \]
while their components are bound by the following relations:

- \(\{T_k\}_{k=0}^m\), where each \(T_k\) is a subset of \(\{1, 2, \ldots, r\}\), and is defined recursively, in the following way:
  - \(T_m = \emptyset\);
  - \(T_{k-1} = \begin{cases} T_k \setminus \{j\} & \text{if } x^i_k = R\delta_j, \\ T_k \cup \{j\} & \text{if } x^i_k = C\delta_j. \end{cases}\)

- \(\{w^{k,j}\}_{k=0,j=0}^{m,r}\), where each \(w^{k,j} = w_1^{k,j}w_2^{k,j} \cdots w_r^{k,j}\) is an assignment in \(\Sigma^r\), defined recursively as follows:
  - \(w_{m,0} = v_i^m\) for \(0 \leq i \leq r\);
  - for \(k < m\):
    - \(w^{k,0} = w^{k+1,r}\);
    - for \(0 < j \leq r\):
      \[w^{k,j} = \text{pref}_{j-1}(w^{k,j-1})w_j^{k,j}\text{suff}_{r-j}(w^{k+1,r}),\]
  where
  \[w_j^{k,j} = \begin{cases} v_j^i_k, & \text{if } j \notin T_k, \\ v_j^{i_{k'}} & \text{where } k' = \min\{k'' > k | j \notin T_{k''}\}, \text{if } j \in T_k.\end{cases}\]

To verify that this description is correct, we refer to the definition of \(\mu'\), transitions of types 5-9. The automaton \(A'\) reads the string
\[\alpha' = v^m_i x^m_i v^{m-1}_i x^{m-1}_i \cdots v^1_i x^1_i v^0_i \perp,\]
from the stack, symbol by symbol. The states \(q_{\text{load}}^{T_k,j}, 1 \leq i \leq r\), are responsible for reading \(v^i_k\) and loading each \(v^i_k\) into register \(j\) (moves of type 5). That is, with the exception of registers in \(T_k\) – these remain unchanged from the reading of \(v^{i_{k+1}}\), or, more specifically, from the reading of the last \(v^{i_{k'}}\), \(k' > k\), for which \(j\) was not in \(T_{k'}\). The automaton refrains from altering these registers, as, by part 1 of Proposition 37 below, \(T_k \subseteq S_i^k\) (moves of type 6 are employed). After reading \(v^i_k\), it arrives to state \(q_{\text{load}}^{T_k,r+1}\), and then \(q_{\text{R}}^k\) or \(q_{\text{C}}^k\), where \(x^i_k\) is read and \(T_{k-1}\) is calculated accordingly (moves of type 8, 9).
Thus, the description of $\{T_k\}_{k=0}^m$ and $\{w^{k,j}\}_{k=0,j=0}^m$ follows directly from the definition of transitions of types 5-9. The only auxiliary statement used in the above reasoning is part of the following proposition (its other statements will be of use further).

**Proposition 37.** We have:

1. $T_k \subseteq S_{i_k}$.
2. If $T_{k-1} = T_k \cup \{j\}$, then $j \notin T_k$.
3. $T_0 = \emptyset$.

**Proof.** The same as the proof of Proposition 32. \hfill \Box

Note, that it follows from the description of $w^{k,j}$s, that $w^{k,r}$ are defined by

$$w^{k,r}_j = \begin{cases} v^{i_k}_j, & \text{if } j \notin T_k, \\ v^{i'}_j, & \text{otherwise} \end{cases} \quad \text{where } k' = \min\{k'' > k | i \notin T_{k''}\}, \text{ if } j \in T_k.$$ 

Let us now define the sequence of instantaneous descriptions

$$c_0, c_1, c_2, \ldots, c_l,$$

that we claim to be an accepting run of $A$ on $\sigma$.

First, we choose arbitrarily $r$ symbols from $\Sigma \setminus [u_0 \sigma]$, which we denote by $\sigma'_1, \sigma'_2, \ldots, \sigma'_r$. These symbols will be used instead of blank symbols in the registers, which have been reset to simulate non-deterministic reassignment, but have not been filled before the halt of the computation. This corresponds to the situation when a guessed symbol has not been compared to an input symbol, thus, any symbol could be guessed, therefore, it is irrelevant which symbol we use in the computation sequence we construct, as long as the content of the registers remains an assignment. This condition is indeed satisfied: since none of the symbols we choose are in $[u_0 \sigma]$, none of them may appear in any of the components of the run of $A'$ on $\sigma$ due to the property of deterministic reassignment, stated in Proposition 5 of Chapter 2.

Now we define $c_i$ the sequence of instantaneous descriptions by writing $c_i = (q^i, u^i, \sigma^i), 0 \leq i \leq l$, where $u^i = u_1^i u_2^i \cdots u_r^i$ and

- for $0 \leq k \leq m$, $u^{ik}$ are defined by

$$u^{ik}_j = \begin{cases} \sigma_{j'}^{i_k}, & \text{if } j \in S_{i_k} \setminus T_k, \\ w^{k,r}_j, & \text{otherwise}. \end{cases}$$
whereas for $i_k < i < i_{k+1}$ ($k < m$) and $i_k < i \leq l$ ($k = m$), $u^i = u^i_k$.

Note that by the definition of $A'$ (move of type 4) and since the given run is an accepting run on $\sigma$, $q^l \in F$ and $\sigma^l = \epsilon$. Thus, if what we defined is indeed a run of $A$, then it is an accepting run. Now, since $q^0 = q_0$, to complete the proof it suffices to show the following.

**Proposition 38.**

1. $u^0 = u_0$.
2. For $0 \leq i < l$, $c_i \vdash_A c_{i+1}$.

**Proof.** See Appendix B. \qed

To summarize the proof of the inclusion $L(A') \subseteq L(A)$, given an accepting run of $A'$ on a word $\sigma \in L(A')$

$$d_{-1}, d_0, d_1, d_2, \ldots, d_i, e_1^m, e_2^m, \ldots, e_{r+2}^m, \ldots, e_1^0, e_2^0, \ldots, e_{r+2}^0$$

we have constructed a sequence of instantaneous descriptions $c_0, c_1, c_2, \ldots, c_l$ and shown it to be an accepting run of $A$ on $\sigma$, which is an evidence of $\sigma \in L(A)$.

Together with the fact $L(A) \subseteq L(A')$, shown previously, we conclude that NR-FMA $A$ and DR-IPDA$_n$ $A'$, constructed from $A$ as described in Section 3.2, indeed accept the same language.

This completes the proof of Theorem 13.
Appendix C

Auxiliary statements for Chapter 4

C.1 The equivalence of FSUBA and FSUBAₙ

The following propositions state that for each automaton there is an equivalent automaton that never replaces or erases symbols in its Θ-registers.

**Proposition 15.** For any FSUBA A there is an FSUBA A', such that L(A) = L(A'), and A' never empties its Θ-registers.

**Proposition 16.** For any FSUBAₙ A there is an FSUBAₙ A', such that L(A) = L(A'), and A' never reassigns its Θ-registers.

The Proposition 15 is very similar to [5, Lemma 5.1], thus, we omit the proof.

**Proof of Proposition 16.** Let $A = (\Sigma, Q, q_0, F, u, \Theta, \mu)$, where $u = u_1 u_2 \cdots u_r$, and let $S_\Theta = \{i | u_i \in \Theta\} = \{i_1, i_2, \ldots, i_m\}$. We define $A'$ to have $m$ additional registers, each new register $r + l$ simulating A’s register $i_l$ from $S_\Theta$, while $A'$’s registers from $S_\Theta$ are empty, until reassigned. After $i_l$ is reassigned, it is remembered in a state of $A'$, and from there on register $i_l$ of $A'$ simulates register $i_l$ of $A$. In this manner, the actual Θ-registers of $A'$ – its new registers – are never reassigned, while the language recognized by it is the same as the one of $A$.

Formally, $A' = (\Sigma, Q', q'_0, F', u', \Theta, \mu')$, where

$$Q' = \{q^s | q \in Q, s \subseteq S_\Theta\},$$
and \( \mu' \) is defined as follows. For each \((p, S, k, q) \in \mu\), and for each \( s \subseteq S_\Theta\), \( \mu' \) contains \((p^*, S, k', q^*) \in \mu\), where

\[
k' = \begin{cases} 
  r + 1, & \text{if } k = i \in S_\Theta \setminus s, \\
  k, & \text{otherwise,}
\end{cases}
\]

and

\[
s' = s \cup (S \cap S_\Theta).
\]

Note, that \( s' \neq s \) if and only if \( S \cap S_\Theta \neq \emptyset \), and \( S \cap S_\Theta \not\subseteq s \). Such transitions are applied when there are \( \Theta \)-registers in \( S \) which are reassigned for the first time during the computation. These registers are added to \( s \), which essentially is a list of altered \( \Theta \)-registers. When a transition register is a \( \Theta \)-register, the input is compared to the corresponding new register (which is never reassigned, since the reassignment set of each transition is a subset of \( \{1, 2, \ldots, r\} \)), unless it belongs to the list of altered registers. This follows from the definition of \( k' \), where the input is compared to the regular register, possibly after reassignment. The equivalence of \( \mathcal{A} \) and \( \mathcal{A}' \) follows easily from the definition by translating computation sequence of \( \mathcal{A} \) into one of \( \mathcal{A}' \) and vice-versa.

**Proposition 39.** The automata models FSUBA and FSUBA\(_n\) are equivalent.

**Proof.** Let \( \mathcal{A} \) be FSUBA\(_n\) as in Definition 14. By Proposition 16, we may assume that its \( \Theta \)-registers are never reassigned. We construct an \((r + 1)\)-register FSUBA \( \mathcal{A}' = (\Sigma, Q', q_0', F', u', \Theta, \mu') \) as in Definition 3, so that \( L(\mathcal{A}) = L(\mathcal{A}') \). Let \( P_{r+1} \) be a set of \( r + 1 \) dimensional vectors whose components are subsets of \( \{1, 2, \ldots, r\} \), such that the components of a vector are pairwise disjoint, and the union of the components of a vector is \( \{1, 2, \ldots, r\} \). Formally,

\[
P_{r+1} = \left\{ \mathbf{p} = (p_1, p_2, \ldots, p_{r+1}) \in (2^{\{1, 2, \ldots, r\}})^{r+1} \bigg| \forall \ i, j, \ p_i \cap p_j = \emptyset; \bigcup_{i=1}^{r+1} p_i = \{1, 2, \ldots, r\} \right\}
\]
In the constructed automaton each configuration \((q, w)\) of \(A\), where \(w = w_1 w_2 \cdots w_r\), is simulated by a configuration \((q^p, w')\) of \(A'\), \(w' = w'_1 w'_2 \cdots w'_{r+1}\), \(p = (p_1, p_2, \ldots, p_{r+1})\), in the following manner. We say that \((q^p, w')\) simulates \((q, w)\), if for each \(i\) such that \(p_i\) is not empty, the \(i\)th register of \(A'\) represents registers of \(A\) whose indices are in \(p_i\), meaning that for each \(j \in p_i\) \(w_j = w'_i\), and for each \(i\) such that \(p_i = \emptyset\), the \(i\)th register of \(A'\) is empty, i.e., \(w'_i = \#\).

To achieve the simulation, we formally define the elements of \(A'\) in the following manner.

- \(Q' = \{q^p\mid q \in Q, p \in P_{r+1}\}\).
- \(q'_0 = q_0^{\{1\}, \{2\}, \ldots, \{r\}, \emptyset}\).
- \(F' = \{q^p\mid q \in F, p \in P_{r+1}\}\).
- \(u' = u\#\). Note that \((q'_0, u')\) simulates \((q_0, u)\).
- For each \((q_1, S, k, q_2) \in \mu\) and each \(p' \in P_{r+1}\), \(\mu'\) contains the transition \((q'_1, k', S', q'_2)\), where \(k'\), \(S'\) and \(p'' = (p''_1, p''_2, \ldots, p''_{r+1})\) are defined as follows.
  - If \(k \in S\), then, by definition of computation of \(A\), and because \(A\) never reassigns its \(\Theta\)-registers, the transition is performed (after the \(k\)th register is reassigned with the input symbol) if and only if the input symbol is not in \(\Theta\). (If the input symbol is in \(\Theta\), the transition cannot be performed, because the content of \(k\)th register cannot possibly be a symbol of \(\Theta\), due to the special property of \(A\).) To produce the same behavior in \(A'\), we define \(k' = \min\{i\mid p'_i = \emptyset\}\), meaning, the transition register will be empty, hence, allowing to reassign it with the input symbol and perform the transition, unless the input symbol is in \(\Theta\). Note, that \(k'\) is well defined, since, by the definition of \(P_{r+1}\), at list one of the \(p''_i\)'s has to be empty. Since all the register in \(S\) are reassigned with input symbol, \(p''_k = S\), and for the rest of \(p''_i\), \(p''_i = p'_i \backslash S\). Finally, \(S' = \{i\mid p''_i = \emptyset\}\). Again, note that \(S' \neq \emptyset\), since at list one \(p''_i\) has to be empty.
  - If \(k \notin S\), then the transition of \(A\) can only be performed if the \(k\)th register already contains the symbol that is encountered in the input. To simulate this behavior in \(A'\), we define \(k'\) to be the
register that represents the register \( k \) of \( \mathcal{A} \). That is, \( k' = j \), such that \( k \in p_j' \).

* If \( k \) is not a \( \Theta \)-register, \( k' \) after the transition represents all the registers of \( \mathcal{A} \) reassigned with the input symbol. Therefore in this case we define \( p''_k = p'_k \cup S \), and for the rest of \( p''_i \)'s, \( p''_i = p'_i \setminus S \).

* On the other hand, if \( k \) is a \( \Theta \)-register, the registers in \( S \) will be emptied. Therefore, in \( \mathcal{A}' \) they must be represented by empty registers. Thus, let \( l = \min\{i|p'_i = \emptyset\} \), and we define \( p''_l = S \), and for the rest of \( p''_i \)'s, \( p''_i = p'_i \setminus S \).

As before, \( S' = \{i|p''_i = \emptyset\} \).

Since, as was noted, \((q'_0, u')\) simulates \((q_0, u)\), to prove the equivalence of the two automata, it suffices to show that if \((q'_1, w')\) simulates \((q_1, w)\), and \((q'_2, v')\) simulates \((q_2, v)\), then \(( (q_1, w), \sigma, (q_2, v) ) \in \mu^c \) if and only if \(( (q'_1, w'), \sigma, (q'_2, v') ) \in \mu'^c \).

This easily follows from the definition and the accompanying remarks.

Now, let \( \mathcal{A} = (\Sigma, Q, q_0, F, \Theta, \mu) \) be an \( r \)-register FSUBA. By Proposition 15, we may assume that it never empties its \( \Theta \)-registers. We construct an \( r \)-register FSUBA \( \mathcal{A}' = (\Sigma, Q', q'_0, F', u', \Theta, \mu') \) in the following manner.

We note, that a transition of \( \mathcal{A} \) depends on whether or not the transition register is empty. If it is, the transition is performed, unless the input symbol is in \( \Theta \); otherwise, it depends on the symbol contained in the register. Thus, \( \mathcal{A}' \) must keep track of the empty registers in its states. Each configuration \((q, w)\) of \( \mathcal{A} \), \( w = w_1w_2\ldots w_r \), is simulated by a configuration \((q^T, w')\) of \( \mathcal{A}' \), \( T \subseteq \{1, 2, \ldots, r\} \), \( w' = w'_1w'_2\ldots w'_r \). We say that \((q^T, w')\) simulates \((q, w)\), if for each \( i \in T \) \( w_i \) is empty, and for \( i \notin T \) \( w'_i = w_i \).

To achieve the simulation, we formally define the elements of \( \mathcal{A}' \) in the following manner. Let \( u = u_1u_2\ldots u_r \).

- \( Q' = \{ q^T | q \in Q, T \subseteq \{1, 2, \ldots, r\} \} \).
- \( q'_0 = q^T_0 \), where \( T_0 = \{i|u_i = \#\} \).
- \( F' = \{ q^T | q \in F, T \subseteq \{1, 2, \ldots, r\} \} \).
- \( u' = u \). Note, that \((q'_0, u')\) simulates \((q_0, u)\).
For each \((q_1, k, S, q_2) \in \mu\), and each \(T' \subseteq \{1, 2, \ldots, r\}\), \(\mu'\) contains the transition \((q_1^{T'}, S', k', q_2^{T''})\), where \(S', k'\) and \(T''\) are defined as follows.

- If \(k \in T'\), then the state \(q_1^{T'}\) describes the situation in \(A\) when the transition register is empty. Thus, the transition must be performed, independently of the specific input symbol, unless it is a symbol from \(\Theta\), in which case the transition can not be performed. In \(\mathcal{A}'\) this is achieved by including the transition register into the reassignment set, and ensuring that it does not contain a symbol from \(\Theta\). As it is shown below, for each configuration \((q^T, w)\) that may appear in a valid computation, \(w\) cannot contain symbols from \(\Theta\) in a register with index in \(T\). Hence we define \(k' = k\), \(S' = \{k'\}\), and \(T'' = (T' \setminus S') \cup S\).

- If \(k \notin T'\), then the state \(q_1^{T'}\) describes the situation in \(A\) when the transition register is not empty. This means, that in \(\mathcal{A}'\) the decision must be made before the reassignment, which is impossible unless the input is in \(\Theta\). Thus, we must exclude the transition register from the reassignment set. Hence, we define \(k' = k\), \(S' = \emptyset\) and \(T'' = T' \cup S\).

Now we show by induction on the length of computation sequence, that indeed, for each configuration \((q^T, w)\) that may appear in a valid computation, \(w\) never contains symbols from \(\Theta\) in a register with index in \(T\). At the beginning of the computation, \(T_0\) contains only empty registers of \(\mathcal{A}'\). Therefore, the condition holds for the initial configuration. At any time during a computation when \(p_2^{T''}\) is entered from \(p_1^{T'}\), the indices that may be added to \(T'\) to obtain \(T''\) are the registers emptied by \(A\). Since the latter never empties its \(\Theta\)-registers, and \(\Theta\)-registers of \(\mathcal{A}'\) are the same as \(\Theta\)-registers of \(\mathcal{A}\), a \(\Theta\)-register may not be added to \(T'\). That is, the condition is preserved during the computation.

Since \((q'_0, u')\) simulates \((q_0, u)\), to prove the equivalence of the two automata it suffices to show that if \((q_1^T, w^T)\) simulates \((q_1, w)\), and \((q_2^S, v^S)\) simulates \((q_2, v)\), then \(((q_1, w), \sigma, (q_2, v)) \in \mu\) if and only if \(((q_1^T, w^T), \sigma, (q_2^S, v^S)) \in \mu'\). This easily follows from the definition and the accompanying remarks.

\(\Box\)
C.2 Proof of Proposition 20

Proposition 20. The automata models NR-FSUBA and FSUBA are equivalent.

Proof. Naturally, we prove instead that NR-FSUBA is equivalent to FSUBA
\( n \).

Let \( L = L(A) \), where \( A = (\Sigma, Q, q_0, F, u, \Theta, \mu) \), \( u = u_1u_2\ldots u_r \), is an FSUBA which does not reassign its \( \Theta \)-registers (it exists by Proposition 16). We construct an \( r + 1 \)-register NR-FSUBA \( A' = (\Sigma \cup \{\#\}, Q', q'_0, F', u', \Theta \cup \{\#\}, \mu') \) that simulates \( A \) in the following manner.

We need the empty register symbol in the alphabet to properly simulate the original automaton, namely, to simulate the registers that are reset when reassignment is performed on input in \( \Theta \). On the other hand, we do not want it to appear in any of the accepted words, therefore we include it into the set of read-only symbols. Thus, we need to concatenate it to the initial assignment. Also, to make sure that the input is never compared to \( \# \), and the accepted language is indeed over \( \Sigma \), we add an indicator \( T \) of the set of registers currently marked as empty to each state.

In spite of the similarity in the operation of the two models, a problem for the construction arises when, for some transition \( (q, S, k, q') \), \( k \notin S \). If the input symbol is not in \( \Theta \), then, in the assignment obtained during such a transition in FSUBA \( n \), the \( k \)th register and the registers in \( S \) contain the same symbol (specifically, the input symbol), while in NR-FSUBA the symbol “guessed” into registers of \( S \) can be different. To solve this problem, we attach to each state a vector \( \bar{p} = (p_1, p_2, \ldots, p_{r+1}) \) from

\[
\bar{P}_{r+1} = \left\{ \bar{p} = (p_1, p_2, \ldots, p_{r+1}) \in (2^{\{1,2,\ldots,r\}})^{r+1} \left| \forall i, j, \ p_i \cap p_j = \emptyset; \bigcup_{i=1}^{r+1} p_i = \{1, 2, \ldots, r\} \right. \right\}
\]

so that a configuration \( ((q, T)^{\bar{p}}, w') \) of \( A' \), \( w' = w'_1w'_2\cdots w'_{r+1} \), simulates the configuration \( (q, w) \) of \( A \), \( w = w_1w_2\cdots w_r \), if and only if for all \( i, j \) such that \( i \in p_j \) (i.e., the \( j \)th register of \( A' \) simulates registers of \( A \) whose indices are in \( p_j \)), \( w_i = w'_j \), and \( T = \{i|w'_i = \#\} \).

Formally, the components of \( A' \) are defined as follows.

1. \( Q' = \{(q, T)^{\bar{p}}\}|q \in Q, T \subseteq \{1, 2, \ldots, r + 1\}, \bar{p} \in \bar{P}_{r+1}\} \).
2. \( q'_0 = (q_0, \{i|u_i = \#\} \cup \{r + 1\}) \).

\(^1\)This technique was also used in the proof of Proposition 17 on page 33.
\( F' = \{(q,T)\bar{p} | q \in F, T \subset \{1, 2, \ldots, r + 1\}, \bar{p} \in \bar{P}_{r+1}\} \).

- \( u' = u\# \). Note, that the initial configuration \((q'_0, u')\) simulates the initial configuration \((q_0, u)\).

- For each \((q_1, S, k, q_2) \in \mu\), each \(T_1 \subset \{1, 2, \ldots, r + 1\}\), and each \(\bar{p}_1 \in \bar{P}_{r+1}\), (with the exception below), there is a transition

\[ ((q_1, T_1)\bar{p}_1, S', k', (q_2, T_2)\bar{p}_2), \]

that belongs to \(\mu'\), where \(k'\), \(S'\), \(\bar{p}_2\) and \(T_2\) are defined as follows.

Let \(\bar{p}_1 = (p_1^1, p_2^1, \ldots, p_{r+1}^1)\), \(\bar{p}_2 = (p_1^2, p_2^2, \ldots, p_{r+1}^2)\), and let \(i\) be such that \(k \in p_i^1\). In addition, let \(\Phi = \{j | p_j^1 = \emptyset\}\).

- If \(k \in S\), the transition in \(A\) must be performed after reassigning the transition register together with the other registers in \(S\) with the symbol that appears in the input. If the input symbol is in \(\Theta\), the transition can not be performed, since the \(k\)th register is reassigned. Therefore, it cannot be a \(\Theta\)-register.
  
  * If \(p_i^1 = \{k\}\), and \(i \neq r + 1\), then we simply reassign the \(i\)th register of \(A'\), and update the interpretation vector so that from hereon it will simulate also the registers in \(S\). Formally, \(k' = i\), \(S' = \{k'\}\), \(p_k^2 = S\), and \(p_j^2 = p_j^1 \setminus S\) for \(j \neq k'\).
  
  * If \(p_i^1 = \{k\}\) and \(i = r + 1\), in which case the \(k\)th register is empty, or, if \(p_i^1 \neq \{k\}\), which means the \(i\)th registers is responsible for simulating other registers besides \(k\), we may not just reassign the \(i\)th register as its contents are needed during the further computation. Instead, we reassign some register that does not simulate any other registers at the moment. Thus, we define \(k' = \min \Phi\) (clearly \(\Phi\) is never empty), and \(S'\) and \(\bar{p}^2\) are defined as in the previous case.

In both cases \(T_2 = T_1 \setminus S'\).

Note, that we do not need to monitor the set of empty registers \(T\), since we reassign the transition register anyway. Besides, we never need to add empty registers to the set, because the reassignment symbol cannot be \(#\), as it is a read-only symbol in \(A'\). In addition, if the input symbol is in \(\Theta\), the transition in \(A'\) cannot be performed, like in \(A\), and for the same reason. Finally, note, that
in both cases, \( k' \leq r \). Thus, the register \( r + 1 \) of \( A' \), containing #, is not reassigned.

- If \( k \notin S \), the transition in \( A \) must be performed if and only if the input symbol equals to the one in the \( k \)th register, without reassigning it. After the transition the registers in \( S \) must also contain the input symbol, unless the symbol is read-only, in which case they must be emptied.

If \( i \in T \), it follows that the \( k \)th register contains #. In this case the transition cannot be performed, since we do not want empty register symbol in our language. Therefore, we do not include the transition for this case into \( \mu' \) (the exception, mentioned above).

We thus assume that \( i \notin T \).

* If \( k \) is not a \( \Theta \)-register in \( A \), the transition cannot be performed on a symbol from \( \Theta \). As follows from the note below, the \( i \)th register may not contain a symbol from \( \Theta \) (otherwise, \( p_1^i \) could not have contained \( k \)). Thus, in this case we define \( k' = i \), \( S' = \emptyset \), \( p_2^i = p_1^i \cup S \), and for \( j \neq i \), \( p_j^2 = p_j^1 \setminus S \).

* If \( k \) is a \( \Theta \)-register, the registers in \( S \) must be emptied. Therefore in the resulting configuration they are simulated by the \((r + 1)\)st register containing #. Thus, in this case we define \( k' = i \), \( S' = \emptyset \), \( p_{r+1}^2 = p_{r+1}^1 \cup S \), and for \( j \neq r + 1 \), \( p_j^2 = p_j^1 \setminus S \).

Since in both cases \( S' = \emptyset \), none of the registers is actually reassigned, \( T_2 = T_1 \) and \((r + 1)\)st register still contains # after the transition.

Note, that from the above definitions it easily follows by induction that for any vector \( \bar{p} = (p_1, p_2, \ldots, p_{r+1}) \in \bar{P}_{r+1} \) that may appear in a valid computation, the following holds. For each \( \Theta \)-register \( j \), \( p_j = \{j\} \).

Since \((q'_0, u')\) simulates \((q_0, u)\), to prove the equivalence of the two automatons it suffices to show that if \(((q_1, T_1)^{\bar{p}}, w')\) simulates \((q_1, w)\), and \(((q_2, T_2)^{\bar{p}}, v')\) simulates \((q_2, v)\), then \(((q_1, w), \sigma, (q_2, v)) \in \mu^c \) if and only if

\[ (((q_1, T_1)^{\bar{p}}, w'), \sigma, ((q_2, T_2)^{\bar{p}}, v')) \in \mu'^c. \]

It easily follows from the definition, considering the remarks accompanying it.
Now let $L = L(A)$ where $A = (\Sigma, Q, q_0, F, u, \Theta, \mu)$, $u = u_1u_2 \cdots u_r$, is NR-FSUBA that does not reassign its $\Theta$-registers (it exists by Proposition 19). We construct an FSUBA$_n$ $A' = (\Sigma \cup \{\#\}, Q', q'_0, F', u', \Theta \cup \{\#\}, \mu')$ that simulates $A$ and accepts $L$ in the following manner.

To simulate $A$, we attach to the states of $A'$ equality constraint sets, similar to those used in the definition of $nM$-grammar in Section 4.4. Let $P_r$ be a subset of $2^{\{1,2,\ldots,r\}}$, such that for each $p = \{p_1, p_2, \ldots, p_n\} \in P_r$, $p_i \cap p_j = \emptyset$ for all $i, j$. We call such $p$ an equality constraint set (shortly ECS), and say that configuration $(q^p, w')$ of $A'$ simulates the configuration $(q, w)$ of $A$, if for each $i \in \{1, 2, \ldots, r\}$ the following holds.

If $i \notin \bigcup p$, then $w'_i = w_i$, and for each $p_k \in p$, for all $i, j \in p_k$, $w_i = w_j$.

This means, that for registers not appearing in ECS, the $i$th register of $A$ simulates the same register of $A'$, and the contents of the registers appearing in the ECS are irrelevant. It is only known that two registers belonging to one equality constraint simulate registers that contain the same symbol in $A$. The intuitive meaning of $p_k$ is that during the computation of $A$, a set of registers $S$ such that $p_k \subseteq S$, was “reassigned by guessing”, and $p_k$ is the set of registers that have not been reassigned again since then. Naturally, we will ignore any empty sets in ECSs.

Formally, we define the components of $A'$ as follows.

- $Q' = \{q^p| q \in Q, p \in P_r\}$.
- $q'_0 = q_0^\emptyset$.
- $F' = \left\{ q^p \bigg| \begin{array}{l} p \in P_r \land (q \in F \lor \\
\exists q_1, q_2, \ldots, q_n \in Q : q_1 = q \land q_n \in F \land \\
\exists S_1, S_2, \ldots, S_{n-1} \subseteq \{1, 2, \ldots, r\} : \\
\forall i \in \{1, 2, \ldots, n-1\}, (q_i, S_i, \epsilon, q_{i+1}) \in \mu \}
\end{array} \right\}$. That is, $F'$ is an $\epsilon$-closure of $F$ with all possible ECS.
- $u' = u$.
- For each sequence of states $q_1, q_2, \ldots, q_n$ and for each $k \in \{1, 2, \ldots, r\}$, such that for some sequence $S_1, S_2, \ldots, S_{n-1}$ of subsets of $\{1, 2, \ldots, r\}$, $(q_i, S_i, \epsilon, q_{i+1}) \in \mu$ for $i = \{1, 2, \ldots, n-2\}$, and $(q_{n-1}, S_{n-1}, k, q_n) \in \mu$, and for each ECS $p^1 \in P_r$, $\mu'$ contains a transition $(q^p_1, S, k, q^p_n)$, where $S$ and $p^2$ are defined as follows.

First, we define a sequence $\tilde{p}^1, \tilde{p}^2, \ldots, \tilde{p}^{n-1}$ of ECSs by
Then $\tilde{p} \overset{\text{def}}{=} \tilde{p}^{n-1}$ is a set of constraints resulting from subsequent reassignment of register sets $S_1, S_2, \ldots, S_i$.

Let us consider the sequence of $A$’s computation steps resulting from subsequent application of the transitions listed above (transitions $1$ to $n-1$). As a cumulative result of these steps, each set $p \in \tilde{p}$ of registers is assigned with some symbol from $\Sigma \setminus \Theta$.

- If there is a $p \in \tilde{p}$ such that $k \in p$, then, unless the input symbol is in $\Theta$, all the transitions in the sequence can be performed, including the last one, involving comparison of the input to the content of the $k$th register. Indeed, if the symbol assigned to the registers in $p$ is guessed correctly, it is the input symbol at the $(n-1)$th step. The same result can be obtained in $A'$ by the means of deterministic reassignment. Namely, $S = p$ and $p^2 = \{p' \setminus \cup \tilde{p} \mid p' \in p^1\} \cup \tilde{p} \setminus \{p\}$. (The registers reassigned during steps $1$ to $n-1$ are removed from constraints in $p^1$, and the constraints resulting from steps $1$ to $n-1$, except $p$, are added. This is because $A'$’s registers in $p$ contain the same symbol as the corresponding registers in $A$.)

Note, that if the input symbol is in $\Theta$, the transition cannot be performed in both automata. The $k$th register, being reassigned in $A$, thus, not being a $\Theta$-register, may not contain a symbol from $\Theta$ in both automata, either before, or after the reassignment.

- If $k \notin \cup \tilde{p}$, then the $k$th register is not reassigned during the sequence of $A$’s steps, and the $(n-1)$th step may be performed only if the input symbol equals the symbol in $k$th register before the first step. In $A'$ this situation is simulated as follows. If $k \notin \cup p^1$, then $S = \emptyset$, and $p^2 = \{p' \setminus \cup \tilde{p} \mid p' \in p^1\} \cup \tilde{p}$.

If, for some $p \in p^1$, $k \in p$, then $S = p \setminus \cup \tilde{p}$, and $p^2 = \{p' \setminus \cup \tilde{p} \mid p' \in p^1, p' \neq p\} \cup \tilde{p}$.

Note, that in the latter case, the $k$th register cannot be a $\Theta$-register in $A$, as it belongs to some ECS. Therefore, if the input symbol is in $\Theta$, the transition cannot be performed in both automatoms.

87
To prove that $A$ and $A'$ accept the same language, it suffices to show the following.

1. $(q'_0, u')$ simulates $(q_0, u)$.

2. If $A'$'s configurations $c', d'$ simulate $A$'s configurations $c, d$, respectively, then $(c', \sigma, d') \in \mu^c$ if and only if there exists a sequence of configurations $c_1, c_2, \ldots, c_n, n \geq 2$, such that $c_1 = c$, $c_n = d$, $(c_i, \epsilon, c_{i+1}) \in \mu$ for $i = 1, 2, \ldots, n-2$, and $(c_{n-1}, \sigma, c_n) \in \mu$.

3. A configuration $c'$ of $A'$ is in $F'$ if and only if there exists a sequence of $A$'s configurations $c_1, c_2, \ldots, c_n$, $n \geq 1$, such that $c'$ simulates $c_1, c_n \in F^c$, and $(c_i, \epsilon, c_{i+1}) \in \mu$ for $i = 1, 2, \ldots, n-1$.

All three statements follow directly from the definition of $A'$ and the accompanying remarks. 

\[ \square \]

C.3 Proof of Theorem 24: the formal construction and the proof of correctness

Proposition 26. For any UBCFG $G$ there is a UBCFG $G'$ that “never guesses into its $\Theta$-registers”, such that $L(G) = L(G')$.

Proof. Let $G = (V, u, \Theta, R, V_0)$, where $u = u_1 u_2 \cdots u_r$, and let $S_\Theta = \{i | u_i \in \Theta\} = \{i_1, i_2, \ldots, i_m\}$. We define $G'$ to have $m$ additional registers, each new register $r + l$ simulating a register $i_l$ from $S_\Theta$, until a production of $G$ is applied that alters $i_l$. At this point $i_l$ “is remembered by a variable of $G'$”, and starting from this derivation register $i_l$ of $G'$ simulates $i_l$ of $G$. In this manner, the actual $\Theta$ registers of $G'$ – its new registers – are never altered, while the set of derivable words is the same as the one of $G$.

Formally, $G' = (V', u', \Theta, R', V'_0)$, where

\[
V' = \{ A^s | A \in V, s \subseteq S_\Theta \},
\]

\[
V'_0 = V_0^\emptyset,
\]

\[
u' = u'_1 u'_2 \cdots u'_r u_{i_1} u_{i_2} \cdots u_{i_m},
\]

\[
u'_i = \begin{cases} u_i, & \text{if } u_i \notin \Theta, \\ \sigma_0, & \text{otherwise,} \end{cases}
\]
and $R'$ is defined as follows. For each $(A, S) \rightarrow a_1 a_2 \cdots a_k \in R$, and for each $s \subseteq S_\Theta$, $R'$ contains $(A^*, S) \rightarrow a'_1 a'_2 \cdots a'_k$, where

$$a'_j = \begin{cases} 
B^s, & \text{if } a_j = B \in V, \\
r + l, & \text{if } a_j = i_j \in S_\Theta \setminus s', \\
a_j, & \text{otherwise},
\end{cases}$$

and

$$s' = s \cup (S \cap S_\Theta).$$

Note, that $s'$ is only different from $s$ when $S \cap S_\Theta \neq \emptyset$, and $S \cap S_\Theta \not\subseteq s$. Such productions correspond to the steps in a derivation sequence when there are $\Theta$-register in $S$ whose content is guessed for the first time during the derivation. These registers are added to $s$, that essentially is a list of altered $\Theta$-registers. When a terminal is produced from a $\Theta$-register, it is produced from a new register, unless it has been altered, i.e. belongs to the list, as seen from the definition of $a'_j$. The equivalence of $G$ and $G'$ follows easily from the definition by translating derivations of $G$ into derivations of $G'$ and vice-versa.

Let $G = (V, u, \Theta, R, V_0)$, where $V = \{V_0, V_1, \ldots, V_m\}$ and $u = u_1 u_2 \cdots u_r$, be a UBCFG over $\Sigma$. By Proposition 26, assume $G$ never guesses into its $\Theta$-registers.

We define a NR-UBPDA $A = (Q, q_0, u', \Theta', \rho, \mu)$ over $\Sigma' = \Sigma \cup V$, where $\Theta' = \Theta \cup V$, as follows. Let $r' = r + |V| + 1$, $V = V_m V_{m-1} \cdots V_1 V_0$, and $S_\Theta = \{i = 1, 2, \ldots, r | u_i \in \Theta\}$. Note, that by Proposition 26 of $G$, for any $S$ that appears in a production of $G$, we have $S \cap S_\Theta = \emptyset$.

- $u' = u'_1 u'_2 \cdots u'_{r'} = uu_1 V$. The first $r$ and the last $|V|$ registers of $A$ simulate the registers and the variables of $G$ accordingly.
- $Q = \{q_0, p\} \cup \{q_{i,V_j,S} | i = 1, 2, \ldots, r + 1, V_j \in V, S \subseteq \{1, 2, \ldots, r\}\}$.

The purpose of each state is as follows:

- the state $q_0$ serves as the initial state of $G$;
- the state $p$ serves for guessing a terminal, and then checking if the symbol at the top of the stack is a terminal or a variable;

---

2The idea of the construction is presented on page 39.
* in the former case, and only if symbol from the stack and one from the input are equal (to the guessed symbol), they are "reduced" ("shift" + "pop");
* in the latter case, if the variable encountered at the top of the stack is $V_j$, then $A$ moves non-deterministically into one of the states $q_{V_j}^{S}$;
  - the states $q_{V_j}^{S}$, $i = 1, 2, \ldots, r$, are responsible for loading the $i$-th element of the assignment from the stack into the $i$-th register;
  - the states $q_{V_j}^{S}$ simulate the derivation of the grammar by performing the reassignment of the registers in $S$ and then placing the generated string into the stack.

- The transition relation is comprised of the following transitions (we define the reassignment function together with the transition relation):

1. $\rho(q_0) = \emptyset$, and $\mu(q_0, \epsilon, r') = \{(p, r'12 \cdots r)\}$;
2. $\rho(p) = r + 1$, and
   - $\mu(p, r + 1, r + 1) = \{(p, \epsilon)\}$;
   - $\mu(p, i, i) = \{(p, \epsilon)\}$, for all $i \in S_\Theta$;
   - $\mu(p, \epsilon, r' - j) = \{(q_{V_j}^{S}, \epsilon) \mid \exists a \ (V_j, S) \rightarrow a \in R\}$, for each $j = 0, 1, \ldots, m$;
3. for all $V_j \in V$, $S \subseteq \{1, 2, \ldots, r\}$,
   - $\rho(q_{V_j}^{S}) = \{i\}$, for all $i \in \{1, 2, \ldots, r\} \setminus S_\Theta$,
   - $\rho(q_{V_j}^{S}) = \emptyset$, for all $i \in S_\Theta$,
   - $\rho(q_{r+1}^{S}) = S$;

and

- $\mu(q_{V_j}^{S}, \epsilon, i) = \{(q_{i+1}^{S}, \epsilon)\}$, $i = 1, 2, \ldots, r - 1$;
- $\mu(q_{V_j}^{S}, \epsilon, r') = \{(q_{r+1}^{S}, r')\}$;
- $\mu(q_{r+1}^{S}, \epsilon, r') = \{(p, a'_1a'_2 \cdots a'_k) \mid (V_j, S) \rightarrow a_1a_2 \cdots a_k \in R\}$, where $a'_1, a'_2, \ldots, a'_k \in \{1, 2, \ldots, r'\}$ are defined by
  $$a'_i = \begin{cases} a_i, & \text{if } a_i \in \{1, 2, \ldots, r\}, \\ (r' - l)12 \cdots r, & \text{if } a_i = V_l, \end{cases}$$
  for $i = 1, 2, \ldots, k$.  


For a string \( X \in (\Sigma \cup (V \times \Sigma^*))^* \), let \( \bar{X} \) denote a string in \( \Sigma^* \) which is obtained form \( X \) by substituting all elements of the form \((V_i, w) \in V \times \Sigma^r \) with \( V_iw \in V\Sigma^r \).

**Lemma 40.** Let \((V_0, u) \Rightarrow L \sigma X^3, \) where \( \sigma \in \Sigma^* \) and \( X \in (V \times \Sigma^*)(\Sigma \cup (V \times \Sigma^*))^* \cup \{\epsilon\}. \) Then for some \( v \in \Sigma^r+1, \) \((p, u', \sigma, V_0u) \vdash^* (p, vV, \epsilon, \bar{X}), \) where \( \bar{X} \in \Sigma^* \) is obtained form \( X \) by substituting all elements of the form \((V_i, w) \in V \times \Sigma^r \) with \( V_iw \in V\Sigma^r \).

**Proof.** The proof is by induction on the length \( n \) of a leftmost derivation of \( \sigma X \) from \((V_0, u)\). If \( n = 0 \), then \( \sigma = \epsilon, \) and \( X = (V_0, u) \). Therefore, for \( v = uu_1, (p, u', \sigma, V_0u) \vdash^0 (p, vV, \epsilon, \bar{X}), \) where \( \bar{X} = V_0u. \)

Assume that the lemma holds for derivations of length \( n \) and prove it for derivations of length \( n + 1 \). Let \((V_0, u) = Y_0 \Rightarrow L Y_1 \Rightarrow L \cdots \Rightarrow L Y_n \Rightarrow L Y_{n+1} = \sigma X \) be a leftmost derivation of \( \sigma X \) from \((V_0, u)\). Clearly, \( Y_n \) contains a variable. Let \( Y_n = \sigma'(V_j, w)X' \), where \((V_j, w) \Rightarrow \sigma''X'', \sigma', \sigma'' \in \Sigma^*, \) \( \sigma = \sigma' \sigma'' \), and \( X = X''X' \).

By the induction hypothesis, for some \( v \in \Sigma^r+1, \)

\[(p, u', \sigma', V_0u) \vdash^* (p, vV, \epsilon, V_jwX').\]

Let \( v = v_1v_2 \cdots v_{r+1}, \) \( w = w_1w_2 \cdots w_r, \) and let \((V_j, w) \Rightarrow \sigma''X'' \) by the production \((V_j, S) \rightarrow a_1a_2 \cdots a_k. \) Then, there is a transition of type 2 \((q_1^{V_j,S}, \epsilon) \in \mu(p, \epsilon, r' - j), \) and since \( V = V_mV_{m-1} \cdots V_0, \) we have

\[(p, v_1v_2 \cdots v_{r+1}V, \epsilon, V_jwX') \vdash (q_1^{V_j,S}, v_1v_2 \cdots v_{r+1}V, \epsilon, wX').\]

Then, by transitions of type 3, we have:

\[(q_1^{V_j,S}, v_1v_2 \cdots v_{r+1}V, \epsilon, w_1w_2 \cdots w_r\bar{X}') \vdash (q_2^{V_j,S}, w_1v_2 \cdots v_{r+1}V, \epsilon, w_2 \cdots w_r\bar{X}') \]

\[\vdash \cdots \vdash (q_r^{V_j,S}, w_1w_2 \cdots w_{r+1}V, \epsilon, w_r\bar{X}')\]

by the transitions \( \mu(q_i^{V_j,S}, \epsilon, i) = \{q_i^{V_j,S}\}, i = 1, 2, \ldots, r - 1, \) and

\[(q_r^{V_j,S}, w_1w_2 \cdots w_{r+1}V, \epsilon, w_r\bar{X}') \vdash (q_{r+1}^{V_j,S}, w_{r+1}V, \epsilon, V_0\bar{X}'),\]

\[^3\text{Here and hereafter the subscript} \ L \text{refers to leftmost derivations.}\]
by the transition $\mu(q_\iota^{V_jS}, \epsilon, r) = \{(q_\iota^{V_jS}, r')\}$.

Note, that before moving from $q_\iota^{V_jS}$ to $q_{i+1}^{V_jS}$ the $i$th register is reassigned to guess the value of $w_i$, if and only if $i \notin S_\Theta$. Otherwise, no guessing is needed, since by the property of $G$ we know that $w_i = v_i = u_i$ for $i \in S_\Theta$. Finally, there is a transition $(p, a'_1 a'_2 \cdots a'_k) \in \mu(q_{i+1}^{V_jS}, \epsilon, r')$ of type 3 that corresponds to the production $(V_j, S) \rightarrow a_1 a_2 \cdots a_k$, where $a'_1, a'_2, \ldots, a'_k$ are obtained from $a_1, a_2, \ldots, a_k$ as described in the definition of $\mu$. By this transition we have

$$(q_{i+1}^{V_jS}, \omega'_{r+1} V, \epsilon, V_0 \tilde{X}') \vdash (p, \omega'_{r+1} V, \epsilon, \tilde{Z} \tilde{X}') = (p, \omega'_{r+1} V, \epsilon, \sigma'' \tilde{X}'', \tilde{X}'),$$

where $\omega'$ is obtained from $\omega$ by replacing symbols with indices in $S$ with the guessed symbol from $\Sigma \setminus \Theta$, and $\tilde{Z} = \sigma'' \tilde{X}''$. This readily follows from the fact that $(V_j, \omega) \Rightarrow \sigma'' \tilde{X}''$ by the production $(V_j, S) \rightarrow a_1 a_2 \cdots a_k$.

From here, by applying the transitions of type 2 $\mu(p, r+1, r+1) = \{(p, \epsilon)\}$ and $\mu(p, i, i) = \{(p, \epsilon)\}$ for $i \in S_\Theta$, we get

$$(p, \omega'_{r+1} V, \sigma'', \sigma'' \tilde{X}'', \tilde{X}') \vdash (p, \omega''_{r+1} V, \epsilon, \tilde{X}'').$$

Clearly, $\tilde{X}'' \tilde{X}' = \tilde{X}$. Thus, by combining the above computation steps of $A$ together, we obtain

$$(p, u', \sigma, V_0 u) \vdash^* (p, v V, \sigma'', V_j \tilde{X}') \vdash^* (p, \omega''_{r+1} V, \epsilon, \tilde{X}),$$

which completes the proof of the lemma.

Now, for any string in $(\Sigma^I \cup (V \Sigma^r))^*$ denoted by $\tilde{X}$, let $X$ denote a string in $(\Sigma \cup (V \times \Sigma))^*$, obtained from $\tilde{X}$ by substituting all substrings of the form $V_i\omega, \omega \in \Sigma^r$, with $(V_i, \omega) \in V \times \Sigma^r$.

**Lemma 41.** If for some $\omega \in \Sigma^{r+1}$, $(p, u', \sigma, V_0 u) \vdash^* (p, v V, \epsilon, \tilde{X})$, where $\sigma \in \Sigma^*$ and $\tilde{X} \in \Sigma^r$, then $X \in (\Sigma^I \cup (V \Sigma^r))^*$, and $(V_0, u) \Rightarrow^*_1 \sigma X$, where $X \in (\Sigma \cup (V \times \Sigma))^*$ is obtained from $\tilde{X}$ by substituting all substrings of the form $V_i\omega, \omega \in \Sigma^r$, with $(V_i, \omega) \in V \times \Sigma^r$.

**Proof.** The proof is by induction on the number $n$ of the appearances of $p$ in the computation of $(p, v V, \epsilon, \tilde{X})$ from $(p, u', \sigma, V_0 u)$. Let $n = 1$, i.e. $(p, u', \sigma, V_0 u) = (p, v V, \epsilon, \tilde{X})$. Then $\sigma = \epsilon$ and $\tilde{X} = V_0 u$. Hence $X \in (\Sigma^I \cup (V \Sigma^r))^*$, $X = (V_0, u)$ and indeed $(V_0, u) \Rightarrow^*_1 \sigma X$. 

92
Assume that the lemma holds for $n$ and prove it for $n + 1$. Let
\[(p, u', \sigma, V_0u) \vdash^* (p, vV, \sigma, \hat{X}) \vdash^* (p, v'V, \epsilon, \hat{Y})\]
be a computation containing $n + 1$ appearances of $p$, where the computation of $(p, vV, \sigma, \hat{X})$ contains $n$ appearances of $p$. Thus, the computation $(p, u', \sigma, V_0u) \vdash^* (p, vV, \epsilon, \hat{X})$ contains $n$ appearances of $p$ as well. By the induction hypothesis, $(V_0, u) \Rightarrow^*_L \sigma \hat{X}$.

It follows from the definition of $\mu$, that there are two possibilities for properly computing $(p, v'V, \epsilon, \hat{Y})$ from $(p, vV, \sigma, \hat{X})$ without additional appearances of $p$.

- The computation is of length 1, by a single transition of type 2. This computation involves reading one symbol from the input. Thus, $\sigma \in \Sigma$. In this case $\hat{X} = \sigma \hat{Y}$. Since, by the induction hypothesis, $\hat{X} \in (\Sigma^1 \cup (V\Sigma^r))^*$, it follows that $\hat{Y} \in (\Sigma^1 \cup (V\Sigma^r))^*$ as well, and $X = \sigma Y$. Thus, $(V_0, u) \Rightarrow^*_L \sigma X = \sigma \sigma Y$, as required to show.

- The computation is of length $r + 2$, by transitions of both type 2 and type 3, and all of the transitions are $\epsilon$-moves. In this case, $\sigma = \epsilon$, $\hat{X} = V_j w \hat{X}'$ and $\hat{Y} = X'' X'$, where $X''$ is such that $(V_j, w) \Rightarrow X''$. Thus, $\hat{Y} \in (\Sigma^1 \cup (V\Sigma^r))^*$, $Y = X'' X'$, and $(V_0, u) \Rightarrow^*_L \sigma X \Rightarrow X'' X' = \sigma \sigma Y$, as required to show.

Since $(q_0, u', \sigma, V_0u) \vdash (p, u', \sigma, V_0u)$, by transition of type 1, Theorem 24 follows from the above lemmas with $X = \hat{X} = \epsilon$. In particular, the inclusion $L(G) \subseteq L(A)$ follows from Lemma 40, and the inclusion $L(A) \subseteq L(G)$ follows from Lemma 41.

### C.4 Proof of Theorem 28

The proof is in many ways similar to the proof of [1, Theorem 3], but features some significant differences, thus, we present it for the sake of completeness.

#### C.4.1 Simple $nM$-grammars

**Definition 42.** An $nM$-grammar is called *simple* if its finite alphabet $\Delta$ contains exactly one symbol, say $\delta$, that appears at most one time in each
production, and all the productions containing $\delta$ are of the form $(A, p) \rightarrow (B, 1 \cdot \delta \cdot r)$.

**Lemma 43.** $nM$-languages are generated by simple $nM$-grammars.

**Proof.** Let $G = (V, u, \Theta, \Delta, R, A_0)$ be an $r$-register $nM$-grammar, where $\Delta = \{\delta_1, \delta_2, \ldots, \delta_m\}$. We shall construct an $(r + m)$-register simple $nM$-grammar $G'$, such that the $(r + k)$th register of $G'$ simulates $\delta_k$, $k = 1, 2, \ldots, m$, in the following manner. At each stage of derivation of $G'$ we first reassign, step by step, each $(r + k)$th register, $k = 1, 2, \ldots, m$, with the “value” of $\delta_k$. After that we reassign the first $r$ registers by referring to the last $m$ registers which correspond to the values of $\delta_k$’s.

Formally, consider $G' = (V', u', \Theta, \{\delta\}, R', A_{(1)}^0) – an (r + m)$-register $nM$-grammar, that is defined as follows.

- $V' = V \times \{1, 2, \ldots, m + 1\}$. The pair $(A, k)$ will be denoted by $A^{(k)}$.
- $u' \in \Sigma^{r+m}$ contains $u$ as a prefix.
- $R'$ consists of the following two types of productions:

1. For each $A \in V$, each $r = 1, 2, \ldots, m$, and each ECS\(^4\) $p \in P_r+m$, $R'$ contains the production

$$(A^{(k)}, p) \rightarrow (A^{(k+1)}),$$

$$12 \cdots r(r+1) \cdots (r+k-1) \delta(r+k+1) \cdots (r+m-1)(r+m)).$$

2. For each production $(A, p) \rightarrow a_1a_2 \cdots a_n \in R$, $R'$ contains the productions $(A^{(m+1)}, p) \rightarrow a'_1a'_2 \cdots a'_n$, such that $a'_i$, $i = 1, 2, \ldots, n$, is defined as follows.

- If $a_i \in \{1, 2, \ldots, r\}$, then $a'_i = a_i$.
- If $a_i = \delta_k \in \Delta$, then $a'_i = r + k$.
- If $a_i = (B, b_1b_2 \cdots b_r)$, then $a'_i = (B^{(1)}, c_1c_2 \cdots c_{r+m})$, where $c_j = j$ for $j > r$, and for $j \leq r$

$$c_j = \begin{cases} b_j, & \text{if } b_j \in \{1, 2, \ldots, r\}, \\ r + k, & \text{if } b_j = \delta_k. \end{cases}$$

Note, that the new production features the same ECS as the original, since the only equality constraints relevant are over the first $r$ registers.

\(^4\)Recall that ECS stands for equality constraint set, introduced on page 40.
Let $X = X_1X_2\cdots X_n \in (\Sigma \cup (V \times \Sigma^r))^*$ and let $X' = X'_1X'_2\cdots X'_n \in (\Sigma \cup (V' \times \Sigma^{r+m}))^*$. We say that $X'$ is an extension of $X$ if for each $i = 1, 2, \ldots, n$, the following holds. If $X_i \in \Sigma$, then $X'_i = X_i$, and if $X_i = (A, w) \in V \times \Sigma^r$, then $X'_i = (A^{(i)}, ww')$, for some $w' \in \Sigma^m$.

Since $u$ is a prefix of $u'$, to prove the lemma it suffices to show that $(A, w) \xrightarrow{G} X'$ if and only if $(A^{(1)}, ww') \xrightarrow{G'} X'$ for some extension $X'$ of $X$.

The latter equivalence easily follows by induction on the length of derivation in $G$ and $G'$: each step $(A, w) \xrightarrow{A} X$ can be “stretched” to

$$(A^{(1)}, ww_0) \xrightarrow{G'} (A^{(2)}, ww_1) \xrightarrow{G'} \cdots \xrightarrow{G'} (A^{(m+1)}, ww_m) \xrightarrow{G'} X',$$

where $f_k(\delta) = f(\delta_k)$, $k = 1, 2, \ldots, m$, and $X'$ is an extension of $X$. Conversely, each sequence of steps

$$(A^{(1)}, ww_0) \xrightarrow{G} (A^{(2)}, ww_1) \xrightarrow{G} \cdots \xrightarrow{G} (A^{(m+1)}, ww_m) \xrightarrow{G} X',$$

where $X'$ is an extension of $X$ and $f(\delta_k) = f_k(\delta)$, $k = 1, 2, \ldots, m$, can be “contracted” to $(A, w) \xrightarrow{G} X$. \hfill \Box

C.4.2 Proof of Theorem 28: constructing UBCFG from simple nM-grammar

We proceed to the proof of Theorem 28. Let $G$ be an $nM$-grammar. By Lemma 43, we may assume that $G = (V, u, \Theta, \{\delta\}, R, A_0)$, $u = u_1u_2\cdots u_r$, is simple. We shall construct an UBCFG $G'$ that simulates $G$ as described below.

The construction

First, we observe, that during each step of a derivation by a simple $nM$-grammar, the assignment may be changed either by altering one single register, or by “rearranging” the terminals in the assignment (possibly leaving some of them out). On the other hand, during one step of a derivation by a UBCFG, the only change that is allowed is resetting the value of a certain set of registers to a new terminal, no “rearranging” can be performed. In addition, applying a production of $G$ is only allowed if the assignment conforms to a certain ECS, while no mechanism for this is available in UBCFG.
We shall simulate both the “rearrangement” of assignment terminals and the ECSs by means of attaching an interpretation vector in $G'$ to each variable of $G$. We shall call interpretation vectors the elements of

\[ \bar{P}_r = \left\{ \bar{p} = (p_1, p_2, \ldots, p_r) \in (2^{\{1,2,\ldots,r\}})^r \left| \bigcup_{i=1}^{r} p_i = \{1,2,\ldots,r\}, \bigcap_{i,j} p_i \cap p_j = \emptyset \text{ for all } i,j \in \{1,2,\ldots,r\} \right. \right\}. \]

Let $v = v_1 v_2 \cdots v_r$, $w = w_1 w_2 \cdots w_r \in \Sigma^r$, $[v] \subseteq [w]$, and let $\bar{p} = (p_1, p_2, \ldots, p_r) \in \bar{P}_r$. We say that $w$ is an interpretation of $v$ by $\bar{p}$, if for each $i \in \{1,2,\ldots,r\}$, for each $j \in p_i$, $v_j = w_i$. Thus, for any interpretation vector $\bar{p}$ and any assignment $w$, there is a unique assignment $v$, such that $w$ is its interpretation by $\bar{p}$: for each $j \in \{1,2,\ldots,r\}$, $v_j = w_i$, such that $j \in p_i$. We also say that $w$ is a proper interpretation of $v$ by $\bar{p}$, if for each pair $i,j \in \{1,2,\ldots,r\}$ such that $v_i = v_j$ there exists $k \in \{1,2,\ldots,r\}$ such that $i,j \in p_k$. That is, that for any symbol in $[v]$, there is a single register in $w$ that interprets all the registers of $v$ containing that symbol.

We say that an interpretation vector $\bar{p} = (p_1, p_2, \ldots, p_r) \in \bar{P}_r$ conforms to ECS $p'$, if for each $p' \in p'$ there is a $j \in \{1,2,\ldots,r\}$, such that $p' \subseteq p_j$. Clearly, if $\bar{p}$ conforms to $p'$, then any assignment interpreted by $\bar{p}$ conforms to $p'$. That is, if $w$ is an interpretation of $v$ by $\bar{p}$, then $v$ conforms to $p'$. In addition, if an assignment is interpreted properly by $\bar{p}$, then $\bar{p}$ conforms to any ECS that the assignment conforms to. Indeed, for any equality constraint $p'$ that holds for $v$, if $w$ is a proper interpretation of $v$ by $\bar{p}$, then for some $i$ $p \subseteq p_i$.

Now we define $G' = (V', u, \Theta, R', A'_0)$ as follows.

- $V' = V \times \bar{P}_r$.
- $A'_0 = (A_0, \bar{P}^0)$, where $\bar{P}^0 = (p^0_1, p^0_2, \ldots, p^0_r)$, and $p^0_i$ is defined by
  \[ p^0_i = \begin{cases} \emptyset, & \text{if } \exists j < i : u_j = u_i, \\ \{ j \mid u_j = u_i \}, & \text{otherwise}, \end{cases} \]
  $i = 1,2,\ldots,r$. Note, that $u$ is a proper interpretation of $u$ by $\bar{p}^0$.
- $R'$ is the minimal set that contains the following productions.
1. For each production \((A, \vec{p}') \rightarrow (B, 12 \cdots (k-1)\delta(k+1) \cdots r) \in R\), and for each interpretation vector \(\vec{p}^1 = (p^1_1, p^1_2, \ldots, p^1_k) \in \vec{P}_r\), that conforms to \(\vec{p}'\), \(R'\) contains a set of productions \(\{((A, \vec{p}^1), S) \rightarrow (B, \vec{p}^{2,j}) \mid j \in T\}\),

where \(\vec{p}^{2,j}, j \in T, S\) and \(T\) are defined as follows.

Let \(i\) be such that \(k \in p^1_i\), and let \(\Phi = \{j \mid p^1_j = \emptyset\}\). For \(j = i\) \(\vec{p}^{2,j} = \vec{p}^1\). For the rest of \(j\)s in \(T\) \(\vec{p}^{2,j} = (p^{2,j}_1, p^{2,j}_2, \ldots, p^{2,j}_r)\) is defined by

\[
p^{2,j}_l = \begin{cases} 
  p^1_l \setminus \{k\}, & \text{if } l = i \\
  p^1_l \cup \{k\}, & \text{if } l = j, \\
  p^1_l, & \text{otherwise},
\end{cases}
\]

and

- if \(|p^1_i| = 1\), then \(S = \{i\}\) and \(T = \{1, 2, \ldots, r\} \setminus \Phi\);
- if \(|p^1_i| > 1\), i.e. \(\{k\} \subsetneq p^1_i\), then the set \(\Phi\) is not empty, and we define \(S = \{\min \Phi\}\) and \(T = (\{1, 2, \ldots, r\} \setminus \Phi) \cup S\).

The rational behind this definition is as follows. In the fist case \((p^1_i = \{k\})\) the register \(k\) of \(G\) is the only one which is interpreted by some register \(i\) of \(G'\), as follows from \(\vec{p}^1\). In this case the symbol, to which \(\delta\) is mapped to, is guessed into register \(i\).

In the second case \((|p^1_i| > 1)\), it follows from \(\vec{p}^1\) that the register \(i\) of \(G'\) that interprets register \(k\) of \(G\) also interprets other register of \(G\). In this case the symbol mapped to \(\delta\) is guessed into the register with the smallest index among those that do not interpret any of the registers of \(G\).

In both cases, the options for interpretation vectors of the derived variable correspond to all the possible ways in which the resulting assignment of \(G'\) may interpret the resulting assignment of \(G\). Namely, according to \(\vec{p}^{2,j}\), the register \(k\) of \(G\) is interpreted by the register \(j\) of \(G'\), that corresponds to the case when \(\delta\) is mapped to the new symbol \((j = i\) in the first case or \(j \in S\) in the second case), or to the symbol that already appeared in the registers of \(G\) and was interpreted by the \(j\)-th register if \(G'\). Therefore, for each resulting assignment of \(G'\), there is a vector among \(\vec{p}^{2,j}\) according to which it interprets properly the resulting assignment of \(G\).
2. For each production \((A, p') \rightarrow a_1a_2 \cdots a_n \in R\) and for each interpretation vector \(\bar{p}^1 = (p_1^1, p_2^1, \ldots, p_r^1)\) that conforms to \(p'\), \(R'\) contains the production \(((A, \bar{p}^1), \emptyset) \rightarrow a'_1a'_2 \cdots a'_n\), where the \(a'_i\)'s are defined as follows.

- If \(a_i \in \{1, 2, \ldots, r\}\), then \(a'_i = k\), where \(a_i \in p_k^1\).
- If \(a_i = (B, b_1b_2 \cdots b_r)\), then \(a'_i = (B, (p_1^2, p_2^2, \ldots, p_r^2))\), where \(p_k^2 = \{j : b_j \in p_k^1\}, k = 1, 2, \ldots, r\).

Proof of correctness

For the proof of the equality \(L(G) = L(G')\), we need the following definition.

Let \(X = X_1X_2 \cdots X_n \in (\Sigma \cup (V \times \Sigma^r))^*\) and let \(X' = X'_1X'_2 \cdots X'_n \in (\Sigma \cup (V^r \times \Sigma^r))^*\). We say that \(X'\) is a reflection of \(X\) if for each \(i = 1, 2, \ldots, n\), the following holds.

If \(X_i \in \Sigma\), then \(X'_i = X_i\), and if \(X_i = (A, w) \in V \times \Sigma^r\), \(w = w_1w_2 \cdots w_r\), then \(X'_i = ((A, \bar{p}), w')\), such that \(w'\) is an interpretation of \(w\) by \(\bar{p}\). If, in addition, \(w'\) is a proper interpretation of \(w\) by \(\bar{p}\), then \(X'\) is called a proper reflection of \(X\).

Lemma 44. Let \((A, w) \Rightarrow_G X\), and let \(((A, \bar{p}^1), w')\) be a proper reflection of \((A, w)\). Then there exists a proper reflection \(X'\) of \(X\) such that

\(((A, \bar{p}^1), w') \Rightarrow_{G'} X'\).

Note. The reason we need the reflections to be proper in this direction of the proof, is as follows. If \(((A, \bar{p}^1), w')\) is not a proper reflection of \((A, w)\), while \((A, w) \Rightarrow_G X\) by \((A, p') \rightarrow a\), it is possible that \(\bar{p}^1\) does not conform to \(p'\). Therefore, there might be no production in \(R'\) for the variable \((A, \bar{p}^1)\). We also want \(X'\) to be proper reflection of \(X\), to be able to consequently apply the lemma to the elements of \(X'\).

Proof of Lemma 44. Let \(w = w_1w_2 \cdots w_r\), \(w' = w'_1w'_2 \cdots w'_r\), and \(\bar{p}^1 = (p_1^1, p_2^1, \ldots, p_r^1)\). And assume that \((A, w) \Rightarrow_G X\) by the application of the production \((A, p') \rightarrow a\). It follows, that \(w\) conforms to \(p'\). Since \(((A, \bar{p}^1), w')\) is a proper reflection of \((A, w)\), \(w'\) properly interprets \(w\) by \(\bar{p}^1\). Therefore, \(\bar{p}^1\) conforms to \(p'\).

Since \(G\) is simple, either \(a\) is of the form \((B, 12 \cdots (k - 1)\delta(k + 1) \cdots r)\), or \(\delta\) does not appear in \(a\).
Let \( \mathbf{a} \) be of the form \((B, 12 \cdots (k - 1)\delta(k + 1) \cdots r)\). Then \( \mathbf{X} = (B, \mathbf{v}) \), where \( \mathbf{v} \) is obtained form \( \mathbf{w} \) by replacing \( w_k \) with some \( \sigma \in \Sigma \setminus \Theta \). Since \( \bar{\mathbf{p}}^1 \) conforms to \( \mathbf{p}' \), there exists a set of productions of \( G' \)

\[
\{(A, \bar{\mathbf{p}}^1), S) \to (B, \bar{\mathbf{p}}^{2,j}) \mid j \in T \},
\]
as described in the definition of \( R' \) on page 96. It suffices to show that one of these productions, when applied to \((A, \bar{\mathbf{p}}^1), \mathbf{w}\), produces \( \mathbf{X}' \) that is a proper reflection of \((B, \mathbf{v})\). That is \( \mathbf{X}' = ((B, \bar{\mathbf{p}}^2), \mathbf{v}') \), where \( \mathbf{v}' \) is a proper interpretation of \( \mathbf{v} \) by \( \bar{\mathbf{p}}^2 \).

The productions in the set differ only in interpretation vector \( \bar{\mathbf{p}}^{2,j} \). Therefore, it suffices to specify the \( j \) and show the desired property of the chosen production.

Let \( i, \Phi \) and \( S \) be as in the definition of \( R' \). We distinguish between the following three cases.

- If \( \sigma = w_k \), then \( j = i \).
- If \( \sigma \in [w] \setminus \{w_k\} \), then there is a \( j \neq i \) such that \( w'_j = \delta \), and \( p^1_j \neq \emptyset \).
  This is the \( j \) that we choose for the production.
- Otherwise, we choose \( j \) such that \( S = \{j\} \).

Now we show that indeed \((A, \bar{\mathbf{p}}^1), \mathbf{w}\) \(\Rightarrow\)\((B, \bar{\mathbf{p}}^2), \mathbf{v}'\) by the application of production \((A, \bar{\mathbf{p}}^1), S) \to (B, \bar{\mathbf{p}}^{2,j})\), where \( j \) is as chosen above and \( \mathbf{v}' \) properly interprets \( \mathbf{v} \) by \( \bar{\mathbf{p}}^{2,j} \). It suffices to show, that for each \( m \) there is an \( l \) such that \( v'_l = v_m \) and for each \( m' \), such that \( v_{m'} = v_m, m' \in p^{2,j}_l \).

- If \( \sigma = w_k \), then \( \mathbf{v} = \mathbf{w} \), and \( \bar{\mathbf{p}}^{2,j} = \bar{\mathbf{p}}^1 \).
  If \( p^1_j = \{k\} \), then \( S = \{i\} \) and \( \mathbf{v}' \) is obtained form \( \mathbf{w}' \) by replacing the contents of \( i \)-th register with some symbol in \( \Sigma \setminus \Theta \). When this symbol is \( \sigma \), we obtain \( \mathbf{v}' = \mathbf{w}' \). Since \( \mathbf{w}' \) is a proper interpretation of \( \mathbf{w} \) by \( \bar{\mathbf{p}}^1 \), it follows that \( \mathbf{v}' \) is a proper interpretation of \( \mathbf{v} \) by \( \bar{\mathbf{p}}^{2,j} \).
  If \( p^1_j \neq \{k\} \), then \( S = \{\min \Phi\} = \{j'\} \), and \( \mathbf{v}' \) is obtained form \( \mathbf{w}' \) by replacing the contents of \( j' \)-th register with some symbol in \( \Sigma \setminus \Theta \). Since \( \mathbf{v} = \mathbf{w}, \mathbf{v}' \) differs from \( \mathbf{w}' \) in the register \( j' \) only, for which \( p^1_{j'} = p^2_{j'} = \emptyset \), and \( \mathbf{w}' \) is a proper interpretation of \( \mathbf{w} \) by \( \bar{\mathbf{p}}^1 \), it again follows that \( \mathbf{v}' \) is a proper interpretation of \( \mathbf{v} \) by \( \bar{\mathbf{p}}^{2,j} \).

99
• If $\sigma \in [w] \setminus \{w_k\}$, then $j$ is such that $w'_j = \sigma$, and $p^1_j \neq \emptyset$. Since $w'$ is a proper interpretation of $w$ by $\overline{p}^1_j$, for all $m$, such that $w_m = \sigma$, $m \in p^1_j$. By definition of $\overline{p}^{2,j}$, it follows that $p^{2,j}_j = p^1_j \cup \{k\}$. Therefore, if $v'_j = w'_j$ (which is true as long as $j \notin S$), then for all $m$, such that $v_m = \sigma$, $m \in p^{2,j}_j$, as required.

If $p^1_j = \{k\}$, then $S = \{i\}$. Therefore, indeed $j \notin S$. Let $m$ be such that $v_m \neq \sigma$. Thus, $v_m = w_m$, and there is an $l$ such that $w'_l = v_m$, and for all $m'$ such that $v_m = v_{m'}(= w_{m'}) m' \in p^1_l$. In addition, $l \neq i$, since it follows from $p^1_i = \{k\}$ that $w_k$ was unique in $w$. Thus, $v_m = w_m \neq w_k = w'_l$. Also, $l \neq j$, for $v_m \neq \sigma$. Therefore, $v'_l = w'_l$, and, by the definition of $\overline{p}^{2,j}, p^{2,j}_l = p^1_l$. Thus, for all $m$ there is an $l$ such that $v'_l = v_m$ and for each $m'$, such that $v_{m'} = v_m$, $m' \in p^{2,j}_l$.

If $p^1_l \neq \{k\}$, then $S = \{\min \Phi\}$. Therefore, also $j \notin S$, for $p^1_j \neq \emptyset$. Let $m$ be such that $v_m \neq \sigma$. Thus, as before, $v_m = w_m$, and there is an $l$ such that $v'_l = v_m$, and for all $m'$ such that $v_m = v_{m'}(= w_{m'}) m' \in p^1_l$. Again, $l \neq j$, for $v_m \neq \sigma$. If $l \neq i$, then the argument from the previous paragraph holds for this $m$. Otherwise, $l = i$, and, by the definition of $\overline{p}^{2,j}, p^{2,j}_l = p^1_l \setminus \{k\}$. But, for all $m'$ such that $v_m = v_{m'}(= w_{m'})$, it holds $m' \neq k$, for $v_{m'} = v_m \neq \sigma$, and $v_k = \sigma$. Thus, for all $m'$ such that $v_m = v_{m'}(= w_{m'})$, $m' \in p^{2,j}_l$, and we conclude that the desired property holds for all $m$.

We are left with the case $\sigma \notin [w]$.

If $p^1_i = \{k\}$, then $S = \{i\}$, and $j = i$, which yields $\overline{p}^{2,j} = \overline{p}^1_i$. In this case $w_k$ is a unique symbol in $w$, and $v_k = \sigma$ is a unique symbol in $v$. Thus, when the contents of the $i$-th register of $w'$, which interprets $w_k$, is replaced with $\sigma$, it is clear that we obtain $v'$, that is a proper interpretation of $v$ by $\overline{p}^1_j(=\overline{p}^{2,j})$.

If $p^1_j \neq \{k\}$, then $S = \{\min \Phi\} = \{j'\}$, and $j = j'$. By the definition of $\overline{p}^{2,j}, p^{2,j}_j = p^1_j \cup \{k\} = \{k\}$. Thus, when the symbol placed into $j$-th register of $w'$ is $\sigma$, we have that for all $m$ such $v_m = \sigma$ (the only such $m$ is $k$) $m \in p^{2,j}_j$, and $v'_j = \sigma$. For the other $m$, $v_m = w_m$, and there is an $l$ such that $w'_l = v_m$, and for all $m'$ such that $v_{m'} = v_m(= w_m), m' \in p^1_l$. Since $p^1_l \neq \emptyset, l \notin S$. Therefore, $w'_l = v'_l$. If $l \neq i$, then $p^1_l = p^{2,j}_l$. Thus, for all $m'$ such that $v_{m'} = v_m$, $m' \in p^{2,j}_l$. Otherwise, $p^{2,j}_l = p^1_l \setminus \{k\}$. In addition, for all $m'$ such that $v_{m'} = v_m, m' \neq k$, because $v_m \neq \sigma$
(the case which we treated separately). Therefore, \( m' \in \mathcal{p}^{2,j} \). Thus, the required property holds for all \( m \).

We have shown, that if \( a \) is of the form \((B, 12 \cdots (k-1)\delta(k+1) \cdots r)\), then there exists a production \(((A, \bar{p}^1), S) \rightarrow (B, \bar{p}^{2,j})\) in \( R' \), by which \(((A, \bar{p}^1), w') \Rightarrow X'\), where \( X' \) is a proper reflection of \( X \).

Now, let us assume that \( \delta \) does not appear in \( a = a_1a_2 \cdots a_n \). Then, by the definition of \( R' \), it contains a production \(((A, \bar{p}^1), \emptyset) \rightarrow a'_1a'_2 \cdots a'_n \), where \( a'_r \)'s are defined as follows.

- If \( a_r \in \{1, 2, \ldots, r\} \), then \( a'_r = k \), where \( a_r \in p^k \).
- If \( a_r = (B, b_1b_2 \cdots b_r) \), then \( a'_r = (B, (p_1^r, p_2^r, \ldots, p_j^r)) \), where \( p_k^j = \{j: b_j \in p^k_1\} \), \( k = 1, 2, \ldots, r \).

We shall show, that if \(((A, \bar{p}^1), w') \Rightarrow X'\) by the above production, then \( X' \) is a proper reflection of \( X \). That is, if \( X = X_1X_2 \cdots X_n \), then \( X' = X'_1X'_2 \cdots X'_n \), and

- if \( X_i \in \Sigma \), then \( X'_i = X_i \),
- otherwise, \( X_i = (B, v), X'_i = (B, \overline{p}^2), v' \), and \( v' \) is a proper interpretation of \( v \) by \( \bar{p}^2 \).

Indeed, if \( X_i = \sigma \in \Sigma \), then \( a'_i = j \in \{1, 2, \ldots, r\} \). Thus, \( a'_i = k \), where \( j \in p^k \). Since \( v' \) is a proper interpretation of \( v \) by \( \bar{p}^1 \), if follows that \( w' \) is a proper interpretation of \( w \) by \( \bar{p}^1 \).

For each \( m \), \( v_m = w_{bm} \). Therefore, there exists an \( l \) such that \( w'_l = v_m \), and \( b_m \in p^l \). By the definition, \( p^l_1 = \{j: b_j \in p^l_1\} \). Therefore, \( m \in p^l_1 \). Let \( m' \) be such that \( v_m = v_{m'} \). Clearly, \( v_{m'} = w_{bm'} = w_{bm} \), and since \( v' \) properly interprets \( v \) by \( \bar{p}^1 \), \( b_{m'} \in p^l_1 \). Thus, \( m' \in p^l_2 \).

**Lemma 45.** If \(((A, \bar{p}^1), w') \Rightarrow X'\) for some reflections \(((A, \bar{p}^1), w') \) of \((A, w)\) and \( X' \) of \( X \), then \((A, w) \Rightarrow \overline{G} X \).
Proof. Let \(((A, \vec{p}^1), w') \Rightarrow G' X'\) by the production \(((A, \vec{p}), S) \rightarrow a'\) in \(R'\). From the definition of \(R'\) it follows, that there is a production \((A, \vec{p}') \rightarrow a\) in \(R\), from which \(((A, \vec{p}), S) \rightarrow a'\) was constructed. That is, \(a'\) is obtained from \(a\) as described in the definition of \(R'\), and \(p'\) is an ECS such that \(\vec{p}^1\) conforms to \(p'\). Since \(((A, \vec{p}^1), w')\) is a reflection of \((A, w)\), it follows that \(w'\) is an interpretation of \(w\) by \(p\). Therefore, \(w\) also conforms to \(p'\). Thus, the production \((A, p') \rightarrow a\) may be applied to \((A, w)\). It remains to prove that \(a\) is such that the result of this application yields \(X\).

Since \(((A, \vec{p}), S) \rightarrow a'\) was constructed from \((A, p') \rightarrow a\), it follows that either \(S \neq \emptyset\), in which case \(a\) is of the form \((B, 12 \cdots (k-1)\delta(k+1) \cdots r)\), or \(S = \emptyset\), in which case \(\delta\) does not appear in \(a\).

Let \(S \neq \emptyset\) and \(a = (B, 12 \cdots (k-1)\delta(k+1) \cdots r)\) for some \(k\). Then, \(a' = (B, \vec{p}^{2j})\) for some \(j\), \(X' = ((B, \vec{p}^{2j}), v')\), \(X = (B, v)\), and \(j\) is such that \(v'\) is an interpretation of \(v\) by \(\vec{p}^{2j}\). We need to show that \(v\) is indeed obtained from \(w\) by replacing \(w_k\) with some \(\sigma \in \Sigma \setminus \Theta\). To do so, \(k\) and \(\sigma\) must be determined using the relationships between the given \(S\), \(\vec{p}^1\) and \(\vec{p}^{2j}\). We omit this part, since it is similar to the proof of the corresponding case in Lemma 44, read backwards.

Finally, let \(S = \emptyset\). Thus, \(\delta\) does not appear in \(a\), and if \(a' = a'_1a'_2 \cdots a'_n\), then \(a = a_1a_2 \cdots a_n\), where each \(a'_i\) was obtained from \(a_i\) as described in the definition of \(R'\), part two. Similarly to the corresponding part of proof of Lemma 44, when read backwards, it can be shown, that if \(X'\) is obtained from the application of \(((A, \vec{p}), S) \rightarrow a'\) to \(((A, \vec{p}^1), w')\), and \(X'\) is a reflection of \(X\), then \(X\) is obtained form the application of \((A, p') \rightarrow a\) to \((A, w)\).

The inclusion \(L(G) \subseteq L(G')\) follows from Lemma 44, by which each \(G\)-derivation can be translated into a \(G'\)-derivation, step by step. Conversely, the inclusion \(L(G') \subseteq L(G)\) follows from Lemma 45, by which each \(G'\)-derivation can be transformed into a \(G\)-derivation.
Appendix D

Corrections to the final proof of Theorem 2 in [1, Section 7]

Our construction in Section 4.5 resembles the construction of an $M$-grammar simulating an infinite alphabet push-down automaton, described in [1, Section 7]. The latter, though, contains a number of inaccuracies, that we correct here.

The following corrections are to the definition of grammar productions, group 2.

D.1 The case $n = 0$

The main problem in the definition is that it fails to distinguish the case $n = 0$, which significantly differs from the general case.

First, in this case the production has to have the form of $((q, q_1), p) \rightarrow a$, instead of $((q, q_{n+1}), p) \rightarrow a$, which is probably a misprint. However, note, that this form cannot be obtained as a special case of the description for an arbitrary $n$ because, even if we consider the list $q_2, \ldots, q_{n+1}$ (which, apparently, also contains a misprint) empty, when $n = 0$, it does not yield that $q_{n+1}$ in the left-hand side must be substituted with $q_1$.

Secondly, as opposed to the general case, where (almost) any partition $p \in P_{2r+1}$ may suit, the productions for $n = 0$ must be included into $R$ only for the partitions of a very special form. Namely, $p$ must satisfy the following condition:

- if $\rho(q) = i$, then for each $l = 1, 2, \ldots, r$, with the exception of $l = i$, $p$
must contain \( p \) such that \( l, r + 1 + l \in p \), and for \( l = i \), \( p \) must contain \( p \) such that \( r + 1, r + 1 + l \in p \).

- if \( \rho(q) \neq i \), then for each \( l = 1, 2, \ldots, r \), with the exception of \( l = \rho(q) \), \( p \) must contain \( p \) such that \( l, r + 1 + l \in p \), while \( p \) that contains \( i, r + 1 + i \) must in addition contain \( r + 1 \).

To understand the reason for this condition, let us recall, that the semantics of this production is to simulate the transition of the automaton from state \( q \) to state \( q_1 \), while “popping” a symbol from the stack. The semantics of the assignment accompanying the variable \((p, p_1)\) in the derivation prior to applying the production in question is as follows. The first and the last \( r \) registers simulate the registers of the automaton before and after the transition, corresponding to the two adjacent states in the computation, \( q \) and \( q_1 \), respectively. Thus, the only difference between the content of the first and the last \( r \) registers has to be brought by the reassignment \( \rho(q) \). In addition, the symbol in the register \( r + 1 \), simulating the head of the stack, has to be the same as the symbol in the register simulating the \( i \)th register of the automaton after the transition, i. e., the \((r+1+i)\)th register. This is exactly what the above condition on \( p \) states.

Finally, \( a \) must also be defined differently in this case. Whereas in the general case \( a \) may be in \( \Delta \), in case \( n = 0 \) all the parameters of the transition simulated by the production in question are already known (namely, the assignments before and after the transition). Thus, we do not need to employ the “guessing” capability of the grammar, and \( a \) must merely be generated from the register, which simulates the \( k \)th register of the automaton after possible reassignment, namely, the \((r + 1 + k)\)th register (unless \( k = \epsilon \), in which case \( a \) also must be defined as \( \epsilon \)).

### D.2 The general case \((n > 0)\)

In the definition of \( b \), the case of \( b \in \Delta \) is originally described by the conditions \( i \neq \rho(q) \) and \( \{i, r + 1\} \in p \). The second condition is, in fact, too strict. It prevents any of the last \( r \) registers from containing the symbol in the \((r + 1)\)st register, which is and must be allowed for the above case. Instead, it must be phrased as “there is a \( p \in p \) such that \( i, r + 1 \in p \)”.

Moreover, the above condition on \( p \) must not be a case condition in the definition of \( b \), but rather a condition on including the production with such
\( p \) into \( R \). Namely, if \( i \neq \rho(q) \), and the above condition on \( p \) does not hold, the production with such \( p \) is illegal, because it simulates a transition in which the symbol on the top of the stack is not equal to the contents of the \( i \)th register, that is not reassigned.

The definition of \( b \) itself is not specific enough. Rather than merely stating that \( b \in \Delta \), it needs to be given a specific value, say, \( \delta_0 \), to distinguish it from other members of \( \Delta \) used in the production.

To summarize, the definition of \( b \) must be as follows:

- if \( i = \rho(q) \), then \( b = r + 1 \) (the case when the register \( i \) is compared to the stack successfully after being reassigned),
- if \( i \neq \rho(q) \) and there is a \( p \in p \) such that \( \{i, r + 1\} \subseteq p \), then \( b = \delta_0 \),
- otherwise (i.e. \( i \neq \rho(q) \) and there is no \( p \in p \) such that \( \{i, r + 1\} \subseteq p \)), the production is not included into \( R \).

The definition of \( a \) for the case \( \rho(q) = k \) is also incorrect. It is defined as some \( \delta \), whereas it must be defined as \( b \). This is because it is the case that corresponds to the situation when the automaton guesses correctly a letter from the input. This fact is reflected by \( b \) in the resulting assignment.

### D.3 The conclusion

The definition of the grammar is followed by stating an immediate conclusion regarding the equivalence of a step in the automaton’s computation and a step in the grammar’s derivation, which exists for some \( q_2, q_3, \ldots, q_{n+1} \in Q \) and some \( w_2, w_3, \ldots, w_{n+1} \in \Sigma^\ast \). Although, formally, it is correct (due to the definition of \( R \), if a production exists for some set of registers, it exists for any, and the same with the assignments), it is essential for the application of the conclusion in the proof, that the grammar’s derivation exists for any \( n \) states and assignments, which is exactly what is obtained by guessing these assignments in the production rules and including a production for each set of \( n \) states. Leaving these options open, we allow the grammar to “guess” correctly the path that the automaton will take while processing the symbols pushed into the stack. Narrowing the options down to one will take place only after the consequent application of the conclusion to the automaton’s whole path.

Also, the conclusion for the case \( n = 0 \) must be stated separately.
Bibliography


שתי החלוקים والاשכולה ניצבים לפנים עםتحرישה בוה.

בניסו הולו
FSUBA

בניסו הולו
FSUBA

DR-UBPDA

DR-UBPDA

משהוerner-חקש
The work of [author] is presented in this M.Sc. thesis. The aim of the research is to develop a \textit{pushdown automaton with deterministic reassignment} (DR-UBPDA) approach for solving the synchronization problem. The work is divided into two main parts:

1. \textit{Unification based pushdown automaton with non-deterministic reassignment} (NR-UBPDA)

2. \textit{Unification based pushdown automaton with deterministic reassignment} (DR-UBPDA)

The main contributions of the work include:

- The development of a new \textit{unification based pushdown automaton with deterministic reassignment} (DR-UBPDA).
- The use of a new \textit{finite-state unification algorithm}.
- The development of a new \textit{unification based pushdown automaton with non-deterministic reassignment} (NR-UBPDA).
- The use of a new \textit{finite-state unification algorithm}.

The work is expected to have applications in various fields, including computer science, artificial intelligence, and formal language theory.


**Theorem**

Theorem 1: The emptiness problem for finite automata is decidable.

**Proof**

The proof of the theorem is based on the construction of a non-deterministic finite automaton (NFA) that recognizes the language of all words that are not in the language of the given finite automaton. This NFA can be constructed by adding a new state for each state of the given automaton, and for each transition in the given automaton, adding transitions to the new state that simulate the transition in the original automaton.

Let $\mathcal{A}$ be a finite automaton with alphabet $\Sigma$ and set of states $Q$. We construct an NFA $\mathcal{A}'$ as follows:

1. For each state $q \in Q$, add a new state $q'$.
2. For each transition $q \xrightarrow{a} q'$ in $\mathcal{A}$, add a new transition $q \xrightarrow{a} q'$ in $\mathcal{A}'$.
3. For each transition $q \xrightarrow{a} q'$ in $\mathcal{A}$, add a new transition $q \xrightarrow{a} q'$ in $\mathcal{A}'$.
4. Add a new start state $s'$ with transitions $s' \xrightarrow{\varepsilon} q'$ for each $q \in Q$.
5. Add a new accept state $a'$ with transitions $q' \xrightarrow{\varepsilon} a'$ for each $q \in Q$.

It can be shown that $\mathcal{A}'$ accepts the language of all words that are not in the language of $\mathcal{A}$. Therefore, we can determine whether $\mathcal{A}'$ accepts an empty language by checking if $\mathcal{A}'$ accepts any word.

This completes the proof of Theorem 1.

**References**


המודל ה_anything is an example of a model with a parameter of higher-order clique.

אך מודל זה לא פمساب על התוכן המסופט של המודלPeriphורי.
אוטומטים מח בכלל 알פבית הפניים: יישוט שוניות והשוואת תופעות

שם מח

לשם מח

רבים

וליה דובב

מחולה לאמיתיות – מרכז טכנולוגיות לישראל

אדר א' תשס"ח חפencers, 2008
viarווטם מחקרים מעל אלפבית אלפאפורומס: גישה שונת והשוואת תצלומיות

יוליה דובוב